ON THE IMPROVEMENT OF SEARCH ENGINE USING SEMANTIC WEB: CASE STUDY OF HUA HIN TOURISM INFORMATION SYSTEM

Chidchanok Choksuchat[1] and **Chantana Chantrapornchai**[2], Non-members

ABSTRACT

This paper describes how to improve the search engine using the ontology for the semantic web technology, the case study on a prototype Hua Hin tourism web page. The methodology contains two parts. First, we describe a way to compute query complexity between the relational database (RDB) and web ontology language (OWL). We present the original RDB, convert to the OWL, and consider the query complexity with the searching time of the prototype system. A number of facts in the ontology, and a number of conjuncts in the conjunctive queries are considered as parameters. The goal is to improve the search engine by reducing the query complexity. Second, the SPARQL queries in web application are considered from the classification and realization of the reasoner. We study from the existing ontology, increasing OWL DL rules, and consider the expressivity. We then compare the searching time between top-down and bottom-up approaches. The advantages are to select the appropriate inference problems for query response through the web application. The prototype Hua Hin semantic web is used to present our methodology throughout the paper.

Keywords: OWL DL, SPARQL, Semantic Web, Expressivity, Ontology

1. INTRODUCTION

Tourism has become one of the world's largest industries. Nowadays, the use of tourism information technology's services reaches 50 percent of all services on the Internet [1].

The information about the tourism industry is mainly distributed on the Internet and in many forms. As a tourist, to collect the information before traveling may be tedious and difficult. Users may input many kinds of queries as Figure 1. Then, the search results may be significant and varied. The users need to browse to the obtained information to select their interesting ones. Moreover, only a human user can understand all of the related results. Hence, information management, interpretation and processing take lots of time. This problem is because the traditional search engine cannot interpret words or phrases. Therefore, the principles of the semantic web play a role to solve the problems. With the semantic web, the meaning of keywords can be stored as in the layer cake. Besides, the inference algorithm can be used to infer related information.

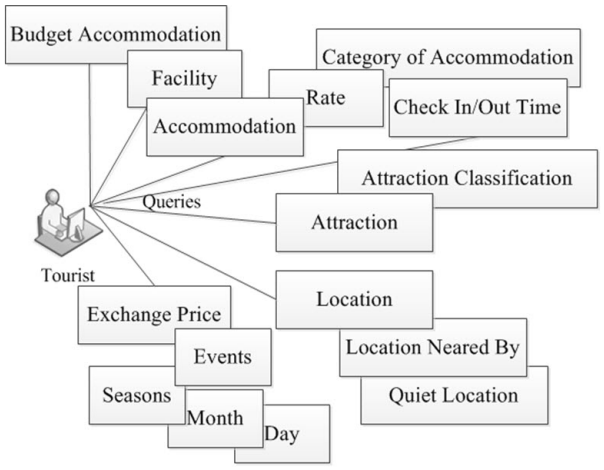

Fig.1: *Many concepts of the tourists' queries*

There are many works on the tourism search engine. For example, Harmo-TEN is the European tourism communication online. It helped exploit the semantic web technologies for harmonizing e-markets [2]. G.Sudha Sadasivam et al.'s work is the India tourism suggestion system. It focused on ontology based information retrieval for e-tourism [3]. AcontoWeb is an Australia e-tourism using the semantic web. It includes the integration and utilization tool [4]. These works are similar in such a way that they solve the tourism problem in the specific domain.

In this research, we focus on the case study of Hua Hin tourism as a prototype semantic web. First, we design the tourism ontology of the domain. It is based on http://www.huahin.go.th version 2009's database. We study the measurement of the quality of the ontology design in terms of complexity as it is related to the speed of data processing. Secondly, we also in-

Manuscript received on September 1, 2011 ; revised on January 31, 2012.

[1,2] The authors are with Department of Computing, Faculty of Science, Silpakorn University, Nakhon Pathom, Thailand., E-mail: cchoksuchat@hotmail.com and ctana@su.ac.th

vestigate the inference process as it is related to the query processing as well. The experimental results on the bottom up and top down search approaches are demonstrated as well as the semantic web application prototype is created. Our prototype web application can be extended to manage the other tourism ontology, annotating webpage, searching on the others module and benchmarking the conjunctive query concept.

Our work has the contributions in two folds.

1. The query complexity can consider from the data complexity view. The OWL reduces the fact terms, joins, and query complexity of the original RDB. This complements the field with the tourism case through the ontology driven.

2. It provides the guidance to the developers to apply the proper inference search technique (either top down or bottom up) to the semantic web search engine.

The rest of the paper is structured as follows. Firstly, we give some backgrounds as well as some related works in Section 2.Section 3 describes our methodology. Section 4 presents the results and analyzes the experiments. Section 5 presents the prototype semantic web application. Section 6 concludes the paper and outlines our next steps.

2. BACKGROUNDS AND RELATED WORK

This section, we describe the background of conventional and semantic web search engine in section 2.1. Section 2.2 is referred to the related works that similar to our work and support our contributions. Section 2.3 is represented the definitions and theorem that support our first goal. Section 2.4 and 2.5 are supports the second aim that ontology's inference can help semantic web search engine.

2.1 Conventional and Semantic Web Search Engine

Today, popular search engine is Google (www.google.com) that uses the keyword indexing capability. The main problem is that search results are imprecise beyond the fact that a number of webs that always increases and the various data are collected along with various websites e.g. image and text in the difference database pattern: difference syntax, difference structure and difference semantic. Therefore, integrating such information takes more time even in the same search topic.

One of the search engines using the semantic web technology is Swoogle (http://swoogle.umbc.edu). It is developed by Computer Science and Electrical Engineering Department of the University of Maryland Baltimore County [5] and the latest version is "Swoogle2007". However, the results of the semantic web search are XML schema, RDF, RDFS, NTRIPLE, or OWL files. The HTML browser cannot represent the XML tags with a friendly GUI. In 2006, Berners-Lee presented *Tabulator* , the Mozilla Firefox's plugin [6]. It is better than the other previous semantic web browser. For example, Piggybank [7] relies on the screen scrapers that collect data, interpret and analyze the websites. However, the user has to download and install the program on their machine. Meanwhile Tabulator can explore and analyze *LinkedData* [8] on the semantic web.

2.2 Semantic Web Search Engine on Tourism Domain

Harmo-TEN was formally known as *Harmonise* [2]. It is a European online communication that supports the tourism information system. The important concept is the minimum ontology, convenient access and simple plan between standard and non-standard data. The Harmonisation process was based on the semantic mapping between the user's data and ontology concepts. However, the study raised the issues that are originated in developing the business plan and clashed mapping problem.

The e-tourism in the semantic web that is closely related to our contributions as follows.

1. *Ontology Based Information Retrieval for E-Tourism* by G.S. Sadasivam et al. [3] that compared between top-down and bottom-up approaches in only overview. They did not separate to test in each case of OWL DL syntax.

2. *Acontoweb* by Abraham [4] that is the Australia e-tourism using the semantic Web. This work was referenced much by others. However, they did not test the rule of OWL DL and did not test the searching time. In addition, Abraham's surveyed from the user concerned tourism industry. The most of them did not design to update the web infrastructure to be a semantic web. However, after T. Burners-Lee recommended the "Linked Data" [8] , the ontology and triple set exchange data were manipulated globally.

2.3 Query Complexity

First we give some definitions and the theorems as they are related to our analysis as follows.

Definition 2.3.1: A terminological knowledge base. Let $\mathcal{T}$ be a triple, then

$$\mathcal{T} = [\mathcal{C}, \mathcal{R}, \mathcal{I}] \tag{1}$$

where $\mathcal{C}$ is a set of class definitions, $\mathcal{R}$ is a set of relation definitions and $\mathcal{I}$ is a set of object definitions. We can apply Equation (1) to the ontology design as in following equation.

$$\mathcal{T} = [\mathcal{CN}, \mathcal{RN}, \mathcal{IN}] \tag{2}$$

Here $\mathcal{CN}$ is the set of all named classes defined in $\mathcal{C}, \mathcal{RN}$ is the set of all relation names and $\mathcal{IN}$ is the set of all object names occurring in the knowledge base.

Definition 2.3.2:The terminological queries.

Let $\mathcal{V}$ be a set of variables disjoint from $\mathcal{IN}$; then a conjunctive query $\mathcal{Q}$ over a knowledge base $\mathcal{T}$ is an expression of the form.

$$Q_{li} \wedge \ldots \wedge Q_{mi} \tag{3}$$

where Q_i are query terms of the form $X : C$ or (x, y):r such that x, y $\in \mathcal{V} \cup \mathcal{IN}$, $\mathcal{C} \in \mathcal{CN}$ and $\mathcal{R} \in \mathcal{RN}$. Then, we apply Equation (3) to the query expressions in the Casual Form with AND operator.

Regarding the Acontoweb [4], Abraham presented the query evaluation as follows. The analysis of query complexity from the business view should not use much more of the mathematic formula. Thus, the advantage of semantic web and capability of OWL reasoning in Description Logics (DL) knowledge base is $< T, A >$. T is a Terminological Box (TBox). It constitutes the vocabulary of an application domain. A is an Assertional Box (ABox). It contains real world assertions of individuals (instances) in vocabulary terms. Vardi's complexity uses data in ABox [9] because it is similar to the relational database case. It is important for the query evaluation model that was designed from a knowledge base. It means the logical query complexity as evaluating a particular data model itself, rather than individual query representational languages.

Three ways measure the complexity of queries for evaluating queries in a specific language over a database of Vardi including. 1). One can fix a specific query in the language and study the complexity of applying this query to arbitrary databases. The complexity is then given as functions of the size of the databases. They called this complexity as data complexity. 2). One can fix a specific database and study the complexity of applying queries represented by arbitrary expressions in a language. The complexity is then given as a function of the length of the expressions. They called this complexity, expression complexity. 3). One can study the complexity of applying queries represented by arbitrary expressions in the language to arbitrary databases. The combined complexity is then given as a function of the combined size of the expressions and the databases. We call this complexity, combined complexity. It turns out that combined complexity is pretty closed to the expression complexity, and for this reason we concentrate in this paper upon the data and expression complexity.

In this research, the expression complexity or query complexity is evaluated. The size of the instances was given, when measuring the complexity of the query answering and generally reasoning. The most important parameter is the size of the data. It is similar to compute data complexity of query answering. Abraham used the following definition as a basis for the complexity measurement [4]. As an example, they considered the language of first-order logic.

Definition 2.3.3

Let φ be a sentence of size s (a sentence represents a query). φ has at most s variables. In order to evaluate φ on a database of size n, it suffices to cycle through at most n^s possible assignments of values from the database to the variables.

This can be done in space $O(logn)$. Thus the set of all databases satisfying φ is in LOGSPACE. Abraham derived the query complexity defined by the formal logic as:

$$\exists\varphi((\varphi \rightarrow s) \wedge (s \equiv n^s)) \tag{4}$$

We explain the equation (4) as follows: for some sentence φ, the sentence has size s, and s equals the number of possible values to variable assignments from the database that may be assigned to φ.

Complexity of φ can, therefore, be expressed as the function:

$$s \equiv n^s \tag{5}$$

We explain Equation (5) as follows: the value of s (query complexity) can then be obtained by simply calculating: n^s.

Abraham proposed a type of measures. A more in-depth mathematical analysis was required to consider the query optimization covered by Calvanese et al.[10]. They characterized the LOGSPACE boundary of the problem. For example, finding maximally expressive DLs for query answering can be done in LOGSPACE. Abraham did not analyze it. Here, we analyze Calvanese's theorem as follows [10].

Theorem 2.3.4: Query answering in $DL-Lite_{\mathcal{R},\prod}$ is First Order Logic (FOL) reducible and therefore is in LOGSPACE with respect to data complexity.

It takes an advantage of DBMS techniques for the data representations. It means if there are the ABox assertions, then the query is reformulated into the SQL. Notably, this case mentions the data complexity of conjunctive query over ontologies. That means the one of FOL queries over databases. They had presented the first fundamental results on the data complexity of query answering in DLs. The complexity is respected to only the size of the ABox. The issues are based on the query that is no longer expressible as an FOL formula. Hence, it is based on an SQL query over the data.

The first fundamental results on the data complexity are the complexity with respect to only ABox size. The data complexity is the result of query answering in DLs. Table 1 summarizes the known complexity results for OWL DL, DL-Lite, and RDF Schema (RDFS) from W3C. Notably, whenever the complexity for a problem is described as Open*, it is meant that still it is an open question; if the star (*) is omitted, then the problem is decidable, however, the precise complexity bounds have not yet been established. If a problem is noted as trivial, it is meant

Table 1: *The total number of operations (per bit) of each function.*

Language	Problems	Complexity[11]	
		Data	Query
OWL DL	R1	Open(NP-Hard)	Not Applicable
	R2	Open(NP-Hard)	Not Applicable
	R3	*Open∗*	*Open∗*
DL-Lite	R1	LOGSPACE	Not Applicable
	R2	LOGSPACE	Not Applicable
	R3	*LOGSPACE*	*NP − complete*
RDFS	R1	Trivial	Not Applicable
	R2	LOGSPACE	Not Applicable
	R3	*LOGSPACE*	*Open*

that the language is not expressive enough for allowing different possible answers to the problem, e.g., every RDFS ontology is known to be consistent [11].

In Table 1, R1 means reasoning problems such as ontology consistency, concept satisfiability, concept subsumption, and instance checking. R2 means reasoning problems that include concept subsumption, and instance checking. R3 means reasoning problems about conjunctive query answering. Therefore the parameters related to the reasoning problem can be summarized as follows: 1) the data complexity: the complexity measured with respect to the number of facts in the ontology, 2) the query complexity: the complexity measured with respect to the number of conjuncts in the conjunctive query.

The fact that the data complexity stays LOGSPACE means that one can exploit a relational database technology for instance checking and conjunctive query answering. The fact that the data complexity goes beyond LOGSPACE means that query answering and instance checking require more powerful engines than the ones provided by relational database technologies. The relationship between the fragments of web ontology language in this research can be shown in Figure 2.

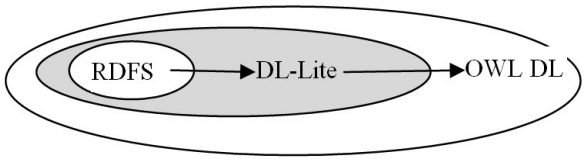

Fig.2: *Relationship between the fragments of OWL DL.*

Figure 2 shows the relationship between the different languages concerned in this research, including OWL DL, DL-Lite and RDFS. From this condition, let two languages L1, L2 be connected by an arrow, i.e, L1→ L2 if L1 is polynomially reducible to L2. In most of the cases the reduction is trivial, since L1 is just a syntactic fragment of L2 and hence every syntactically valid ontologies written in L1 is also valid in L2 [11].

2.4 Expressivity

For modelling such complex knowledge, expressive representation languages based on the formal logic are commonly used. This also allows us to do logical reasoning on the knowledge, and thereby enables the access to knowledge, which is only implicitly modelled [12].

The expressive meaning of $\mathcal{SHOIN}^{(\mathcal{D})}$ that covers the OWL DL is summarized as follows [13]. All languages constructed from $\mathcal{ALC}$ are defined formally by the syntax rules as is the rule (6) of the complex class.

$$C, D ::= A|\top|\bot|\neg C|C \sqcap D|C \sqcup D|\forall R.C|\exists R.C \quad (6)$$

Let A be an atomic class, i.e. a named class and let R be an abstract role. Then class expressions C, D are constructed using the rule in Equation (6). Then, OWL DL language constructs can be represented in $\mathcal{ALC}$ including classes, roles, individuals, class membership and role instances, owl:Thing and owl:Nothing, class equivalence, class conclusion, negation of classes, disjointness, conjunction, disjunction, role restriction using owl:all-ValuesFrom and owl:someValuesFrom, rdfs:domain and rdfs:range.

$\mathcal{SHOIN}^{(\mathcal{D})}$ covers all OWL DL expressive meanings and also has the equality and inequality between individuals, closed classes, cardinality restrictions, role inclusion axioms and role equivalences, inverse roles, inverse functionality, functionality, transitivity, symmetry of roles and data types [14]. As mentioned above, OWL DL is very closely related to $\mathcal{SHOIN}^{(\mathcal{D})}$ [14], which explains the constructor by certain letters. The letter S stands for $\mathcal{ALC}$ (6) plus role transitivity. $\mathcal{H}$ stands for role hierarchies. O stands for the nominal. $\mathcal{I}$ stands for the inverse roles. $\mathcal{N}$ stands for cardinality restrictions and $\mathcal{D}$ stands for data types.

We can measure the DL expressivity and all the metric values are calculated by Protégé for OWL ontologies, which is the most widely used tool, for creating or modifying the ontology [15] . Metrics are classified into six categories. The first one is related to general metrics such as counters for classes, object properties, data type properties and individuals. The second category is related to class axioms and includes counters for subclass axioms, equivalent class axioms, and disjoint class axioms.

The third category includes counters for object property axioms. These counters are a total of subobject properties; equivalent, inverse, disjoint, transitive, functional, inverse functional, symmetric, antisymmetric, reflexive and irreflexive object properties. Besides this, the object property domain and range counters are also included. The fourth category is dedicated to data type property counters. This category includes total values for subdata type properties, equivalent, disjoint and functional data type properties as well as counters for the data properties domain

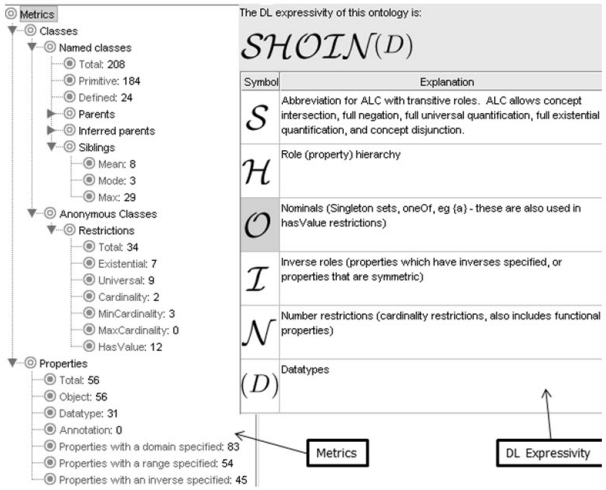

Fig.3: *Protégé metrics screen snapshot that shows all the metric values calculated*

and range. The fifth category is focused on individuals. It defines counters for class assertions, object and data type property assertions, negative object and negative data type assertions and same or different individual axioms. Finally, the last category involves annotation axioms and defines just two metrics, entity annotation axioms count and axiom annotation axioms count. All these metrics represent simple counters for the items in the ontology.

2.5 Top Down VS Bottom Up Search

G.S. Sadasivam et al. [3] mentioned in the tourism recommendation system, whenever the end user selects the instances, then ontology will be traversed and meaningful results will be displayed based on the top down approach. If the end user specifies the name that may belong to some named class or superclass then ontology will be traversed using the bottom up approach. Andrew and Edward [16] presented two methods for constructing a hierarchical of gene ontology. Then, we can explain in the same way with the tourism ontology as in Figure 4.

Figure 4 presents the comparison of the top-down and bottom-up methods of describing tourism ontology. In a top-down approach (Figure 4(a)), a hierarchical ontology describes different aspects of tourism classes as in the box numbers 1 to 6. Then the reasoner assigns instances as the circles labelled A-C to the categories according to asserted conditions "Test A-C". Although we depict here a method in which instances are placed into a single category, instances can be associated with multiple categories where different associations can be weighted different ways. Meanwhile, in the bottom-up approach, instances are imported from the diverse data sets and integrated into a complex network of instances as highlighted boxes in Figure 4(b). The dashed rectangle regions 1-4 define processes and the lines between the rect-

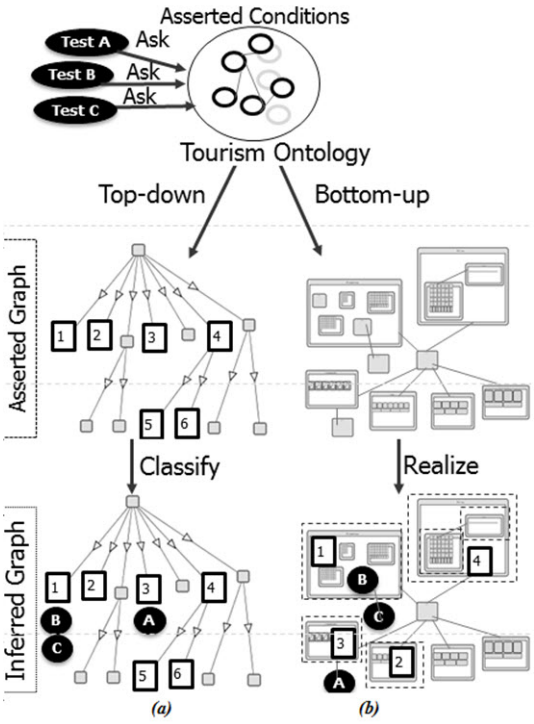

Fig.4: *Comparison of (a) top-down and (b) bottom-up methods.*

angle and the dash rectangles define the associations between instances and processes.

In addition, Chen [17] described a top-down tree problem related to the number of nodes. The searching by the top-down approach takes less computation time than that of the bottom-up approach. However, this is not always the case in our web application. G.S. Sadasivam et al. [3] proposed the top down and bottom up approaches to extract information from ontology. When no details of instances are provided for the bottom up approach, top down approach is suitable. Thus the trade-off between top down and bottom up approaches is not only based on the performance but also on their applicability. However, their works did not consider in OWL DL rules.

3. METHODOLOGY

In this section, we first describe our methodology from setting data until they are converted to ontology. Then we focus on the following:

1) Measure complexity by comparing the SQL and SPARQL query complexity across the same domain.

2) Measure expressivity and test SPARQL query performance against of OWL data.

3) Look for the appropriate method to set query answering for the user. Ontology classification and realization are considered.

4) Study the search engine from a specific tourism domain and how it could be extended to be the ontology search engine.

5) Creating the application of such a search engine using the real data from Hua Hin District.

Step 1) and 2) support our first contribution and Steps 3), 4) and 5) support the second one mentioned in Section 1.

In the following, we describe the data application setup procedures first. Then, we present the methodology for the procedures mentioned above.

3.1 Dataset of Original System

The original system is a web application with a relational database (RDB). In the RDB, tables are used. Normalization is usually required to minimize redundancy. However, it is not always the case that the resulting normalized tables are correct.

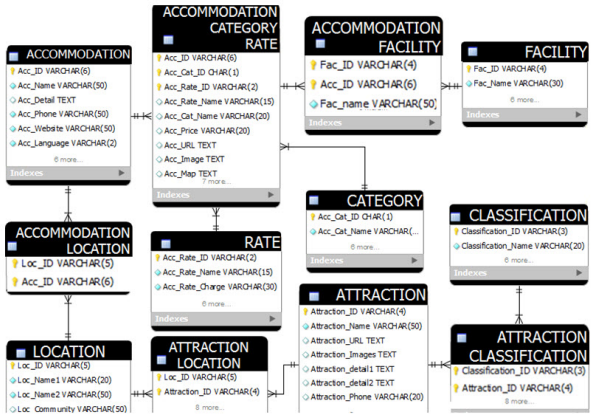

Fig.5: *The data model structure.*

After normalization, the original RDB is liked Figure 5 from MySQLWorkBench 5.2.29 CE. It gives an overview of the dataset and the properties of each class. The domain includes the accommodation by categories and rate of price, the accommodation facilities, and the attraction by activity types, the classification, and the locations of accommodations and attractions. RDB is based on the structure of a normalized model of www.huahin.go.th in version 2009.

3.2 Conversion to Web Ontology Language

After that, we create the ontology by Protégé 3.3.1[18] and save as the Hua Hin tourism OWL file for the SPARQL query. These ontologies include classes, subclasses, cardinality restrictions, transitive properties, class disjointness and class intersections.

The overview of the original Hua Hin tourism ontology graph is designed in Figure 6. The web ontology language contains the class, properties and transitive properties between classes.

The circle dashed line in Figure 7 is the overview of web ontology language graph in Figure 6. It is viewed by Jambalaya; Protégés plugin. The outer circle is the group of individuals of Figure 7.

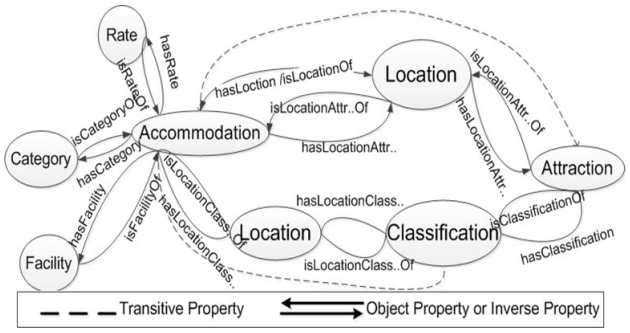

Fig.6: *OWL ontology graph of Hua Hin tourism.*

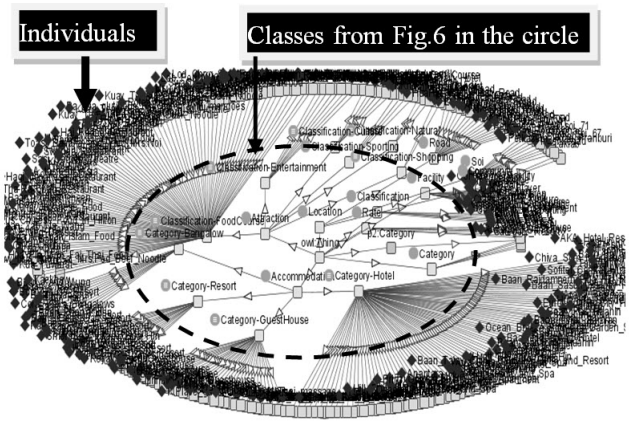

Fig.7: *Overview of tourism ontology by Jambalaya (Protégé's plugin)[19].*

3.3 Query Complexity Measurement

3.3.1 Query Complexity Methodology

The objective of this section is to reduce the query complexity from RDB by OWL engineering.

For measuring query complexity, we use Vardi's definition as a basis for the complexity measurement. We are interested in comparing the RDB and OWL query complexity. We use the following procedure to setup the conjunctive query.

1). RDB side, we transform the problem domain to SQL, and then conjunctive query.

2). OWL side, we transform the problem domain to SPARQL query, ontology concept form, and then conjunctive query.

Eventually, we compare the query complexity for a conjunctive query and find the answer in query language forms as shown in the flowchart in Figure 8. We perform the tests on five kinds of queries (Test #1-Test#5). On the left side, we test assuming the RDB is used. On the right side, it is the tests for OWL case. We have to transform from DL to FOL to compare with the RDB.

3.3.2 Setup the Queries

We set up the query examples from the given domains in Figure 6. Each query may traverse many domains for searching its results. Conjunctive opera-

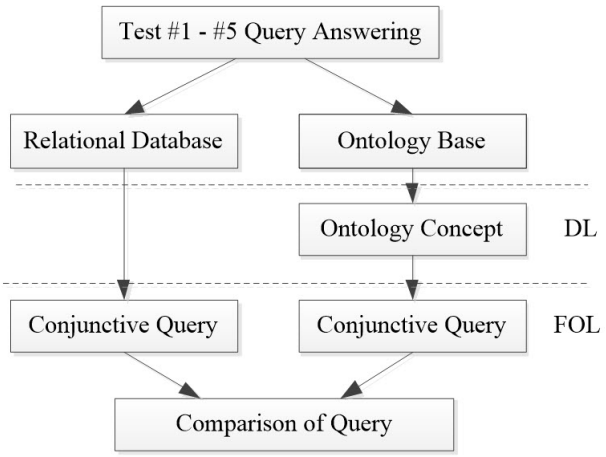

Fig.8: *The view of query expressions.*

tors may be used.

The queries used in the experiment are presented in Table 2. Column "Size" presents the size of data. Column "Basic Concepts" is the first variable. Column "Other Concepts" is the second variable. We rank their complexity degrees in Column "No.". Row No.1 means only the basic concepts are used.

Table 2: *Complexity (per bit) of different iterative timing recovery schemes.*

No.	Size	Basic Concepts	
1	+	Accommodation, Rate, Category, Facility	
No.	**Size**	**Basic Concepts**	**Other Concepts**
2	++	Yes	Location
3	+++	Yes	Location, Attraction
4	+++	Yes	Attraction, Classification
5	++++	Yes	Location,Attraction,Classification

The basic concept includes the related of Accommodation, Rate, Category and Facility. In Rows No.2-5, we use the basic concepts and other concepts as listed in Column "Other Concepts".

We consider each level and then show the example of the ontology concept form. It is in DL level of Figure 9. After that, all DL forms were transformed to conjunctive query.

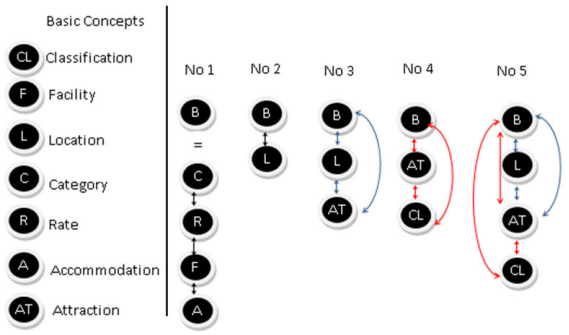

Fig.9: *Overview of the selected query.*

Consider the level 1 (called Test1). It includes the concept of Accommodation, Category, Rate and Facility. We select only constant values.

((Category Π (∃ hasRate{ Room Rate}) Π
(∃ hasFacility{ Facility A}) Π
(∃ hasFacility{ Facility B}) Π
(∃ hasFacility{ Facility C}))

In Test 2,it selects the *basic concept* related with the Location and other constant values.

((Category Π (∃ hasRate{ Room Rate}) Π
(∃ hasFacility{ Facility A}) Π
(∃ hasFacility{ Facility B}) Π
(∃ hasFacility{ Facility C}) Π
(∃ hasLocation{ Loc_Name}))

In Test3, it selects the *basic concept* related with the Location, Attraction and other constant values.

((Category Π (∃ hasRate{ Room Rate}) Π
(∃ hasFacility{ Facility A}) Π
(∃ hasLocation { Loc_Name}) Π
(∃ hasAttraction{Attraction B}) Π
(∃ hasAttraction {Attraction C}))

In Test4, it selects the basic concept related with the Attraction, Classification, and other constant values.

((Category Π (∃ hasAttraction{ Attr_Name }) Π
(∃ hasRate{ Room Rate }) Π
(∃ hasFacility{ Facility A }) Π
(∃ hasFacility{ Facility B }) Π
(∃ hasClassification{Classification A}))

In Test5, it selects the basic concept related with the Location, Attraction, Classification, and other constant values.

((Category Π (∃ hasLocation{ Loc_Name }) Π
(∃ hasRate { Room Rate}) Π
(∃ hasFacility{ Facility A }) Π
(∃ hasFacility{ Facility B }) Π
(∃ hasAttraction {Attraction C }) Π
(∃ hasClassification{Classification D})Π
(∃ hasClassification{ Classification E }))

In Figure 9, the circle numbers corresponds to the "No." of Table 2.

We show the examples of classes or concept constructors and ontology axioms of OWL in Table 3.

Table 3: *Concept constructors and ontology axioms.*

Class/Concept Constructors	
DL Syntax by Constructor	**DL Example**
1)$C_1 \sqcap \ldots \sqcap C_n$	Accommodation ⊓ Rate
FOLSyntaxby Constructor	**FOL Example**
2)$C_1(x) \wedge \ldots \wedge C_n(x)$	Accommodation $(x) \wedge$ Rate(x)
Ontology Axioms	
OWLSyntax[DLSyntax]	**Example**
3)subClassOf $[C_1 \sqsubseteq C_2]$	Hotel ⊑ Accommodation
4)equivalentClass$[C_1 \equiv C_2]$	Test1 ≡ Accommodation ⊓ Rate ⊓ Facility ⊓ Category
5)transitiveProperty$[P+ \sqsubseteq P]$	hasLocationAttraction$^+$ ⊑ hasLocationAttraction

We can explain Table 3 as follows. 1) is the DL syntax of $intersectionOf$ constructor and its example. 2) is the FOL syntax of intersectionOf constructor and its example. 3)-5) are OWL syntax, DL syntax and the example of subclassOf, equivalentClass, and transitiveProperty respectively. The parameter C is a concept (class), P is a role (property), x is an individual name respectively. Note that we use the conjunctive query in the FOL concept in the complexity experiment.

As the example of the FOL level, Test1 transforms the DL concepts to FOL as follows.

Q(X) ← Category-Bangalow(X) ∧ hasRate(X,A) ∧ hasFacility(X,B) ∧ hasAccommodationFacility(X,C) ∧ hasAccommodationFacility(X,D) ∧ A=Room_Rate_2 ∧ B=Beach ∧ C=Refrigerator ∧ D=Air Conditioning.

In the experiments Test 1, we compare between the above and FOL of RDB as follows.

Q(X) ← Accommodation(X) ∧ hasRate(X,A) ∧ hasCategory(X,B) ∧ hasAccommodationFacility(X,C) ∧ hasAccommodationFacility(X,D) ∧ hasFacility(X,E) ∧ A= Room_Rate_2 ∧ B= Category_Bangalow ∧ C=Beach ∧ D= Refrigerator ∧ E= Air Conditioning.

Regarding to the query language, in OWL side we use SPARQL query language, and then in the RDB side, we use the SQL one. The example of SPARQL query of OWL basic concept is as follows.

PREFIX Q:¡http://www.owl- ontologies.com/HuaHinProj.owl# >
Select ?Name WHERE {
?Accommodation Q:hasAccFacility Q:Beach.
?Accommodation Q:hasAccFacility Q:Refrigerator.
?Accommodation Q:hasAccFacility Q:Air_Conditioning.
?Accommodation Q:hasCategory Q:Category_Bangalow.
?Accommodation Q:hasRate Q:Room_Rate_2.
?Accommodation Q:Name ?Name.}

3.3.3 Comparison of Queries
The following steps are used to count the conjunctive query terms and measure the time spent.

Count Terms:
1. For (int Q_i= 0; $Q_i \leq$ noQueryAnswering; Q_i++)
2. {
3. Read QueryFile[Qi];
4. StartTime
5. Count ConjunctiveQueryTerm w/o fact.
6. EndTime
7. Write number of terms, and time.
8. }

Find the First and Second Variables of Vardi's Definition.

Let basic concepts is X^s. Other concepts are V^s.
1. For (int $Q_i = 0$; $Qi \leq$ noQueryAnswering; Q_i++)
2. {
3. Connect Database //or Ontology File
4. // check database connection
5. If it is the database connection
6. Then use connect to database
7. // check ontology connection
8. // If it is the ontology information
9. // Then create a graph model
10. Read QueryFile[Q_i];
11. StartTime
12. Execute the query;
13. EndTime
14. Write the answer, the number of answers, and time used.
15. }

In particular, we measure the factors as follows:
1. # Terms: Number of terms from conjunctive query.
2. Query complexity: X^{s*} V^s from Vardi's concept. X^s and V^s are the first and second parameter respectively.
3. # Answer: the number of the answers of the query.
4. Collecting the usage time between two machines that we used to compare.

3.4 Expressivity of OWL

The next step is to improve the ontology with more expressivity. We create the tourism ontology that is derived from ontology experts in Table 4.

Table 4: *Lists of sample tourism ontology.*

Ontology Name (.owl)	*DL expressivity*	# of Classes	# of object Properties	# of Restrictions
travel[20]	$\mathcal{SOIN}^{(\mathcal{D})}$	34	6	16
Etp-tourism[21]	$\mathcal{SHOIN}^{(\mathcal{D})}$	193	41	29
e-tourism[22]	$\mathcal{ALCHIF}^{(\mathcal{D})}$	19	12	2
price [23]	$\mathcal{SHIF}^{(\mathcal{D})}$	6	2	0
travelontology[24]	$\mathcal{SHIN}$	174	25	84

We can explain Table 4 as follows. The travel 1.0 ontology [20] is a small ontology for tourism by Holger Knublauch. It has $\mathcal{SOIN}^{(\mathcal{D})}$ expressivity that differs from $\mathcal{SHOIN}^{(\mathcal{D})}$ without the role hierarchy. E-tourism [22]'s $\mathcal{ALCHIF}^{(\mathcal{D})}$ means symmetric roles that are expressible by a TBox $\{R \subseteq R^-\}$, so that the concept $\exists R.T$ has no tree models [25]. It differs from $\mathcal{SHOIN}^{(\mathcal{D})}$ with the concept constructors $\mathcal{F}$; role functionality. Price [23] has $\mathcal{SHIF}^{(\mathcal{D})}$ expressivity. The SHIF-concept $\neg A \wedge C \wedge \forall U^-.C$, where $C \equiv \exists R^-.A \wedge (\leq 1R)$, has only infinite models with respect to the TBox:$\{R \subseteq U$, Trans(U) $\}$ [25] and data types. It differs from $\mathcal{SHOIN}^{(\mathcal{D})}$ in case of without nominal and cardinality restrictions. However, it has the role functionality instead. Finally, travelontology [24] has concept satisfacability as Hardness. It follows from TBox internalization in any extension of $\mathcal{SH}$ and ExpTime-hardness of $\mathcal{ALC}$-concept satisfiability with general TBoxes. It is an upper bound for $\mathcal{SHIQ}$, for $\mathcal{SHIO}$, and for $\mathcal{SHOQ}$. ABox consistency with ExpTime-Complete[25]. The next three columns are the number of classes, the

number of object properties, and the number of restrictions of each ontology files. Those are measured by Protégé ontology metrics tool.

We use these triple set testing in both of top-down and bottom-up views for All File, All Value From, Some Value From, Cardinality, Equivalent Restrictions from OWL and web application view.

Table 5: *Some OWL DL rules of classes/concepts*

Classes constructors	OWL DL Rules
intersectionOf	Accommodation⊓ (hasPriceRate ∃ Price-Rate3)
unionOf	ElephantTrekking ⊔ Paintballing ⊔ Trekking
complementOf	¬(Entertainment ⊔ FoodAndDining ⊔ Markets)
oneOf	{Pala-U Waterfall}
allValuesFrom	∀isLocalityOf.Amphoe
someValuesFrom	Accommodation hasRoom ∃ ConferenceRoom
minCardinallity	Accommodation hasFacility ⩾ 1
Cardinallity	Price hasCurrency = 1

We define the OWL DL rules and create asserted conditions in Protégé. We represent them in Compact OWL Class Display, the default format of Protégé and showed some example rules as in Table 5. From the defined rules, they become the asserted condition in Protégé in a property view.

Table 6: *Function lists of annotation tools.*

OWL Syntax	Example
subClassOf	Churches ⊑ Religion
equivalentClass	AllDestination ≡ LocationType ⊓ (isLocationTypedOf ∃ PostalAddress)
subPropertyOf	hasAccommodationPrice ∃ hasPrice
transitiveProperty	$\text{hasLocality}^{+} \sqsubseteq \text{hasLocality}$
Type $a : C$	SingleBed:BedFacility

From Tables 5-6, OWL syntax can be represented in OWL file such as Hotel class as follows.

<! – http://www.owl-ontologies.com/HHOntoTourism .owl#Hotel –>
<owl:Class rdf:about="#Hotel ">
<owl:equivalentClass>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasCategory"/>
<owl:hasValue rdf:resource="#Category_Hotel"/>
</owl:Restriction>
</owl:equivalentClass>
<rdfs:subClassOf rdf:resource="#Accommodation"/>
<owl:disjointWith rdf:resource="#Camp"/>
<owl:disjointWith rdf:resource="#Bangalow"/>
<owl:disjointWith rdf:resource="#Chalet"/>
</owl:Class>

According to OWL tags, the above example means Hotel class has property hasCategory with owl:hasValue restriction or the use of filler information with Axiom:DL as $\{\mathcal{R} : \dashv\}$. The SUFFICIENT & NECCESSORY asserted conditions in Protégé are translated to owl:equivalent-Class. The conjunctive operator AND is used. Owl:sub-ClassOf defines the superclass of Hotel. Owl:disjointWith defines the disjoint class of Hotel class. It means the set of classes: Camp, Bangalow, Chalet has different instances from Hotel. The Axiom:DL is $\{C_1 \sqcap C_2 \equiv \bot\}$.

Ontology visualization is viewed by Ontograf and Jambalaya in Figure 10. Part (a) shows only the

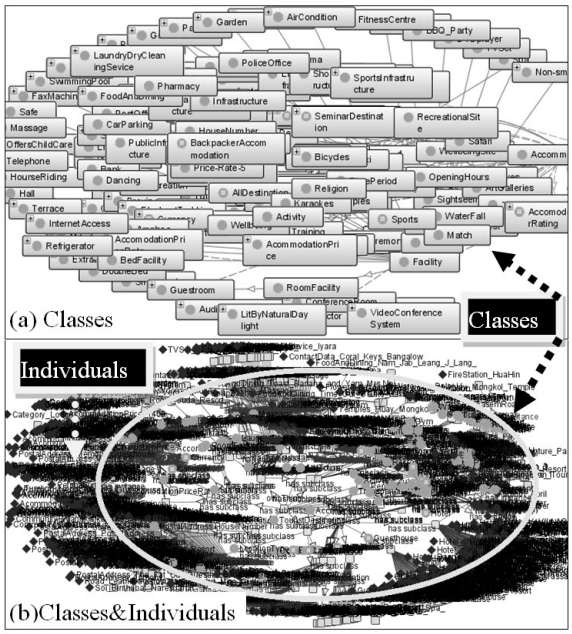

Fig.10: *Overview of tourism OWL DL (a) Show class by Ontograf (b) Show classes & individual by Jambalaya.*

classes and network of relations of classes. It is viewed by Ontograf [26]; Protégé 4.1's plugin. Part (b) shows the individuals of all classes (in the ellipse) that related with own classes. They are more flexible than the original OWL in Figure 7. It is viewed by Jambalaya; Protégé 3.3.1's plugin. The detail of more classes, restrictions and properties are in Section 4.2 on the experiments of expressivity.

3.5 Classification and Realization by Pellet Reasoner

Classification is to compute the subclass relations between every named class to create the complete class hierarchy. The class hierarchy can be used to answer queries such as getting all or only the direct subclasses of a class[27]. We find the answers by Pellet $\mathcal{SHOIN}^{(\mathcal{D})}$ reasoner shown in Figure 11.

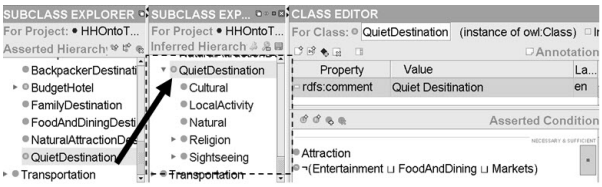

Fig.11: *Asserted condition in Protégé's properties view.*

At the Asserted Hierarchy box in Figure 11, we focus on QuietDestination class that is defined as the condition in Defining Classes box as follows.

"Find QuietDestination that is a type of Tourist Destination to Attraction class."

We model the above query as $\mathcal{ALC}$ structure of equivalence mode as follows.

$$(QuietDestination \sqsubseteq Attraction) \sqcap (\neg (Entertainment \sqcup FoodAndDining \sqcup Markets)) \quad (7)$$

When we execute a reasoner, it changes its direct subclasses from the set of

$$\{Cultural, LocalActivity, Natural, Religion, Sightseeing\} \sqsubseteq \textbf{Attraction} \quad (8)$$

to

$$\{Cultural, LocalActivity, Natural, Religion, Sightseein\} \sqsubseteq \textbf{QuietDestination} \quad (9)$$

Consider the Inferred Hierarchy box (dashed rectangular) in Figure 11. At the same time, the resources with types and expressions are shown in the Usage of QuietDestination box. It includes the classes, object properties that are superclasses, and inferred type of individuals. The object and data type properties appear in the properties box and the disjoint classes are shown in the Disjoints box. This is the classification method to look for the instances in the top-down approach.

Get Inferred Subclasses is the function in Protégé for finds the subclasses of the test class. These steps are connecting to the reasoner, computing subclasses by querying the reasoner. The reasoner log responses the concepts and records the total times in seconds. For example, subclasses of QuietDestination after the inference are {*Cultural, LocalActivity, Natural, Religion, Sightseeing*} as Equation (9).

Computing individuals belonging to classes is the function that we used to find the instances answer of test classes. These steps are connecting to the reasoner, computing individuals belonging to the class by the querying reasoner. The response shows the reasoner log of individuals belonging to the test class and the total time in seconds. For example, individuals belonging to QuietDestination after the inference are all of individuals belonging to these classes {*Cultural, LocalActivity, Natural, Religion, Sightseeing*}.

In the contrary, if we traverse from instances to find the type of instances or from class to find the superclasses, then we call the realizing methods as the bottom-up manner.

Realization finds the most specific classes that an individual belongs to; i.e., realization computes the direct types for each of the individuals. Realization can only be performed after the classification step since direct types are defined with respect to a class hierarchy. Using the classification hierarchy, it is also possible to get all the types for each individual [32]. Then, we still find the answers by Pellet $\mathcal{SHOIN}^{(\mathcal{D})}$ reasoner shown in Figure 12.

For example, the test class requests for "the accommodations that can hold a seminar where they are supplied with the conference room. The facilities were selected by someValueFrom of enumerations of conference supplies."

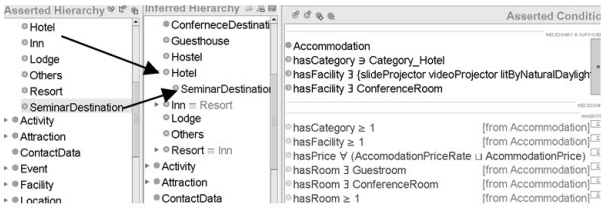

Fig.12: *Query condition finding superclass.*

From Figure 12, we model the above query as $\mathcal{ALC}$ structure of the equivalence mode as follows.

SeminarDestination $\equiv$ *hasFacility* $\exists$ ({*slideProjector videoProjector litByNaturalDaylight stage videoConferenceSystem screen audioEquipment*}) $\sqcap$ (*hasCategory* $\ni$ *Hotel*) $\sqcap$ (*hasFacility ConferenceRoom*) $\exists$ *Accommodation*

Get Inferred Super Classes is the function for finds the superclasses or type of the query. For computing superclasses steps include the query reasoner, the synchronize reasoner, finding the concepts, and calculating the total time in seconds. The time to synchronize equals to the time to clear knowledgebase + time for DIG conversion + time to update reasoner.

From Figure 12, the query response of Accommodation Type of SeminarDestination is as follows.

$$SeminarDestination \sqsubseteq Hotel \sqsubseteq Accommodation \quad (10)$$

Compute Type is the function that we use to get the type of an individual. These steps include get inferred type for instances and give the total time in seconds. For example, if we compute the type of individual 'a' that is presented in Equation (10), and then the inferred type for that individual as follows.

$$\{a\} \sqsubseteq SeminarDestination \sqsubseteq Hotel \sqsubseteq Accommodation \quad (11)$$

The difference between Equations (10) and (11) is that Equation (10) computes by the bottom up manner from a class to superclasses, and Equation (11) computes the type of an individual.

3.6 Query Complexity of OWL DL

Query Complexity of OWL DL in Table 1 is not possible to find since the OWL DL does not consider

the exact computational complexity. We know that DL-Lite is a fragment of OWL DL[11] especially tailored for handling efficiently large number of facts. The main focus is to provide efficient query answering on the data and to allow the use of RDBM technologies for such a purpose. DL-Lite also includes most of the main features of conceptual models, like UML class and ER diagrams. More specifically, DL-Lite includes the following features of OWL DL: a constrained form of someValue-From restrictions, conjunction, concept disjointness, domains and ranges of properties, inverse properties, and inclusion axioms for object properties.

The language DL-Lite is indeed a fragment of OWL DL. The variant provided here is called DL-LiteR since it allows for the property inclusion axioms. Other variants trade property inclusion axioms for functionality and inverse-functionality of object properties. Hence, our OWL DL has all concepts of DL-Lite$_{\mathcal{R}}$.

From Calvanese's theorem in Section 2, if we consider the complexity that is derived from DL-Lite, then the query complexity of DL-Lite$_{\mathcal{R}}$ is in LOGSPACE. Consequently compared to OWL DL, DL-Lite includes the portions of features of OWL DL, and then DL-Lite is the subset of OWL DL. If the class and properties in OWL DL is setup similar to the DL-Lite, then the data complexity of OWL DL in this research also is in LOGSPACE. Thus the query complexity of OWL DL in this work will be in LOGSPACE as well.

3.7 Testing OWL in Top Down and Bottom Up Perspective

3.7.1 Testing Area

The experiences in previous work [28] show that the ontology design usually goes with the inference rules and use the proper theories that leads to the ontology that reduces the query complexity degree. Then, we study the effect of performance of the top-down and bottom-up searches. We can summarize the methodology of testing as Figure 13.

Figure 13 shows the OWL environments. The tests for top-down and bottom up perspective are done using by Pellet 2.2.2 reasoner that was plugged in Protégé 3.3.1. According to the web application side, we test using SPARQL queries and compare the execution time between top-down and bottom-up queries.

3.7.2 Searching Algorithm

When the tourist looks for the accommodations from the webpage, the program connects to the middleware to search in OWL DL file. The Java Servlet program, controller of MVC design, search through the DL rules by LARQ (Lucence + ARQ[29]) algorithm with SPARQL query [30] to find the answers. In terms of the above process, the following algorithms are used for testing.

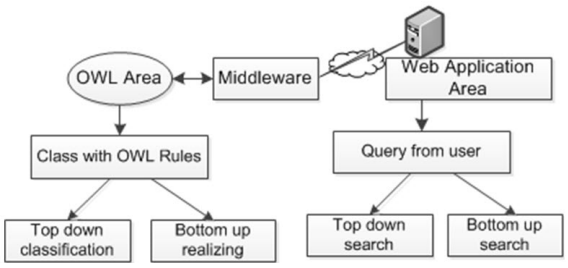

Fig.13: *Activity diagram of OWL and web application testing area.*

(0) Create model and receivethe user select search type.

Input: Select searching Type

Output: Searching UI by user selection

Process

1. Create Model
2. Read and index all literal strings
3. Index statements to add them to the model
4. Finish indexing and create the access index
5. If the user selects "advanced search"
6. Call controller (2) Advance Search.java
7. Else
8. Call controller (1) default Text Search.java

Step (0) is the step to select for selecting the search type. If a user would like to an advance search, the controller redirect to Step (2). If a user selects text search, the controller goes to Step (1).

(1) Text Search Case

Input: Text, Words.

Output: JSP result format

Process

1.Query using Lucene indexing method.
2.query-String = "joined Strings"
3.Defined Tourism Ontology prefix as "$Q_1, Q_2, \ldots, Q_n$:"
4.Defined Jena ARQ property prefix as "pf:"
5.Call SELECT function with concatenate pf:textMatch and input text. ("textMatch" as an implied relationship in the data)
6. Created Query Factory
7. *Start time*
8. Executed Query
9. *End time*
10.Calculated execute time=End time - Start time
11. Call view JSP page controller

Step (1), Query execution using a Lucene index for the query with the property function pf:textMatch. It can be thought of as an implied relationship in the data. The model of "http://jena.hpl.hp.com/ARQ/property#text-Match" can be looked up the literal from subjects with a prop-

erty value retrieving matched literals.

(2) Advance Search Case (and query search test)
Input: Includes two parts: 1). Text and conditions from UI, e.g., checkbox, and dropdown list. 2). User defined answering.
Output: The output result format with the set of answer columns.
Process
1.Query using Lucene indexing method.
2.Defined Input Strings as"$k_1, k_2, \ldots, k_n$:"
3.Defined Tourism Ontology prefix as "$Q_1, Q_2, \ldots, Q_n$:"
4.Call SELECT function with concatenate $Q(1 \ldots n)$ relationships and input text $k(1 \ldots n)$.
5. Created Query Factory
6. *Start time*
7. Executed queries
8. *End time*
9.Calculated the execute time=End time - Start time
10. Call the showing JSP page controller

Step (2) is different from (1) in the call SELECT at line 4. It concatenates the SPARQL queries from the user selection. In this step, the index builder traverses to every node in ontology. Then ResultSetFormatter is called to organize the data with SPARQL tags.

3.7.3 Measuring Top Down and Bottom Up View

From the preparation in Section 3.7.1 and 3.7.2, we setup the measurement for the top down and bottom up approaches.

1) We collect the execute time from searching individual belong to test classes for the top down equivalent session.

2) We use *Get inferred subclasses* for retrieving subclasses for the top down view.

3) We use *Computing individuals belonging to class* for find the instance answers for the top down view.

4) We use the *Computed-type* mode to run from instances to find the superclasses in the bottom up view.

5) We use Get *inferred superclasses* for retrieving superclasses for the bottom up view.

The execute time is shown in Equations (12)-(13).

$$T_{EX} = TotalTime - T_{SYN} \tag{12}$$

where

$$T_{SYN} = T_{CKB} + T_{DIG} + T_{UPD} \tag{13}$$

Let T_{EX} be the time of executing the answer set that may be inferred types of the bottom-up or individual belonged to the top-down approach. The execute time equals to the total time of running reasoner subtracted by the synchronize reasoner time (T_{SYN}). This optional set includes T_{CKB} as the time to clear knowledgebase, T_{DIG} as the time for DIG (DL implementation group) conversion, T_{UPD} as the time to update the reasoner, and T_{SYN} as the time to synchronize. These time variables are the optional of reasoner running time. They occur when we newly start the program or sometimes of newly run the reasoner.

The test class comes from the OWL DL restrictions that we setup in the OWL file. They include AllFile computes from owl:Thing, Cardinality includes maxCardinality, and minCardinality, allValueFrom and someValueFrom restrictions, and the others restrictions that come from the asserted condition according to $\mathcal{SHOIN}^{(\mathcal{D})}$ expression. We run all of conditions in top down and bottom up view until steady.

Then we run the web application that selects the query answering design from 1) to 5). The example results from the web application are in Figure 14. They include the execution time, and the resulting column that is selected by the user .e.g. name, contact info, and images.

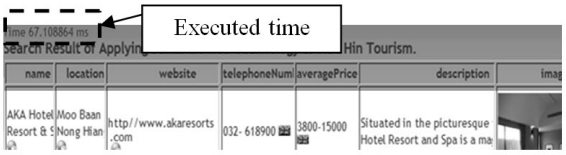

Fig.14: *Results of query answering from the web application.*

4. EXPERIMENT RESULTS AND ANALYSIS

In the experiments, we show the results according to our goals by:

1) Section 4.1: we measure the query complexity between RDB and OWL as we explained in Section 3.3.

2) Section 4.2: we measure OWL expressivity and comparing between original OWL and OWL DL as we described in Section 3.4.

3) Section 4.3: we measure the execution time of OWL DL and web application as we mentioned in Section 3.7. We compare between the top down and the bottom up approach queries.

Note that Step 1) is conformed to our first contribution. Step 2) and 3) are matched up with the other one mentioned in Section 1.

4.1 Measuring the query complexity between RDB and OWL

After preparing the query (Section 3.3), the experiments were run on two different speed machines as follows:

Machine#1: The experiment was conducted on a processor: Intel (R) CPU T2050 1.60 GHz; 798 MHz; memory: 0.99 GB hard disks: 80GB 32-bit Operating System running Window XP Professional.

Machine#2: The experiment was conducted on a processor: Intel (R) Core(TM) i5 CPU M430 2.27GHz; memory: 4GB hard disks: 320GB 64-bit Operating System running Window 7 Home Premium.

The software was installed in 2 machines: Apache-Web Server 2.2.8, MySQL Database 5.0.51b, php-MyAdmin Database Manager 2.10.3. Protégé 3.3.1. RacerPro 1.9.0 reasoner. Java 1.6.0_18 was used.

There are the uncontrollable factors in this step. These include the load time and execution time. Then we measure the total of them as the usage time.

The results of section 3.3 Query Complexity Measurement is represented as follows.

Case 1: According to Term factors, the number of terms is counted from the conjunct operators in the conjunctive queries both of RDB and OWL. The experiment # 1 results are shown in Figure 15.

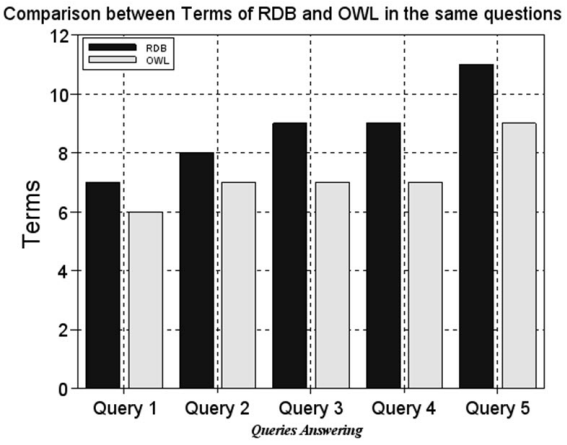

Fig.15: *The number of terms between RDB and OWL.*

The number of terms between RDB and OWL are related. The RDB and OWL expression terms increase along with the depth of graph. However, the total of OWL expression terms still increases less than the RDB terms because subclasses in OWL can reduce the expression terms. We use the subclass of Accommodation class in OWL. When the SPARQL is queried, it can be found out within the subclass constantly. We can imply the OWL expression by subclass terms instead the class ones. This is the reason why the number of terms of OWL less than RDB terms in the Figure 15. In addition, we can see the difference between the terms of RDB and OWL slight wider, when the number of terms and joins increases.

Case 2: The experiment # 2 and 3 results were represented in Figure 16.

The number of the answers implies the query complexity.We can see clearly in RDB that the queries used to find the answer are much more complex, i.e., the #answer = 5 and query complexity =6600. The recall is (5/6000) ¡ 1. This value is compared with that of OWL. It is seen that the recall of answers for both cases are very close. When there is a small number of the answers, we can find them easier from the ontology.

One of the reasons, OWL finds the answers easily because of the transitive property.

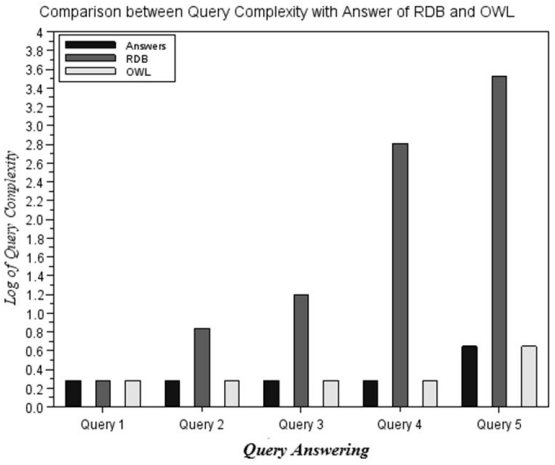

Fig.16: *Comparison between query complexity with answer of RDB and OWL from SCILAB [31].*

For example, *has-LocationAttraction* in the Accommodation ontology has eliminated the number of joins for a relational model, and also reduces the number of query terms. In the next line, we can see the Accommodation class connects Location class with hasLocation property. Location class connects to Attraction class with hasAttraction Property.

$$Accommodation \rightarrow Location \rightarrow Attraction$$

We can formulate easier if we use transitive property hasLocationAttraction to connect between Accommo-dation and Attraction as below:

$$Accommodation \rightarrow Attraction$$

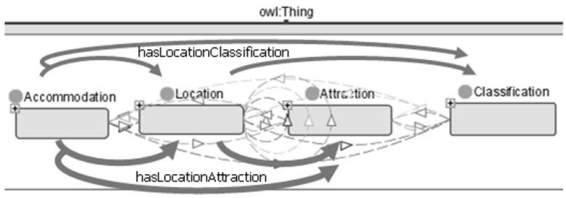

Fig.17: *Example of transitive Properties.*

For the other transitive properties of original OWL, we present in Figure 17. As a result, OWL3, OWL4 and OWL5 can reduce the expression terms from the relational model in Figure 5. In additional, the usage of the transitive property reduces of the query complexity as well.

The results of experiment # 4, on the usage time between 2 machines, are represented in Figures 18

and 19. They show the usage time by comparing between that of RDB and that of OWL.

The machine#2 (Figure 19) can reduce usage time more than the machine1 (Figure 18). However, we show that regardless of the speed of machines, the OWL still uses less time than the RDB.

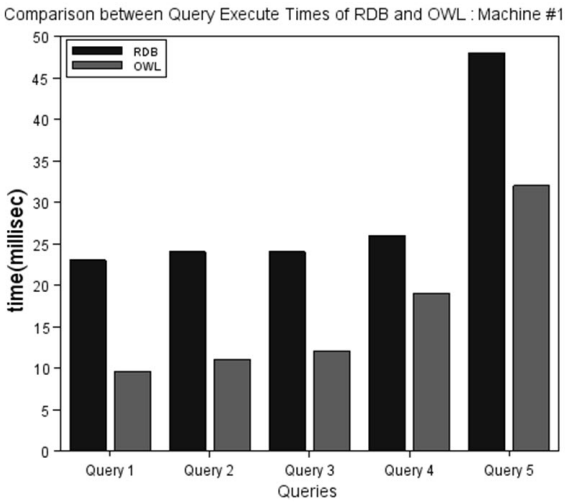

Fig.18: *Query times between RDB and OWL:machine#1.*

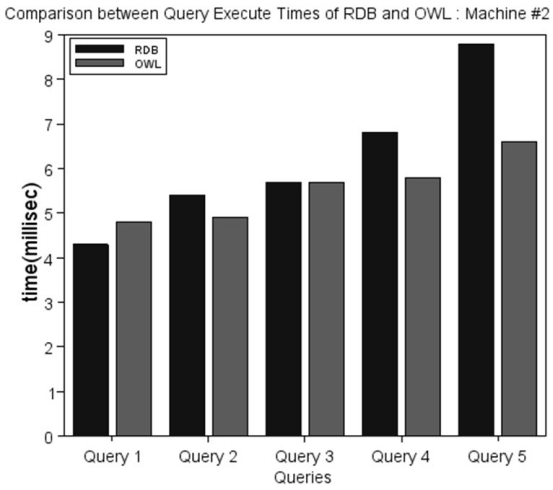

Fig.19: *Query times between RDB and OWL:machine#2.*

4.2 Measuring OWL expressivity

We use OWL model metric tool of Prot?g? to evaluate the DL expressivity, called SHOIN(D), which has been recently successfully implemented in practical reasoning systems, such as Pellet [32]. For this section, Pellet is used to classifying and realization the ontology because RacerPro does not support the nominal [33]. In reasoning with OWL, ontology classification is the computation of subsumption hierarchies for classes and properties. It is one of the most important tasks for all OWL reasoners. Description logics (DL) have been found useful in numerous applications such as information integration and conceptual modelling [34]. We will measure the time between inference query from the top-down and bottom-up approaches. In addition, we use Prot?g?'s metrics tool to examine the ontology by the logical perspective. The results are in Table 7.

In the Table 7, each column shows that OWL DL relates to $SHOIN^{(D)}$ features that we run in our tests. We can say that OWL DL has more complex classes, properties, super/sub-properties and restrictions than the original OWL that is an ontology file in previous work [28]. We can explain in the *DLexpressivity* row as follows.

DL expressivity of original OWL is SOI(D). It has a reasoning problem in the concept of satisfiability and complexity in ExpTime-Complete as same as SHOIN(D). However, the original OWL differs from the current OWL DL without the nominal and role hierarchy. This logic is between ALCIO and SOI [25]. From the description logic property as we mentioned in Equation (6) of Section 2. The OWL DL expressivities of this paper are investigated in Table 7 for the case that is related to SHOIN(D). The other rows are the number of primitive and defined named classes. The next rows are the number of restrictions that include existential, universal, cardinality, min-cardinality, max-cardinality, and has-value. The others are the number of properties.

Table 7: *Comparison of the OWL DL Expressivity.*

Metrics	*Original*	*OWL DL*
DL Expressivity	$\mathcal{SOI}^{(\mathcal{D})}$	$\mathcal{SHOIN}^{(\mathcal{D})}$
Primitive Named classes	6	156
Defined Named classes	19	22
Existential Restrictions	0	5
Universal Restrictions	0	9
Cardinality Restrictions	0	8
MinCadinality Restrictions	0	10
MaxCadinality Restrictions	0	2
HasValue Restrictions	41	11
Object Properties	16	44
Datatype Properties	10	70
Properties with a domain specified	25	106
Properties with a range specified	15	43
Properties with an inverse specified	16	28

We can summarize from Table 7 to the histograms in Figure 20.

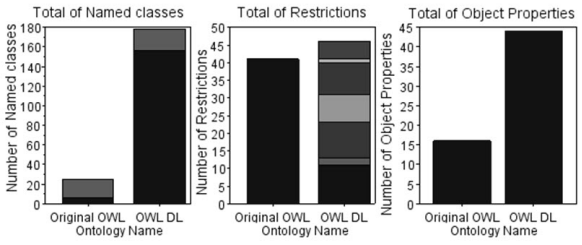

Fig.20: *Total of ontology metrics.*

In Figure 20, the total of named classes, object properties, and restrictions are shown for the original OWL and OWL DL. The total of named classes

includes the primitive and the defined named classes respectively. The named classes are created increasingly in OWL DL. The total of OWL DL classes as 178 equal to 156 primitive named classes + 22 defined named classes are greater than the original OWL classes as 25 from 6 primitive named classes +19 defined named class. In the next histogram, the total of restrictions includes Existential, Universal, Cardinality, and Has-Value restrictions. Likewise the named classes, the restriction are added to OWL DL. Hence, the total of OWL DL restrictions is greater than that of the original OWL restrictions. The last bar is the total of object properties. It is seen that the OWL DL also has the number of object properties more than original OWL.

4.3 Measuring performance of top-down and bottom-up search

From Section 3.7, we look for the answer in a top-down manner, which is meant to find the instances that belong to those classes. On the other hand, we search in a bottom-up manner that searches for the types of individuals/classes.

According to the uncontrollable factors are in Section 4.1. At this step, when we executed the Pellet reasoner and Protégé function, the load time. Using the execution time in Equations (12)-(13) in Section 3.7.3, we can measure the execution time in this step.

The results of Section 3.7, measurement of the searching time between reasoner's activity and web application algorithm, are shown in Figure 21 as follows.

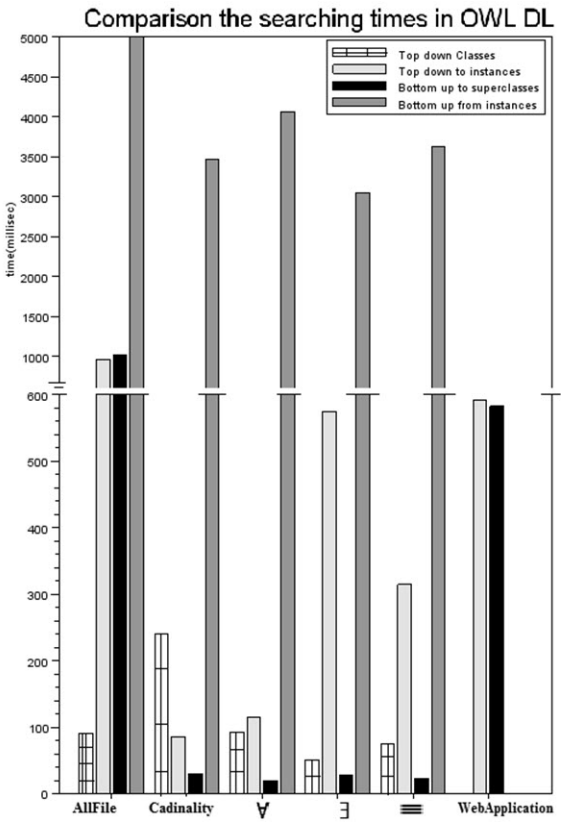

Fig.21: *Comparison the searching times in OWL DL.*

In Figure 21, we measure from the logic perspective of OWL DL by running each case several times until steady. The "Top-Down" and "Bottom-Up" bars show the time for finding the answers in milliseconds. The legend is explained the symbols as follows. "*Top-down classes*" refers to getting inferred subclasses and "*Top-down to instances*" refers to computing individuals belonging to classes. "*Bottom-up to superclasses*" refers to getting inferred superclasses and "*Bottom-up from instances*" refers to computing types from the instances. According the test cases: "All Files" is the test from owl:Thing in top-down and from the deepest node of ontology individuals in the bottom-up manner. "Cardinality" is the conditions containing cardinalities. "∀" is the condition containing All-Values-From (∀)."∃" is the condition containing Some-Value-From (∃). "≡" is the triples that containing owl:Equivalent tags. Finally, "Web Application" is the execution time from our web application.

We investigate the results into three cases.

Case 1: We compare between the searching time of top-down classes and instances. Getting inferred subclasses takes times are less than the time to get the individuals belonging to the class from Allfiles, ∀, ∃, and "≡" respectively. Because we get individual nodes as tree leaves, the numbers of nodes in top-down instances are greater than subclasses except in "Cardinality" case. This is, because the time of getting to subclasses from parents class is greater than that of the inherited condition by the top down through instances manner. Hence, it shows the average searching times in top-down subclasses are more than that of instances.

Case 2: We compare between the searching time of the bottom-up classes and instances Allfiles to "≡" respectively. Getting the super-classes uses less time than the time for computing types of instances explicitly. This is because the reasoner realizes that the number of super-class nodes is less than that of instances.

Case 3: We compare between the average time of inference in the top-down manner from Allfiles to "≡" and in the bottom-up manner. The reasoner classifies in the top down through instances manner using less time than the realization in the bottom-up through instances manner.

From between "Top-down classes" and "Top-down to instances", we choose "Top-down to instances" since it is suitable for searching to get to individuals although it may take more time to search to individuals. It still takes less time compared to "Bottom-up from instances". Thus, we choose "Top-down instances" in our web application as shown in the last

bar.

"Bottom -up to superclasses" takes not much time to search compared to "Bottom-up from instances". This is because some search condition that does not need to go from instances. Thus, in our web application, we choose "Bottom-up from instances".

5. APPLICATION

5.1 System Architecture

The architecture of the system is as in Figure 22. The web application is connected to the webserver. The administrator manages the ontology management tool, annotation tool, FTP client tool, and semantic web searching tool. The users find the tourism information of Hua Hin district. They can search through the Internet and request the information from the web server.

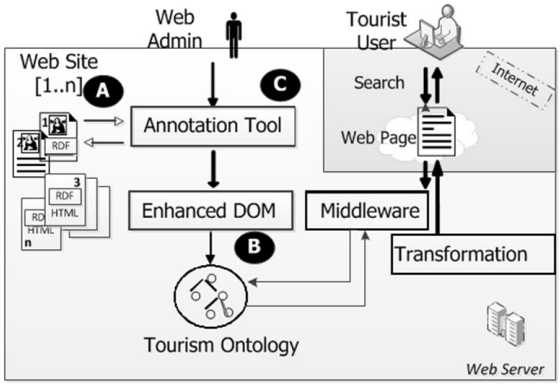

Fig.22: *Architecture of Semantic Web Application.*

The input query will be transformed from the HTML input to the SPARQL queries that can be searched from the graph model. The model is handled by the Jena middleware. The LARQ function is called by ARQ in Jena. It indexes the text, creates the graph model, and then traverses through the graph to search the answer to the query. After that, if it finds the answer, the output in the SPARQL/XML tag is sent to transform by XSLT. Finally, the user will receive the answer representing to them in the HTML format. The conversion is done by the middleware.

5.2 Implementation

From Figure 22, we explain the lists of tasks between the web administrator and annotation tools as in Table 8. The functions of annotation tools are divided into A) FTP Client B) ontology manager tool and C) website annotation tool. For example, the user input of the resort requirement could be automatically transformed into the RDF triples and embedded in the HTML webpage.

To explain the Figure 22 related to Table 8. First, we use an FTP client tool to retrieve the webpage

Table 8: *Function lists of annotation tools.*

Web Administrator Task	Annotation Tool Activities	Tool
1. Download website	Retrieve website or host messages	A
2. Selects ontology	Ontology Requisition and XML Log	B
3. Create annotation	Add RDF tags in web annotation	C
4. Edit annotation	Update RDF tags in web annotation	C
5. Upload website	Upload website or host messages	A

from our interesting web. Next, we select the appropriate ontology for annotating the webpages. For the annotation, we use DOM technologies to traverse along the RDF/OWL node graph that is modelled by Jena. This annotation tool uses to manage the webpage detail in the XML format. The web administrator uses this tool to annotate the webpages for preparing the format for searching on ontology later. On server side, the details of webpages are saved to the OWL file. On client side, the RDF tags are embedded to the header tags of HTML files. After finishing the annotation and setting the metadata tags to the server file and the client web, there are the display issue that we should solve. Since the regular browser, e.g., Internet Explorer, Mozilla Firefox, and Safari cannot understand OWL, RDF/XML tags, to help the user not to download the plug-in software for displaying them. We use the SPARQL query results XML format and then convert XML to HTML with XSLT [35]. Hence, the resulting webpages that embed the metadata can be displayed on the normal browser.

The last but not least, the searching tool part of web application is used the advantage from the Section 4.2-4.3 experiment to improving the search engine for semantic web. Then, we can show the application when the user uses the advanced search.

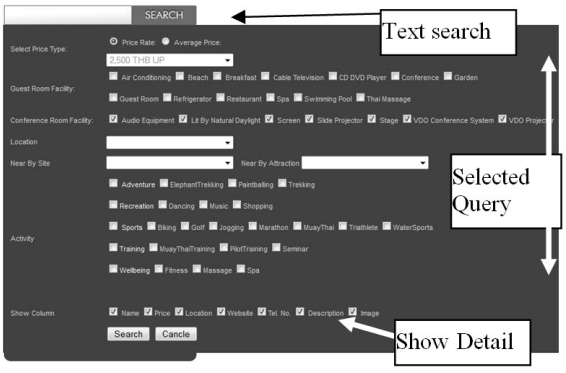

Fig.23: *Advanced search page.*

From Figure 23, 'Advanced search page' is the page that using technologies from this research. It is separated to three main parts. There is 'Text Search' part that can expand to select more searches below by JQuery. 'Selected Query' part is the part helping user select from ontology restriction, e.g., symmetric properties, transitive property, superclasses/subclasses search, and individuals. They are connected by the equivalence of OWL DL. The last

part is 'Show Detail' that user can select the column follow by the detail.

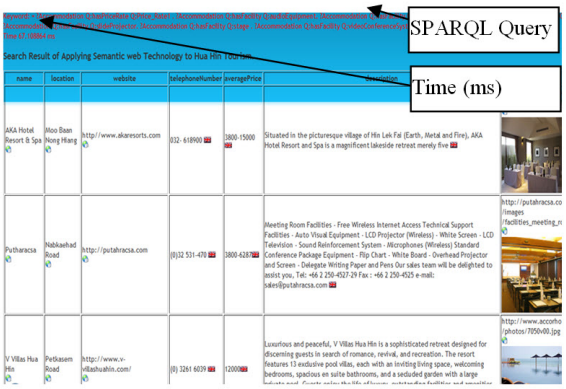

Fig.24: *Result page.*

The result as in the Figure 24, we show Name, Price, Location, Website, Telephone Number, Description, and Image. From Figure 24, 'Result page' is the result of the advanced search page. It comes from the ontology selection and shows the column for the user from Figure 22. Next, SPARQL queries are executed through LARQ of Jena middleware by the XML result set formatter. The SPARQL tags are transformed again by an XSLT technology tool. Then the result is showed in Figure 23, which includes 3 parts. There are the SPARQL query, searching time in milliseconds and the detailed column of the user selection. The examples of search and results use the 'SeminarDestination' problem that transform from DL (Section 3.4) to SPARQL queries.

6. CONCLUSION

In this work, as our first contribution, we study the issues that help improve the search engine. The ontology is used to develop the prototype search engine for Hua Hin tourism. We compare the developed one with the traditional RDB version in many aspects. From our study, we show that the OWL has the advantages over the RDB in terms of query complexity. By using the properties and restrictions, we are able to find the answer of queries in less time.

For the 2nd contribution, we study the searching time of top-down and bottom approaches. We create a more complex ontology in OWL DL with expressivity as $\mathcal{SHOIN}^{(\mathcal{D})}$. The rule helps analyze the searching time in a separate part of ontology. From the experiments, we also know that the ontology design in the top-down view will use the searching time to instances less than that of the bottom-up view. We test this by developing the web application and analyze the response to the user query. Looking for the instances from the top-down view fits the kind of query such as searching a list of accommodation and attractions. To find the type to which the instances are related, we apply the realization of a class type by the bottom up method. Our finding helps the developers design the ontology. For example, one can select the appropriate restrictions for query responses and create the set of rules. Thus, the query time can be optimized to improve the search in the semantic web engine.

In the future work, we can use the result to improve the search engine in Semantic Web technologies. For example, one can select which class and which relation to answer the user query. In case, we obtain the benefits from complexity reduction or the selection from what restrictions by reducing the searching time. This may not depend on whether the queries' answers are searched in the bottom up or the top down manner.

7. ACKNOWLEDGEMENT

This work is supported in part by Royal Golden Jubilee Program, Contract no. PHD/005/2554, and Silpakorn University Research and Development Institute and Faculty of Science Research Funding, Silpakorn University, Thailand.

The authors would like to thank you the Muang Hua Hin Municipality for giving the www.huahin.go.th (2009) database structure and the reviewers for all constructive comments.

References

[1] Hannes, W., "Intelligent systems in travel and tourism," in *Proceedings of the 18th international joint conference on Artificial intelligence, 9-15 August 2003*, Morgan Kaufmann Publishers Inc.: Acapulco, Mexico, 2003.

[2] Dell'Erba, M., et al., "Exploiting Semantic Web Technologies for Harmonizing E-Markets," *Journal of Information Technology and Tourism*, vol.7, no.3-4, pp. 201–219, 2005.

[3] Sadasivam, G.S., C.Kavitha, and M.SaravanaPriya, "Ontology Based Information Retrieval for E-Tourism," *(IJCSIS) International Journal of Computer Science and Information Security*, vol.8, no.2, pp. 78–83, May 2010.

[4] Abrahams,B.,*Tourism Information Systems Integra-tion and Utilization within the Semantic Web*, in Faculty of Business and Law,School of Information Systems, Victoria University, pp.5–112, 2006.

[5] D. Li, et al., "Swoogle: a search and metadata engine for the semantic web, " in *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, ACM: Washington, D.C., USA, 2004.

[6] T. Berners-lee,et al., "Tabulator:Exploring and analy-zing linked data on the semantic web," in *Proceedings of the 3rd International Semantic Web User InteractionWorkshop*,Athens,Georgia,USA,2006.

[7] D. Huynh, S. Mazzocchi, and D. Karger, "Piggy Bank: Experience the Semantic Web inside your web browser," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no.1, pp. 16–27, 2007.

[8] T. Berners-Lee, *Linked Data*, updated 2009/06/ 18 18:24:33 [cited 2011 October 21]; Available from: http://www.w3.org/DesignIssues/ Linked Data.html.

[9] Y.V. Moshe, "The complexity of relational query languages (Extended Abstract) ," in *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, ACM: San Francisco, California, United States,1982.

[10] D. Calvanese, et al., "Data complexity of query answering in description logics," In *Proc. of KR 2006*, pp. 260–270, 2006.

[11] D. Calvanese, et al., *OWL 1.1 Web Ontology Language Tractable Fragments*. 2006.

[12] P. Hitzler, M. Krötzsch, and S. Rudolph, *Foundations of Semantic Web Technologies*, Chapman & Hall/CRC, 2009, ch.4.

[13] P. Hitzler, M. Krötzsch, and S. Rudolph, *Foundations of Semantic Web Technologies*, Chapman & Hall/CRC, 2009, ch.5.

[14] I. Horrocks, et al., "OWL: a Description-Logic-Based Ontology Language for the Semantic Web," *The Description Logic Handbook Theory,Implementation and Applications*,Cambridge University Press, 2007.

[15] F.J. García-Peñalvo, J. García, and R. Therón, "Analysis of the OWL ontologies: A survey," *Scientific Research and Essays*, vol.6, no.20, pp. 4318–4329, 2011.

[16] A.G. Fraser, and E.M. Marcotte, "A probabilistic view of gene function," *Nature Genetics*, vol. 36, pp. 559–564, 2004.

[17] Y. Chen and Y. Chen, "On the top-down evaluation of tree inclusion problem," *Int. J. Information Technology,Communications and Convergence*,in press.

[18] *Protégé*, [Online], http://protege.cim3.net/ download/old-releases/3.3.1/full/.

[19] M. Storey, et al. "Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protege," in *Workshop on Interactive Tools for Knowledge Capture (K-CAP-2001)*, 2001.

[20] *travel.owl*,[Online],Available from: protege. cim3.net/ file/pub/ontologies/travel/travel.owl

[21] *ETP-tourism Ontology*, [cited 2010 25 August]; Available from: http://intra.info.uqam.ca/ Members/valtchev_p/mbox/ETP-tourism.owl/view.

[22] *E-tourism Ontology*. [cited 2009 15 August]; Available from: e-tourism.deri.at/ont/e-tourism.owl.

[23] *Price Ontology*, [cited 2010 20 August]; Available from: http://gaia.fdi.ucm.es/ontologies/price. owl.

[24] *TravelOntology.owl*, [cited 2010 25 July];Availabl from: http:augmented.man.ac.uk/ontologies/ TravelOntology.owl.

[25] *Complexity of reasoning in Description Logics*, [cited 2010 28 July]; Available from:http://www.cs.man.ac.uk/ ezolin/dl/ #LutzImproved2004_Anchor.

[26] S. Falconer, *OntoGraf*, updated 2011 June 17 at 18:24 [cited 2011 September 19,]; Available from: http://protegewiki.stanford.edu/wiki/OntoGraf.

[27] Clark & Parsia, L. *Pellet Features*, [cited 2010 28 August];Available from: http://clarkparsia.com/pellet /features/.

[28] C.Choksuchat, and C. Chantrapornchai, "Benchmark-ing Query Complexity between RDB and OWL," in *Future Generation Information Technology (FGIT2010)*, Springer Berlin / Heidelberg. pp. 352-364, 2010.

[29] *ARQ-A SPARQL Processor for Jena*,[cited 2011 Sept 21]; Available from: http://openjena.org /ARQ/

[30] *LARQ - Free Text Indexing for SPARQL*, [cited 2011 September 21]; Available from: http://jena .sourceforge.net/ARQ/lucene-arq.html.

[31] *scilab-5.3.3*, [cited 2011 26 September]; Available from: http://www.scilab.org/.

[32] E. Sirin, et al., "Pellet: A practical OWL-DL reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.

[33] V. Haarslev, et al., "The RacerPro knowledge representation and reasoning system," *Semantic Web (IOS Press)*, e-pub date: Friday, March 25, 2011.

[34] B. Motik, U. Sattler, and R. Studer, "Query Answering for OWL-DL with rules," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no.1,pp. 41–60, 2005.

[35] D. Beckett, and J. Broekstra, *SPARQL Query Results XML Format*, [cited 2010 28 July]; Available from: http://www.w3.org/TR/rdf-sparql-XMLres/.

Chidchanok Choksuchat received the B.S. in Computer Science and M.S. degrees in Information Technology from Silpakorn University, Thailand in 2004 and 2011, respectively. She is currently a Ph.D student in the Computer Information System in the Department of Computing at Silpakorn University, Thailand. Her research interests include the Parallel Computing, CUDA Architecture, Ontology Engineering, Linked Data and Semantic Web.

Chantana Chantrapornchai received the B.S. in Computer Science, from Thammasat University, Thailand. She received a Ph.D. degree from University of Notre Dame, USA, in 1998. Currently, she is an associate professor at Faculty of Science, Department of Computing, Silpakorn University, Thailand. Her research interest includes high-level synthesis, computer architecture, parallel computing, embedded systems and fuzzy logic.