OWLizr: Knowledge Representation System for Bahasa Indonesia Based on Web Ontology Language Description Logic (OWL DL)

Fariz Darari, Adila Alfa Krisnadhi, and Ruli Manurung
Information Retrieval Laboratory
Faculty of Computer Science, Universitas Indonesia
Email: fariz@ui.ac.id, adila@cs.ui.ac.id, maruli@cs.ui.ac.id

***Abstract*—In this paper we present OWLizr, a system that constructs formal knowledge representations using the Web Ontology Language (OWL) from natural language text in *bahasa Indonesia*. The design of OWLizr is mainly concerned with the representation of knowledge about real-world events using the reification technique. Such knowledge, which is commonplace in naturally occurring texts, is not typically handled by logics for ontologies such as Description Logic. OWLizr consists of four modules: the NLP Semantic Analyzer, KB Generator, KB Reasoner, and SPARQL Query Generator. We also developed an ontology to support the knowledge representation and reasoning process in the KB Generator and KB Reasoner. The NLP Semantic Analyzer reuses the semantic analyzer program developed by Mahendra [2]. Our system supports question-answering (QA) on the knowledge base using the SPARQL Query Generator module.**

I. Background

MANY sources of knowledge can be found available as natural language text. One primary example is the wealth of information available on the Web. The Semantic Web research agenda aims to create similar resources that can also be processed and reasoned with by software agents. Naturally, one way to populate the Semantic Web is to develop an automated system that is capable of processing natural language text on the Web and convert it into formally represented knowledge using Semantic Web standards and tools. Such a system should be able to retrieve knowledge from a textual document and perform automated reasoning on the extracted knowledge.

A. Previous Work

There are several previous research works about how to process natural language text, specifically those written in *bahasa Indonesia*, into semantic representations. Two examples are the works done by Larasati [1] and Mahendra [2]. Both of these works mainly focused on linguistic aspects, i.e. building the syntactic and semantic apparatus that affords the transduction of logical representations.

Larasati [1] presents a model of deep syntactic and semantic processing to support QA in Bahasa Indonesia. The model uses a unification-based grammar and specifies lexical semantics for each lexeme and semantic attachment rules for each grammar rule using lambda-calculus notations. The model implementation is in Prolog language and the output knowledge is in the form of a set of conjunctively-interpreted first order logic literals.

Mahendra [2] extended the model by adding a number of axioms designed to encode useful knowledge for answering questions, thus increasing the inferential power of the QA system. The axioms broadly fall into two categories, NLP axioms and world knowledge axioms. The model also adopts a simple ‘flat’ semantic representation [3], where a logical expression is simply a conjunction of first order logic literals.

B. NLP and Event Representation

One specific type of knowledge is event knowledge, which concerns representation of events and occurrences. Such knowledge, which is commonplace in natural language texts, is not typically handled by logics for ontologies such as Description Logic. There are in fact many techniques to represent events. One such technique is by using reification, which treats events as objects. There exists a well-known knowledge representation model based on this reification, the so-called Neo-Davidsonian approach [4]. It represents arguments of events with *thematic roles*, e.g., agent, patient, theme, time, and location. Figure 1 shows an example of semantic analysis using the Neo-Davidsonian approach.

.

$$\text{Brutus stabs Caesar}$$
$$\mapsto \exists e[\mathit{stab}(e) \land \text{Agent}(e) = b \land \text{Patient}(e) = c]$$

Fig. 1. Neo-Davidsonian Semantic Analysis

Our research calls on the fields of Natural Language Processing (NLP) and Description Logic (DL) as its foundations. NLP is a field in which text is parsed and processed, so that machines can understand its meaning. The information sources for NLP are the lexicon, grammar, and corpus. One popular NLP technique is syntax-driven semantic analysis. It is based on *Frege's principle of compositionality*, which states that the meaning of a linguistic constituent is a mapping function from the meaning of its parts [5]. The implementation of this technique was proposed by Montague using lambda calculus with beta reduction. Our research's position is in the middle between NLP (Septina [1] and Mahendra [2] works) and DL (Franconi's work [8]), acts as a bridge that connects them.

C. Description Logic

Description Logic (DL) is a very promising knowledge representation language. It has many advantages over previous knowledge representation languages, such as semantic networks and frames [6]. One of the implementations of DL is the Web Ontology Language, or OWL specifically the OWL-DL variant. OWL is a web ontology language recommended by W3C, and is designed to support the Semantic Web vision [7]. Our research utilizes OWL-DL as the main language for knowledge representation and reasoning. OWL-DL consists of two main components, the *TBox* and *ABox*. The TBox contains class and property definitions and axioms, whereas the ABox contains concept and property assertions.

In previous research, there is KODIAK, a knowledge representation language for lexical semantics using relation-based DL [8]. KODIAK contains syntactic types such as Relations, Aspectuals, and Absolutes, and basic operators such as Manifest, Dominate, Instantiate, and Disjointness. For example, the representation for the sentence *"Giotto paints the Sermon to the Birds"* is shown in Figure 2.

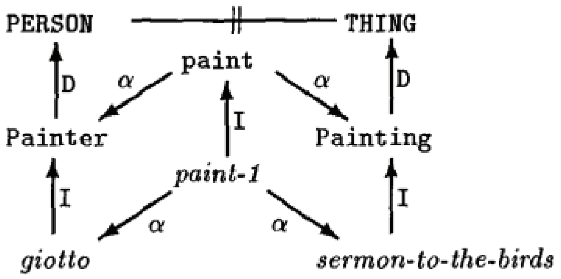

Fig. 2. KODIAK Knowledge Representation

The relation *paint* manifests two aspectuals, *Painter* and *Painting*. They are dominated by absolutes *PERSON* and *THING*, which are disjoint with each other. *paint-1* is the instantiation of *paint* and manifests the instances *giotto*, which is the *Painter*, and *sermon-to-the-birds*, which is the *Painting*.

II. OWLizr Knowledge Representation

OWLizr mainly uses *event* as the knowledge representation of a declarative sentence. We treat events as objects. This is called *reification*. An event can have an agent, patient, action, or location retrieved from appropriate phrases in a sentence. For example, *"Budi buys a car"* or *"Budi membeli mobil"* will have *"Budi"* as an agent, *"membeli"* as an action, and *"mobil"* as a patient. This approach is similar with the Neo-Davidsonian approach, which represents arguments of event as *thematic roles*.

OWLizr is also able to represent *background knowledge* of events. The thematic roles such as agent, person, location, or attribute can have background knowledge, i.e., the underlying knowledge that explains the definition of the objects. For example, an agent or patient could be a person, or a non-living object. Then, as we can see from the example above, *"Budi"* will be defined as a person, and *"mobil"* will be defined as a vehicle, which is a non-living object. This approach is similar to the knowledge representation of KODIAK, which from the example above, states that *giotto* is a person and *sermon-to-the-birds* is a thing [8]. OWLizr must be able to express the knowledge representation of an event and the background knowledge of every argument of that event. This knowledge representation is implemented in the OWLizr base ontology, which is shown in Figure 3.

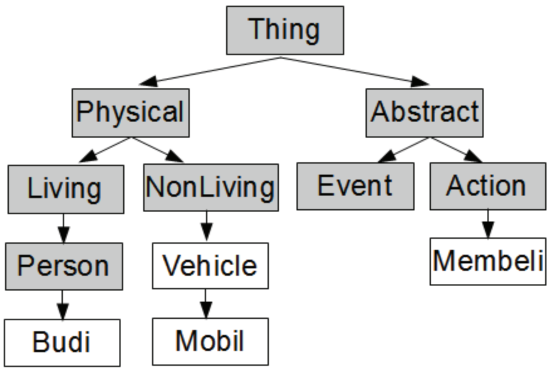

Fig. 3. OWLizr Base Ontology

DL knowledge is formed by a collection of classes, properties, and instances. From the example, we know that *"Budi"* is an instance from class *"Person"*. In turn, the class *"Person"* itself is a subclass of the *"Living"* class, where its definition is the class of living things, such as animals, plants, or people. The class can be built by using logical operators among classes, such as intersection, union, or negation. Then, the definition of *"Living"* or *"LivingPhysicalObject"* class will be:

$$LivingPhysicalObject \equiv Person \sqcup Animal \sqcup Plant$$

There are also various other classes, each of which has its own definition, e.g., Time, Process, AbstractTerm, Manner, Attribute, Quality, Quantity, Grade, and Intensity. Together they form all the classes in the base ontology.

Next, we build properties on the ontology according to the thematic roles of events. The technique is rather straightforward: we designate a specific DL property for every thematic role. The properties on DL can have domain and range. For example, the property *"hasAction"* has class *"Event"* for its domain and class *"Action"* for its range. For some properties, there are also inverse properties. For example, the property *"hasAction"* has as its inverse property *"isActionOf"*. The list of properties in the ontology is shown in Table I.

TABLE I
DL PROPERTIES

Property	Domain	Range
hasAction	Event	Action
hasAgent	Event	PhysicalObject, AbstractTerm, Process
hasPatient	Event	PhysicalObject, AbstractTerm, Process
hasLocation	Event, PhysicalObject	Location
isActionOf	Action	Event
isAgentOf	PhysicalObject, AbstractTerm, Process	Event
isPatientOf	PhysicalObject, AbstractTerm, Process	Event
isLocationOf	Location	Event, PhysicalObject

III. OWLIZR ARCHITECTURE

OWLizr consists of four modules, i.e. the NLP Semantic Analyzer, SPARQL Query Generator, KB Generator, and KB Reasoner. Each module works interdependently with each other. The NLP Semantic Analyzer acts as a text processing tool, extracts the semantic notations (also called canonical representations) from natural language text. The KB Generator transforms semantic notations into knowledge form using OWL-DL and the KB Reasoner discovers implicit knowledge from the knowledge base. Finally, the SPARQL Query Generator converts semantic notations from interrogative sentences into a SPARQL query and executes that query on the knowledge base. The system itself has two modes of operation, i.e. knowledge assertion mode and query mode. The difference between the two modes is in the modules which are invoked to process the semantic notations.

For the knowledge assertion mode, the modules involved are the NLP Semantic Analyzer, KB Generator, and KB Reasoner, whereas query mode involves only the NLP Semantic Analyzer and SPARQL Query Generator. The process on knowledge assertion mode works sequentially as follows: First, the text is processed by the NLP Semantic Analyzer, parsed and translated into semantic notations using syntax-driven semantic analysis. Next, the semantic notations are used as references for the KB Generator to assert knowledge. Finally, the KB Reasoner infers new sound knowledge from the asserted knowledge. The process on both the KB Generator and the KB Reasoner is highly dependent with the ontology model used by the system.

On the other hand, the process on the query mode works as follows: The question in natural language is processed by the NLP Semantic Analyzer, producing semantic notations which are divided into a question variable and conditional variables. The SPARQL Query Generator then translates the question variable into a SELECT clause and the conditional variables into WHERE clauses of the query. Finally, the query is executed and we can obtain the answer for the question.

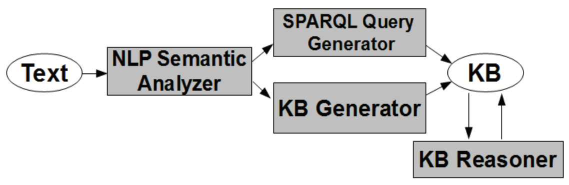

Fig. 4. OWLizr Architecture

The KB Generator and KB Reasoner cannot be set apart from the ontology for representing knowledge. We developed the OWLizr ontology as two components: the base ontology and the domain ontology. The base ontology is an ontology that contains base and general terms (see Section 2). The vocabulary concerns events, qualities, quantities, actions, and so on. The next component is the domain ontology. A domain ontology is an ontology that represents knowledge of some specific domains, such as economy, education, military, and many more. The base ontology acts as an upper ontology for the domain ontology. The base ontology can be extended with one or more domain ontologies. The ontology is represented in OWL-DL and developed using the *Protégé-OWL editor* with a top-down approach.

A. NLP Semantic Analyzer

The NLP Semantic Analyzer module reuses the semantic analyzer program in previous research [2]. It is used to parse the natural language text. Next, the parse results are used to produce semantic notations. The technique used is syntax-driven semantic analysis with lambda-calculus. The implementation is in the Prolog language. The module is divided into four parts: lexicon, grammar, lexical semantics, and semantic attachment rules. The lexicon contains a word list and relevant linguistic information of the words. The grammar specifies how to build sentences by structures and constituents via syntax. The lexical semantics stores semantics conveyed by individual words in the lexicon. Lastly, semantic attachment rules are instructions to build semantic representation based on the grammar rules.

The NLP Semantic Analyzer works in two modes, knowledge assertion mode and query mode. In the knowledge assertion mode, the input is the declarative sentence. The arguments of the semantic notations produced by this mode are defined completely without the question variable. For example, the result from semantic analysis on the sentence *"Pabrik memproduksi mobil"* or *"The factory produces the car"* is the following semantic notation:

[location(x5,pabrik), event(x1,memproduksi), agent(x1,x5), patient(x1,x6), objectx(x6,mobil)]

On the other hand, question answering mode involves interrogative sentences as the input, and produces semantic notations with one question variable inside an "ans" predicate. The question variable is usually reserved for an agent or patient. For example, the semantic notation for *"Apa yang memproduksi mobil?"* or *"What produces the car?"* is:

[ans(x8), objectx(x2,x8), event(x4,memproduksi), agent(x4,x2), patient(x4,x1), objectx(x1,mobil)]

B. *KB Generator*

The next module is the KB Generator. This module parses and transforms semantic notations from the NLP Semantic Analyzer into knowledge as OWL. The resulting knowledge form is very dependent with the ontology model of the system, i.e., the base ontology and domain ontology. The KB Generator is implemented using Java language and developed using the Eclipse IDE. There are two functions of the KB Generator, instance assertion and property assertion. Instance assertion is the process of asserting instances from semantic notations such as person, object, attribute, quality, and so on. Property assertion is the process of asserting properties that link two or more instances. The subset mapping from semantic notations into its knowledge form is shown in Tables II and III.

The KB Generator asserts knowledge based on semantic notations. The notations are processed based on the predicates. The KB Generator has two types of predicate lists according to their functions, one for instance assertions and the other one for property assertions. Instance assertion is executed first before property assertion. The processes are different between instance assertion and property assertion. The instance assertion process reads predicates and arguments of the notations, differentiates between

TABLE II
INSTANCE ASSERTION MAPPING TABLE

Semantic Notations	Knowledge Form
event(x, ActionName)	Event(event_ID), hasAction(event_ID, ActionName_action)
person(x, ID)	Person(ID)
objectx(x, ClassName)	ClassName(ClassName_ID)
attribute(x, AttributeName)	AttributeName(AttributeName_ID)
location(x, ID)	Location(ID)
location(x, LocationName)	LocationName(LocationName_ID)

TABLE III
PROPERTY ASSERTION MAPPING TABLE

Semantic Notations	Knowledge Form
agent(x, y)	has Agent(x, y)
patient(x, y)	hasPatient(x, y)
theme(x, y)	hasTheme(x, y)
attrib(x, y)	hasAttribute(x, y)
di(x, y)	hasLocation(x, y)

person, event, location, object, or other instance assertion predicates, invokes instances into the knowledge base, and then adds each instance in a hash table with the first argument as the key. This hash table is used as an index to maintain predicate-argument structur. For example, the expected result for the semantic notation
[location(x5,pabrik), event(x1,memproduksi), agent(x1,x5), patient(x1,x6), objectx(x6,mobil)] is:
Factory(factory_1), Event(event_1), Car(car_1)

Next, property assertion processes the predicates and arguments of predicates, differentiates between agent, patient, theme, or other property assertion predicates, and then invokes properties between instances in the knowledge base based on the hash table content by first and second arguments. The process of the property assertion is shown in Figure 5 below.

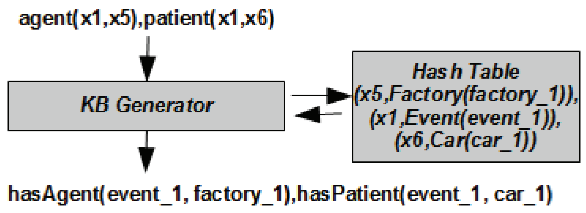
agent(x1,x5),patient(x1,x6)
KB Generator
Has h Table (x5,Factory(factory_1)), (x1,Event(event_1)), (x6,Car(car_1))
hasAgent(event_1, factory_1),hasPatient(event_1, car_1)

Fig. 5. Property Assertion Process

C. *KB Reasoner*

The KB Reasoner has two main uses; consistency checking and reasoning about implicit knowledge. There is the possibility that the knowledge asserted is not consistent. For example, *"Mobil membeli radio"* or *"The car buys the radio"* will produce an error because the domain ontology states that a car cannot buy something – a reflection of so-called common sense knowledge, often referred to as a *selectional restriction*. If we insist to assert this knowledge, the knowledge base will be inconsistent. The consistency checking is implemented using the Protege OWL API *computeInconsistentConcepts()* function. The module will check consistency after each property assertion has been invoked on the knowledge base.

The second feature is reasoning. After the knowledge assertion process, the KB Reasoner will perform its function to obtain inferred knowledge. The inference type is similar to instance checking on Description Logic. The module reuses the function *getIndividualsBelongingToClass()* from the Protégé-OWL API. Finally, the output, which contains knowledge from the text with inferred knowledge, is successfully produced by OWLizr. Both consistency checking and reasoning functions are executed on Pellet, a free OWL DL Reasoner implemented on Java.

D. SPARQL Query Generator

The SPARQL Query Generator module translates semantic notations into SPARQL query, which is formed by two components, a SELECT clause and a WHERE clause. The SELECT clause lists the variables to appear in the query results, and the WHERE clause provides the basic graph pattern to match against the data graph [9]. The module reuses the Protégé-OWL API function, i.e., *executeSPARQLQuery()* to execute the query. The whole process specifically works as follows:

1) The module translates semantic notations from the NLP Semantic Analyzer. The results from the NLP Semantic Analyzer consist of a question variable, which is inside the "ans" predicate, and conditional variables, which is inside other predicates other than the "ans" predicate. Next, the question variable is translated into a SELECT clause and the conditional variables into the WHERE clause.

2) The module concatenates both SELECT clause and WHERE clause, and then executes it. The results will be given according to the query. Then, the query process from the example in Section 3A is shown in Figure 6.

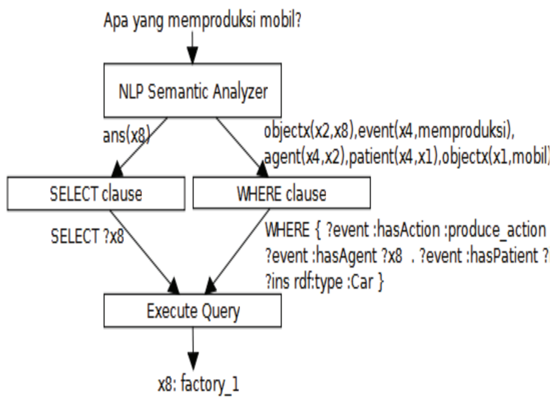
Apa yang memproduksi mobil?
NLP Semantic Analyzer
ans(x8)
objectx(x2,x8),event(x4,memproduksi),
agent(x4,x2),patient(x4,x1),objectx(x1,mobil)
SELECT clause
WHERE clause
SELECT ?x8
WHERE { ?event :hasAction :produce_action
?event :hasAgent ?x8 . ?event :hasPatient ?
?ins rdf:type :Car }
Execute Query
x8: factory_1

Fig. 6. Query Process on OWLizr

IV. Evaluation Results

Our evaluation serves as a proof-of-concept for the architecture, tests the ability to assert knowledge from natural language text and to infer knowledge from various ontological features. We developed a domain-specific ontology for our evaluation process. The topic is about *simple economic activities*. We chose this domain because it is common and rather easy to understand.

The domain ontology is developed iteratively and manually by listing some terms of the domain, such as "price" or "harga", "expensive" or "mahal", "buy" or "membeli", "sell" or "menjual", "buyer" or "pembeli", and "shop" or "toko", and then, implementing those terms in OWL DL form. The domain ontology extends the base ontology. Our domain ontology was also carefully designed to highlight the inferential power of OWL DL. We implemented ontology features such as subclass, intersection, union, and other features on class and property definitions. For example, a buyer is a person who buys something. So we define the buyer in the ontology as:

```
Buyer = Person and (isAgentOf some (hasAction some Buy))
```

A. Knowledge Assertion Mode

After that, we tested the system to process natural language text. The knowledge retrieval process consists of two stages, assertion and inference. For example, the retrieval process for the sentence *"Anto buys the car in the shop"* or *"Anto membeli mobil di toko"* are shown in the points below.

1) The NLP Semantic Analyzer processes the sentence. The output is the following semantic notation: [person(x6,anto), event(x4,membeli), agent(x4,x6), patient(x4,x3), objectx(x3,mobil), di(x4,x1), location(x1,toko)]".

2) The system then processes the output from the NLP Semantic Analyzer module. After this, there will exist pairs [(x6, Person(anto)), (x4, Event(event_1)), (x1, Shop(shop_1)), (x3, Car(car_1))] in the hash table. The program will then produce the instance assertion output on the console, as follows:

```
ASSERTED KNOWLEDGE:
Instance

Person(anto)
Event(event_1)
Car(car_1)
Shop(shop_1)
```

3) Next, the system asserts the properties. The consistency checking function is also executed after each assertion. There is no error, so the system will produce the following output:

```
ASSERTED KNOWLEDGE:
Property

hasAction(event_1,buy_action)
hasAgent(event_1,anto)
hasPatient(event_1,car_1)
hasLocation(event_1,shop_1)
```

4) Lastly, the KB Reasoner infers some new knowledge:

```
AbstractObject(event_1), Buyer(anto),
Product(car_1), PhysicalObject(anto),
PhysicalObject(shop_1),
PhysicalObject(car_1), Store(shop_1),
AutomotiveStore(shop_1),
NonLivingPhysicalObject(shop_1),
NonLivingPhysicalObject(car_1),
Location(shop_1), AutomotiveProduct(car_1),
LivingPhysicalObject(anto), Thing(event_1),
Thing(anto), Thing(car_1), Thing(shop_1)
```

The inferred knowledge is acquired by processing the ontology features with the asserted knowledge. From the result above, the instance *"anto"* can be included in the class *"Buyer", "LivingPhysicalObject", "PhysicalObject",* and *"Thing"*. There are reasons for these cases. *"anto"* is in the *"Buyer"* class because *"anto"* is a person who has action *"Buy"*. Thus, it satisfies the intersection of *"Person"* class and *"(isAgentOf some (hasAction some Buy)"* anonymous class. *"anto"* is also included in the *"LivingPhysicalObject", "PhysicalObject",* and *"Thing"* classes because the *"Person"* class in which *"anto"* resides is the subclass from those classes. So, the ontology features used are intersection and subclass reasoning.

B. Query Mode

When we have executed assertion and knowledge reasoning, we can now ask questions by operating query mode. For example, if the question is *"Siapa yang membeli mobil?"* or *"Who buys the car?",* the process will be:

1) The NLP Semantic Analyzer processes the question. The input is *"Siapa yang membeli mobil?"* and the given output is the semantic notation of the question, i.e., [ans(x7), person(x5,x7), event(x4,membeli), agent(x4,x5), patient(x4,x2), objectx(x2,mobil)]".

2) The SPARQL Query Generator then translates the notation into SPARQL query form. The module transforms "ans(x7)" into SELECT clause, i.e., "SELECT ?x7", and the rest of the notation into the WHERE clause, i.e., "WHERE { ?event :hasAction :buy_action . ?event :hasAgent ?x7 . ?event :hasPatient ?ins . ?ins rdf:type :Car }". Next, the query is concatenated and then executed. Finally, the system returns the output "x7:Anto".

V. CONCLUSIONS

This research combines Natural Language Processing (NLP) and Description Logic (DL) to build OWLizr, a system to retrieve knowledge from texts written in bahasa Indonesia. The value of this research is not more on the experimental side, but rather formalization attempt to natural language text. The system can produce inferred knowledge by processing ontology features, such as subclass, equivalence, intersection, union, and negation, on the asserted and inferred knowledge. The final result of the retrieval process will be represented in the form of an OWL DL knowledge base. The system also supports natural language question-answering, and uses SPARQL for querying the facts on the knowledge base. For future work, we plan to implement TBox assertion, so that we can construct a domain ontology automatically from natural language texts. We also consider integrating existing ontologies, e.g., SUMO, Cyc, or DOLCE, in order to be implemented on OWLizr. For the sake of expressivity, we can also increase the ontology features up to OWL 2.

REFERENCES

[1] S. D. Larasati and R. Manurung, "Towards a Semantic Analysis of Bahasa Indonesia for Question Answering," in *Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics (PACLING 2007)*, Melbourne, Australia, 19-21 September 2007

[2] R. Mahendra, S. D. Larasati, and Ruli Manurung, "Extending an Indonesian Semantic Analysis-based Question Answering System with Linguistic and World Knowledge Axioms," in *Proceedings of the 22nd Pacific Asia Conference on Language, Information, and Computation (PACLIC 2008)*, pp.262-271, Cebu, Philippines, 20-22 November 2008.

[3] Hobbs, J. (1985). "Ontological Promiscuity". *Proceedings of the 23rd Annual Meeting of the Association of Computational Linguistics*, (pp. 61-69)

[4] Parsons, T. (1990). *Events in the Semantics of English: A Study in Subatomic Semantics (Learning, Development, and Conceptual Change)*. MIT Press.

[5] Manurung, H. M. (2007). Computational Semantics : Processing Meaning from Natural Language Utterances. Represented on Summer School on Computational Logic and Logic Foundations of Computer Science, Hanoi University of Technology, 31 July - 7 August.

[6] Baader, F., Calvanese, D., McGuinness, D., & Patel-Schneider, P. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge: Cambridge University Press.

[7] McGuinness, D. L., & van Harmelen, F. (2004, February 10). OWL Web Ontology Language Overview. Retrieved June 14, 2010, from W3C: http://www.w3.org/TR/owl-features/

[8] Franconi, E., & Rabito, V. (1994). *A Relation-Based Description Logic*. In Working Notes of the 1994 Description Logic Workshop, (pp. 55-60). Bonn, Germany

[9] Prud'hommeaux, E., & Seaborne, A. (2008, January 15). SPARQL Query Language for RDF. Retrieved June 14, 2010, from W3C: http://www.w3.org/TR/rdf-sparql-query