

TESIS

**SISTEM PENCARIAN INFORMASI BERBASIS ONTOLOGI UNTUK
JALUR PENDAKIAN GUNUNG MENGGUNAKAN QUERY BAHASA
ALAMI DENGAN PENYAJIAN PETA INTERAKTIF**

***INFORMATION RETRIEVAL SYSTEM BASED ON ONTOLOGY FOR
MOUNTAIN CLIMBING TRACK USING NATURAL LANGUAGE QUERY
WITH INTERACTIVE MAP PRESENTATION***



FADHILA TANGGUH ADMOJO

10/310730/PPA/03452

**PROGRAM STUDI S2 ILMU KOMPUTER
JURUSAN ILMU KOMPUTER DAN ELEKTRONIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS GADJAH MADA
YOGYAKARTA**

2015

TESIS

**SISTEM PENCARIAN INFORMASI BERBASIS ONTOLOGI UNTUK
JALUR PENDAKIAN GUNUNG MENGGUNAKAN QUERY BAHASA
ALAMI DENGAN PENYAJIAN PETA INTERAKTIF**

***INFORMATION RETRIEVAL SYSTEM BASED ON ONTOLOGY FOR
MOUNTAIN CLIMBING TRACK USING NATURAL LANGUAGE QUERY
WITH INTERACTIVE MAP PRESENTATION***

Diajukan untuk memenuhi salah satu syarat memperoleh derajat

Master of Computer Science



FADHILA TANGGUH ADMOJO

10/310730/PPA/03452

**PROGRAM STUDI S2 ILMU KOMPUTER
JURUSAN ILMU KOMPUTER DAN ELEKTRONIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS GADJAH MADA
YOGYAKARTA**

2015

HALAMAN PENGESAHAN

TESIS

SISTEM PENCARIAN INFORMASI BERBASIS ONTOLOGI UNTUK JALUR PENDAKIAN GUNUNG MENGGUNAKAN QUERY BAHASA ALAMI DENGAN PENYAJIAN PETA INTERAKTIF

Telah dipersiapkan dan disusun oleh

Fadhila Tangguh Admojo
10/310730/PPA/03452

telah dipertahankan di depan Dewan Pengaji
pada tanggal
9 Juni 2015

Susunan Dewan Pengaji

Pembimbing Utama

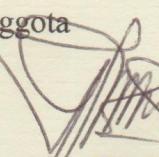

Drs. Edi Winarko M.Sc., Ph.D
NIP. 196302231987031002

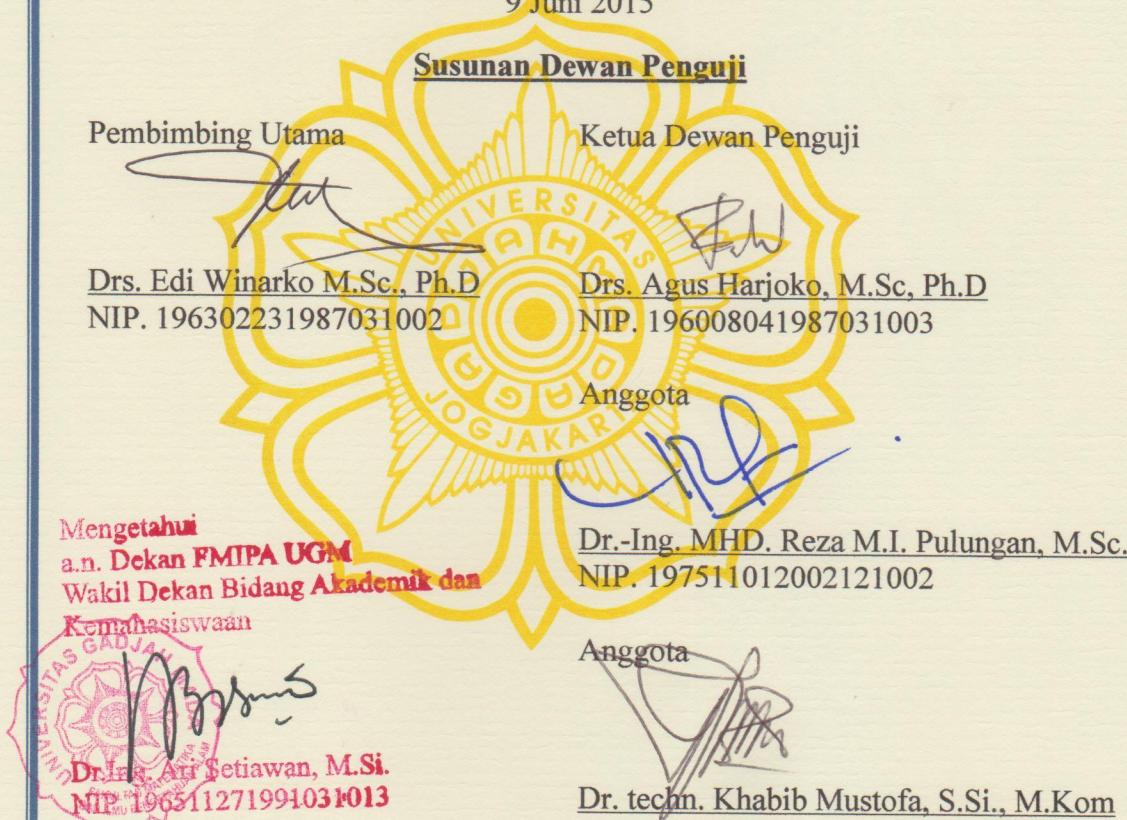

Drs. Agus Harjoko, M.Sc, Ph.D
NIP. 196008041987031003

Anggota

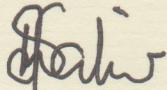

Dr.-Ing. MHD. Reza M.I. Pulungan, M.Sc.
NIP. 197511012002121002

Anggota


Dr. techn. Khabib Mustofa, S.Si., M.Kom
NIP. 197207221998031002



Tesis ini telah diterima sebagai salah satu persyaratan
untuk memperoleh gelar *Master of Computer Science*
Tanggal, 15 Juni 2015



Prof. Sri Hartati, M.Sc., Ph.D

Pengelola Program Studi Monodisiplin S2 Ilmu Komputer

PERNYATAAN

Dengan ini saya menyatakan bahwa dalam Tesis ini tidak terdapat karya yang pernah diajukan untuk memperoleh gelar Master di suatu Perguruan Tinggi, dan sepanjang pengetahuan saya juga tidak terdapat karya atau pendapat yang ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis diacu dalam naskah ini dan disebutkan dalam daftar pustaka.

Yogyakarta, April 2015

FADHILA TANGGUH ADMOJO

HALAMAN PERSEMBAHAN

Untuk para pendaki gunung Indonesia

PRAKATA

Puji syukur atas petunjuk dan karunia Allah SWT akhirnya penulis dapat menyelesaikan penelitian ini. Penelitian ini juga tidak mungkin dapat diselesaikan tanpa dukungan dan bantuan dari berbagai pihak. Oleh karena itu dengan segala kerendahan hati penulis ingin mengucapkan rasa terima kasih yang sedalam-dalamnya kepada:

1. Ayahanda Sugiarto Purnomo Yadi (Alm) dan Ibunda Marhumah yang selalu mendukung setiap ambisi dan mimpi penulis (termasuk penelitian ini).
2. Adik-adik (Asti – Surya, Tyo – Nova, Nifa) yang selalu memberikan semangat untuk cepat menyelesaikan penelitian ini.
3. Drs. Edi Winarko M.Sc., Ph.D yang telah membimbing dan mengarahkan penulis selama melakukan penelitian.
4. Seluruh dosen di Program Studi Magister Ilmu Komputer UGM yang telah mengajarkan ilmu dan pengetahuan baru.
5. Teman-teman yang telah memberikan makna 5 tahun perjalanan penulis selama menempuh pendidikan di Jogja. Teman-teman Pra S2 2010 Genap (Arsa, Arfan, Chris, Cornelis, Derwin, Dwi, Erma, Gana, Hamid, Imin, Leo, P. Tarigan, Yoga, Zen). Teman-teman MCS angkatan 2010 (Ikbal, Maya, Nova, Pte, Zaky). Teman-teman MCS angkatan 2011 (Adit, Akin Anna, Awink-Ayu, Effan, Evin, Hamidudin, Sidhi, Winaya - Hanin). Teman-teman MCS angkatan 2013 (Noe, Sukron, Vida). Maaf apabila ada nama-nama yang terlewat disebutkan.
6. Pak Sugeng, Mbak Rini, Mas Kuncoro atas pelayanan dan kerja samanya selama penulis menempuh perkuliahan.
7. Ummu yang telah menambah lika-liku hari penulis selama melakukan penelitian.
8. Para pahlawan yang telah berjuang mengorbankan jiwa dan raganya untuk memerdekakan bangsa ini sehingga penulis bisa menikmati kemerdekaan.

Penulis menyadari di dalam penelitian ini terdapat banyak kekurangan, apabila terdapat pertanyaan atas hal-hal yang kurang jelas dapat mengirimkan email ke fadhila.tangguh@yahoo.com.

Semoga penelitian ini dapat bermanfaat bagi siapa saja yang membacanya.

Yogyakarta, 2015

(Penulis)

Fadhila Tangguh Admojo

DAFTAR ISI

HALAMAN PENGESAHAN	iii
HALAMAN PERNYATAAN	iv
HALAMAN PERSEMPAHAN	v
PRAKATA	vi
DAFTAR ISI	viii
DAFTAR TABEL	x
DAFTAR GAMBAR	xi
INTISARI	xv
ABSTRACT	xvi
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	6
1.3 Batasan Masalah	7
1.4 Tujuan Penelitian	7
1.5 Manfaat Penelitian	7
1.6 Metodologi Penelitian	8
1.7 Sistematika Penulisan	9
BAB II TINJAUAN PUSTAKA	11
BAB III LANDASAN TEORI	17
3.1 Bahasa	17
3.1.1 Kata	17
3.1.2 Frasa	20
3.1.3 Klaus	22
3.1.4 Kalimat	22
3.2 NLP (<i>Natural Language Processing</i>)	24
3.3 Ontologi	26
3.3.1 Komponen ontologi	28
3.3.2 Metode pengembangan ontologi	29
3.4 Semantik <i>Web</i>	31
3.4.1 Arsitektur <i>web</i> semantik	32
3.4.2 XML	34
3.4.3 RDF (<i>Resource Description Framework</i>)	35
3.4.4 RDFS (<i>RDF Schema</i>)	38
3.4.5 OWL (<i>Ontology Web Language</i>)	39
3.4.6 SPARQL	41
3.5 Peta	44
3.6 KML (<i>Keyhole Markup Language</i>)	45

BAB IV ANALISIS DAN RANCANGAN SISTEM.....	49
4.1 Gambaran Umum Sistem.....	49
4.2 Desain Sistem	50
4.2.1 Penerimaan <i>input</i>	50
4.2.2 <i>Preprocessing</i>	51
4.2.3 Pemroses Bahasa	52
4.2.4 Penyusun Informasi.....	54
4.3 Perancangan Ontologi.....	55
4.3.1 Rancangan ontologi Bahasa	56
4.3.2 Rancangan ontologi <i>Mountaineering</i>	59
4.4 Perancangan Data KML	68
4.5 Perancangan Sistem dengan UML	69
4.6 Perancangan Antarmuka.....	73
BAB V IMPLEMENTASI.....	75
5.1 Implementasi Ontologi	75
5.1.1 Implementasi ontologi Bahasa	75
5.1.2 Implementasi ontologi <i>Mountaineering</i>	79
5.2 Implementasi KML.....	84
5.3 Implementasi Aplikasi.....	86
5.3.1 Penerimaan <i>input</i> dan fitur <i>spelling checker</i>	86
5.3.2 <i>Input handling</i> dan <i>preprocessing</i>	91
5.3.3 Pembentukan <i>parse tree</i> dengan pengecekan aturan gramatikal.....	98
5.3.4 Ekstraksi informasi semantik	107
5.3.5 Pencarian informasi.....	109
5.3.6 Penyajian informasi.....	112
5.3.7 Implementasi KML parser	114
5.4 Implementasi Antarmuka Aplikasi	115
BAB VI HASIL DAN PEMBAHASAN	118
6.1 Hasil Implementasi Sistem	118
6.1.1 Fitur <i>spelling checker</i>	119
6.1.2 Kemampuan sistem menggunakan <i>thesaurus</i> kata.....	121
6.1.3 Kemampuan sistem memahami <i>input</i> bahasa alami	121
6.1.4 Kemampuan sistem mendeteksi <i>input</i> yang tidak sesuai dengan kaidah tata bahasa Indonesia	132
6.2 Pengujian Kuantitatif.....	135
BAB VII KESIMPULAN DAN SARAN	138
7.1 Kesimpulan.....	138
7.2 Saran	139
DAFTAR PUSTAKA	140
LAMPIRAN A HASIL SURVEI PENDAKI	145
LAMPIRAN B ATURAN GRAMATIKAL.....	148
LAMPIRAN C <i>INPUT</i> DARI RESPONDEN	158

DAFTAR TABEL

Tabel 2.1 Rangkuman perbandingan penelitian	15
Tabel 4.1 Relasi antar <i>class</i> pada ontologi Bahasa	57
Tabel 4.2 <i>Property</i> untuk mendeskripsikan konsep kata	58
Tabel 4.3 <i>Property</i> untuk mendeskripsikan informasi semantik.....	59
Tabel 4.4 Hubungan antar <i>class</i> utama pada ontologi <i>Mountaineering</i>	60
Tabel 4.5 <i>Property</i> untuk mendeskripsikan konsep gunung	60
Tabel 4.6 <i>Property</i> untuk mendeskripsikan konsep kawah.....	61
Tabel 4.7 <i>Property</i> untuk mendeskripsikan konsep jalur pendakian	62
Tabel 4.8 <i>Subclass</i> dari <i>class ClimbingPoints</i>	62
Tabel 4.9 <i>Property</i> untuk mendeskripsikan konsep <i>basecamp</i>	63
Tabel 4.10 <i>Property</i> yang digunakan untuk mendeskripsikan konsep pos	63
Tabel 4.11 <i>Property</i> untuk menjabarkan konsep <i>campground</i>	64
Tabel 4.12 <i>Property</i> untuk mendeskripsikan konsep sumber air	64
Tabel 4.13 <i>Property</i> yang digunakan untuk mendeskripsikan konsep puncak	65
Tabel 4.14 <i>Property</i> untuk mendeskripsikan konsep titik unik.....	65
Tabel 4.15 <i>Subclass</i> dari <i>class AdministrativeRegions</i>	66
Tabel 4.16 <i>Property</i> untuk mendeskripsikan konsep kabupaten.....	66
Tabel 4.17 <i>Property</i> untuk mendeskripsikan konsep provinsi	66
Tabel 4.18 <i>Property</i> untuk mendeskripsikan konsep orang	67
Tabel 4.19 Struktur <i>tag</i> yang digunakan untuk mendeskripsikan <i>placemark</i>	69
Tabel 4.20 Struktur <i>tag</i> yang digunakan dalam fitur <i>polyline</i>	69
Tabel 6.1 <i>Input</i> yang dipilih untuk melihat kemampuan sistem	119
Tabel 6.2 <i>Input</i> dari responden yang tidak dapat diproses oleh sistem	136

DAFTAR GAMBAR

Gambar 1.1 Ilustrasi pencarian informasi jalur pendakian di internet	2
Gambar 1.2 Informasi peta jalur pendakian berbentuk <i>file</i> citra kompresi (www.merbabu.com)	4
Gambar 1.3 Perbandingan informasi jalur pendakian gunung merapi dari <i>Google Maps</i> dengan data hasil observasi (diakses tanggal 27 Februari 2015)	5
Gambar 3.1 Klasifikasi kata dalam sebuah kalimat	23
Gambar 3.2 Arsitektur <i>web</i> semantik (Berners-Lee, 2006).....	32
Gambar 3.3 Contoh dokumen XML (Antoniou dan van Harmellen, 2008).....	34
Gambar 3.4 Contoh dokumen XML <i>Schema</i> (Antoniou dan van Harmellen, 2008)	35
Gambar 3.5 Contoh dokumen XML <i>Namespace</i> (Antoniou dan van Harmellen, 2008).....	35
Gambar 3.6 Contoh RDF <i>statement</i> yang dinyatakan dengan notasi RDF <i>triple</i> (Antoniou dan van Harmellen, 2008)	37
Gambar 3.7 Contoh RDF <i>statement</i> yang dinyatakan dengan notasi RDF <i>graph</i> (Antoniou dan van Harmellen, 2008)	37
Gambar 3.8 Contoh RDF <i>statement</i> yang dinyatakan dalam notasi RDF/XML (Antoniou dan van Harmellen, 2008)	37
Gambar 3.9 Contoh definisi <i>class</i> dengan menggunakan kosakata RDFS yang berbasis notasi RDF/XML (Yu, 2010)	38
Gambar 3.10 Contoh <i>query</i> SPARQL dalam bentuk SELECT (Yu, 2010).....	42
Gambar 3.11 Contoh <i>query</i> SPARQL, CONSTRUCT (Yu, 2010)	43
Gambar 3.12 Contoh <i>query</i> SPARQL, DESCRIBE (Yu, 2010)	43
Gambar 3.13 Contoh dari <i>query</i> SPARQL dalam bentuk ASK (Yu, 2010)	44
Gambar 3.14 Contoh penggunaan <i>placemark</i>	46
Gambar 3.15 Contoh penggunaan <i>Ground Overlay</i>	47
Gambar 3.16 Contoh penggunaan <i>path/polyline</i>	47
Gambar 3.17 Contoh penggunaan <i>polygon</i>	48
Gambar 4.1 Ilustrasi tahapan kerja sistem.....	49
Gambar 4.2 Arsitektur sistem.....	50
Gambar 4.3 Ilustrasi proses <i>Spelling Checker</i>	51
Gambar 4.4 Ilustrasi <i>Preprocessing</i>	52
Gambar 4.5 Ilustrasi komponen Pemroses Bahasa	53
Gambar 4.6 Ilustrasi pembentukan <i>parse tree</i>	53
Gambar 4.7 Ilustrasi pembentukan <i>query</i> SPARQL	54
Gambar 4.8 Ilustrasi komponen Penyusun Informasi	55

Gambar 4.9 <i>Class-class</i> pada ontologi Bahasa	57
Gambar 4.10 Relasi <i>class</i> yang ada dalam ontologi <i>Mountaineering</i>	67
Gambar 4.11 Ilustrasi pengambilan geodata dari KML	68
Gambar 4.12 Penggunaan <i>tag <ExtendedData></i>	69
Gambar 4.13 <i>Usecase diagram</i>	70
Gambar 4.14 <i>Activity diagram</i>	72
Gambar 4.15 Rancangan halaman <i>input</i>	73
Gambar 4.16 Rancangan halaman <i>output</i>	74
Gambar 5.1 Implementasi URI pada ontologi Bahasa	75
Gambar 5.2 Implementasi <i>class</i> pada ontologi Bahasa.....	75
Gambar 5.3 Implementasi <i>disjoint</i> pada ontologi Bahasa	76
Gambar 5.4 Implementasi <i>property</i> pada ontologi Bahasa (a) <i>data property</i> . (b) <i>object property</i>	77
Gambar 5.5 Implementasi kardinalitas pada ontologi Bahasa	77
Gambar 5.6 Contoh deskripsi <i>instance gunung2</i> pada <i>class Lexicon</i>	78
Gambar 5.7 Implementasi <i>instance gunung2</i> dari <i>class Lexicon</i> pada ontologi Bahasa menggunakan <i>tool Protege</i>	78
Gambar 5.8 Hasil pemeriksaan inkonsistensi pada ontologi Bahasa	79
Gambar 5.9 Implementasi URI ontologi <i>Mountaineering</i>	79
Gambar 5.10 Implementasi <i>class</i> pada ontologi <i>Moutnaineering</i>	80
Gambar 5.11 Implementasi relasi <i>disjoint class</i> pada ontologi <i>Mountaineering</i> ..	81
Gambar 5.12 Implementasi <i>property</i> pada ontologi <i>Mountaineering</i> (a) <i>data</i> <i>property</i> . (b) <i>object property</i>	81
Gambar 5.13 Implementasi kardinalitas pada ontologi <i>Mountaineering</i>	82
Gambar 5.14 Contoh deskripsi <i>instance Merapi</i> pada <i>class Mountain</i>	83
Gambar 5.15 Implementasi <i>instance Merapi</i> dari <i>class Mountain</i> menggunakan <i>tool Protege</i>	83
Gambar 5.16 Hasil pemeriksaan inkonsistensi pada ontologi <i>Mountaineering</i>	84
Gambar 5.17 Cuplikan data KML berupa <i>placemark</i> untuk titik (<i>point</i>).....	85
Gambar 5.18 Cuplikan data KML berupa <i>placemark</i> untuk jalur (<i>linestring</i>).....	85
Gambar 5.19 Cuplikan kode dari halaman <i>utama.jsp</i>	86
Gambar 5.20 Cuplikan kode dari <i>script gb.js</i>	87
Gambar 5.21 <i>method doGet</i> di dalam <i>class wordchecker</i>	87
Gambar 5.22 <i>method checkSpelling</i> di dalam <i>class wordchecker</i>	88
Gambar 5.23 <i>Class BahasaGenerator</i>	89
Gambar 5.24 Cuplikan kode <i>method getAllKata</i> (a) <i>class OntoBahasaReader</i> . (b) <i>class MountaineeringReader</i>	90
Gambar 5.25 <i>method doGet</i>	91
Gambar 5.26 <i>method GetStringArrayFromString</i>	92
Gambar 5.27 <i>Handling</i> untuk <i>non-alphanumeric</i>	92

Gambar 5.28 <i>Handling</i> untuk <i>input</i> kosong	92
Gambar 5.29 <i>Handling</i> untuk spasi lebih dari satu	92
Gambar 5.30 <i>method ValidateInput</i> pada <i>class Preprocessor</i>	93
Gambar 5.31 (a) <i>method CheckIfWordInBahasa</i> . (b) <i>method</i> <i>CheckIfWordInGunung</i>	94
Gambar 5.32 Cuplikan hasil proses validasi	95
Gambar 5.33 (a) <i>method createSimpleQuery</i> . (b) <i>runRawQuery</i> . (c) <i>simpleResultSetToListBahasa</i>	96
Gambar 5.34 <i>Method queryTemplate</i>	97
Gambar 5.35 Cuplikan <i>list</i> hasil <i>preprocessing</i>	97
Gambar 5.36 <i>Method generateResponseMessage</i>	98
Gambar 5.37 Cuplikan hasil <i>preprocessing</i> untuk <i>input</i> yang tidak dikenali	98
Gambar 5.38 <i>Method cekSintaksis</i>	99
Gambar 5.39 Pembentukan <i>Parse Tree</i>	100
Gambar 5.40 Cuplikan kode dari <i>Method siValidSintax</i>	101
Gambar 5.41 (a) <i>Method checkLeftFN</i> implementasi aturan gramatikal pembentuk <i>FRASA_NOMINAL</i> . (b) Aturan gramatikal gramatikal pembentuk <i>FRASA_NOMINAL</i>	101
Gambar 5.42 (a) <i>Method generatedAccepted</i> untuk menggabungkan dua kata yang membentuk <i>FRASA_NOMINAL</i> . (b) aturan gramatikal penggabungan dua kata yang membentuk <i>FRASA_NOMINAL</i>	103
Gambar 5.43 Cuplikan hasil proses pengecekan urutan kata	104
Gambar 5.44 Ilustrasi proses pengecekan menggunakan aturan gramatikal.....	105
Gambar 5.45 <i>method checkStack</i>	106
Gambar 5.46 Cuplikan kode untuk mengirimkan pesan bertipe <i>ERROR_SYNTAX</i> kepada halaman <i>tampiltoken.jsp</i>	107
Gambar 5.47 <i>method getQueryLogic</i>	108
Gambar 5.48 Ilustrasi pembentukan <i>query SPARQL</i> oleh <i>method getQueryLogic</i>	108
.....	
Gambar 5.49 Cuplikan hasil ekstraksi dari <i>method getQueryLogic</i>	109
Gambar 5.50 Implementasi eksekusi <i>query SPARQL</i>	109
Gambar 5.51 Cuplikan kode pada <i>class Token</i> untuk mengirimkan	110
Gambar 5.52 Cuplikan kode <i>getStringDesk</i>	111
Gambar 5.53 Cuplikan kode <i>getCuaca</i>	111
Gambar 5.54 Cuplikan kode (a) <i>getStringPmId</i> . (b) <i>getStringMaps</i>	112
Gambar 5.55 Cuplikan kode pada halaman <i>tampiltoken.jsp</i>	113
Gambar 5.56 Cuplikan kode dari <i>parserKML.js</i>	114
Gambar 5.57 Modifikasi <i>library geoxml3</i> untuk melakukan <i>parsing</i> <i><ExtendedData></i>	115
Gambar 5.58 Tampilan halaman <i>utama.jsp</i>	115

Gambar 5.59 Tampilan halaman <i>tampiltoken.jsp</i>	117
Gambar 6.1 Tampilan fitur <i>spelling checker</i> ketika memberikan saran perbaikan kata.....	120
Gambar 6.2 Cuplikan proses <i>spelling checker</i>	120
Gambar 6.3 Hasil pencarian dengan penggunaan <i>thesaurus</i> kata	121
Gambar 6.4 Hasil pencarian informasi menggunakan <i>input</i> berupa frasa	122
Gambar 6.5 Cuplikan proses untuk <i>input</i> yang mengandung satuan sintaksis berupa frasa.....	123
Gambar 6.6 Hasil pencarian informasi menggunakan <i>input</i> yang mengandung klausा	124
Gambar 6.7 Cuplikan proses untuk <i>input</i> yang mengandung satuan sintaksis berupa klausа	124
Gambar 6.8 Hasil pencarian infromasi menggunakan <i>input</i> berupa kalimat tunggal berpola S-P	126
Gambar 6.9 Cuplikan proses pemahaman sistem untuk <i>input</i> berupa kalimat tunggal berpola S-P	126
Gambar 6.10 Hasil pencarian informasi menggunakan <i>input</i> berupa kalimat tunggal berpola P-S	127
Gambar 6.11 Cuplikan proses untuk <i>input</i> berupa kalimat tunggal berpola berpola P-S	127
Gambar 6.12 Hasil pencarian untuk <i>input</i> berupa kalimat tanya tunggal berpola S-P-O	128
Gambar 6.13 Cuplikan proses untuk <i>input</i> berupa kalimat tunggal berpola S-P-O	129
Gambar 6.14 Cuplikan hasil pencarian untuk <i>input</i> berupa kalimat tunggal berpola P-O-S	129
Gambar 6.15 Cuplikan proses untuk <i>input</i> berupa kalimat tunggal berpola P-O-S	130
Gambar 6.16 Cuplikan hasil pencarian untuk <i>input</i> kalimat majemuk bertingkat berpola S-P-O-K	131
Gambar 6.17 Cuplikan hasil pencarian untuk <i>input</i> kalimat majemuk bertingkat berpola K-S-P-O	131
Gambar 6.18 Cuplikan hasil pencarian menggunakan analisis sintaksis dan semantik	132
Gambar 6.19 Cuplikan pesan untuk kesalahan sintaksis.....	133
Gambar 6.20 Cuplikan proses untuk kesalahan sintaksis.....	133
Gambar 6.21 Cuplikan pesan untuk kesalahan semantik	134
Gambar 6.22 Cuplikan pesan kesalahan untuk input yang tidak lengkap.....	135

INTISARI

SISTEM PENCARIAN INFORMASI BERBASIS ONTOLOGI UNTUK JALUR PENDAKIAN GUNUNG MENGGUNAKAN QUERY BAHASA ALAMI DENGAN PENYAJIAN PETA INTERAKTIF

Oleh

FADHILA TANGGUH ADMOJO
10/310730/PPA/03452

Tingginya minat masyarakat Indonesia dalam melakukan kegiatan mendaki gunung menyebabkan meningkatnya kebutuhan akan informasi jalur pendakian. Sulitnya mendapatkan literatur tentang jalur pendakian gunung di Indonesia dalam bentuk buku menjadikan internet media utama dalam mencari informasi jalur pendakian.

Mencari informasi jalur pendakian gunung di internet memunculkan suatu permasalahan baru dikalangan para pendaki, yaitu hasil pencarian yang banyak dan beragam, sehingga membutuhkan waktu lebih untuk menelusuri dan membandingkan setiap informasi. Selain itu banyaknya informasi yang didapat di Internet justru membingungkan para pendaki.

Penelitian ini bertujuan untuk memberikan solusi dari permasalahan yang dihadapi pendaki dengan mengembangkan sistem pencarian informasi jalur pendakian gunung menggunakan pendekatan berbasis ontologi.

Sistem dikembangkan dengan menggunakan dua buah ontologi, yaitu ontologi Bahasa sebagai representasi pengetahuan linguistik dan ontologi *mountaineering* sebagai representasi pengetahuan pendakian gunung. Sistem dirancang untuk dapat melakukan pencarian informasi dengan kemampuan memahami *input* bahasa alami berupa kata, frasa, klausa dan kalimat. Proses pemahaman bahasa alami didasarkan pada analisis sintaksis dan analisis semantik menggunakan aturan-aturan tata bahasa Indonesia baku. Sistem juga dilengkapi dengan kemampuan untuk menyajikan informasi hasil pencarian ke dalam peta interaktif.

Berdasarkan hasil penelitian yang telah dilakukan, sistem mampu memahami *input* bahasa alami secara sintaksis dan secara semantik. Sistem juga mampu menggunakan *thesaurus* dalam melakukan proses pencarian dan mampu mendeteksi *input* yang tidak sesuai dengan kaidah tata bahasa Indonesia. Hasil pengujian kuantitatif menunjukkan sistem dapat memproses 69% *input* yang diambil secara acak dari responden.

Kata kunci: pencarian informasi, pendakian gunung, ontologi, web semantik, NLP, *parsing*, analisis sintaksis, analisis semantik, KML parser, peta interaktif.

ABSTRACT

INFORMATION RETRIEVAL SYSTEM BASED ON ONTOLOGY FOR MOUNTAIN CLIMBING TRACK USING NATURAL LANGUAGE QUERY WITH INTERACTIVE MAP PRESENTATION

By

FADHILA TANGGUH ADMOJO

10/310730 / PPA / 03452

Indonesian high public interest in mountain climbing activities led to the increasing need for mountain climbing tracks information. Literature of Indonesian mountain climbing tracks in the form of the book is very difficult to obtain, which make the Internet as a primary tool in getting information about mountaineering.

Finding information about mountain hiking track on the internet raises a new problem among the climbers. The problem is, the search results are numerous and varied and thus require more time to browse and to compare any existing information. Other than that the amount of information that is obtained on the internet even confuse the climbers.

This research aims to provide a solution to the problems faced by climbers, by developing an information retrieval system for mountain climbing track using ontology based approach .

The system is developed by using two ontology. Ontology Bahasa as a linguistic knowledge representation and ontology Mountaineering as a representation of knowledge in mountaineering. The system is designed to perform a search of information with the ability to understand natural language input in the form of words, phrases, clauses and sentences. The process of natural language understanding based on syntactic analysis and semantic analysis using the rules of Indonesian grammar. The system is also equipped with the ability to present information from the search results into an interactive map.

Base on the research that has been conducted, the system is able to understand natural language input both syntactically and semantically. The system is also capable of using thesaurus in the search process and detects the input that does not comply with the rules of Indonesian grammar. Quantitative test results show that the system can process 69% of inputs are taken at random from the respondents.

Keyword: information retrieval, mountaineering, ontologies, semantic web, NLP, parsing, syntactic analysis, semantic analysis, KML parser, interactive maps.

BAB I

PENDAHULUAN

1.1 Latar Belakang

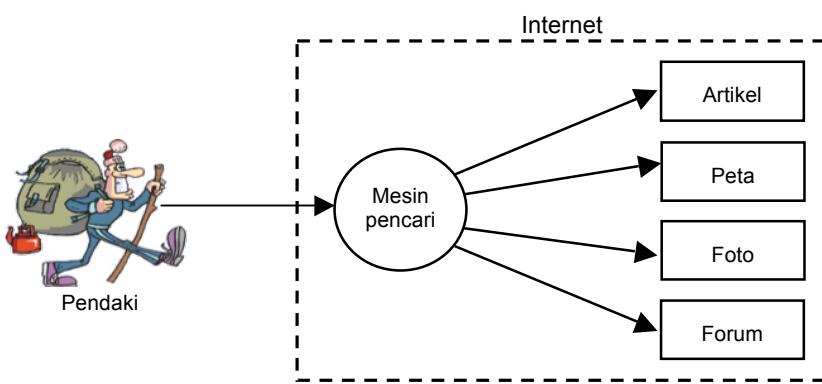
Secara geografis Indonesia terletak diantara pertemuan beberapa rangkaian gunung berapi aktif (*Ring Of Fire*), menjadikan Indonesia negara dengan jumlah gunung berapi terbanyak di dunia. Hargo (2014) memaparkan lebih dari 400 gunung yang telah terdata dan 127 diantaranya merupakan gunung berapi aktif.

Kekayaan alam pegunungan yang dimiliki Indonesia mendorong tumbuh dan berkembangnya minat masyarakat dalam melakukan olahraga mendaki gunung (*mountain climbing*). Sumitro (1997) memaparkan mendaki gunung adalah suatu aktivitas bertualang di alam terbuka menuju ke suatu tempat yang lebih tinggi yaitu puncak gunung. Green (2006); Ibrahim (2007); Parfet (2009); Apriyuandi (2011) memaparkan bahwa mendaki gunung merupakan olahraga keras dan ekstrem yang tidak hanya membutuhkan fisik dan mental yang tangguh saja, karena kunci keberhasilan dari suatu pendakian adalah informasi dan persiapan. Penting untuk mengumpulkan informasi sebanyak mungkin tentang daerah atau lokasi (gunung) sebelum melakukan pendakian agar dapat mempersiapkan jadwal, perlengkapan dan perbekalan yang dibutuhkan, karena kegiatan di alam bebas (mendaki gunung) dapat berjalan dengan lancar tergantung bagaimana pelakunya mempersiapkan diri (Sastha, 2007).

Penyebab utama banyaknya kasus pendaki yang tersesat dan hilang di gunung dikarenakan kurangnya informasi tentang jalur dan medan pendakian. Informasi jalur dan medan pendakian merupakan informasi yang paling vital dalam suatu pendakian. Sulitnya literatur lengkap dalam bentuk buku mengenai informasi pendakian gunung-gunung di Indonesia dan seiring pesatnya perkembangan teknologi informasi, menjadikan internet sebagai media utama dalam mengumpulkan informasi tersebut. Berdasarkan hasil kuesioner dari para pendaki yang dapat dilihat pada Lampiran A, sebagian besar pendaki mengumpulkan informasi di internet dengan memanfaatkan *search engine* (mesin

pencari). Ilustrasi proses pencarian informasi jalur pendakian di internet dengan memanfaatkan mesin pencari dijabarkan pada Gambar 1.1.

Mencari informasi pendakian gunung yang spesifik di internet memunculkan suatu permasalahan baru dikalangan pendaki. Permasalahan utama yang muncul berdasarkan dari hasil kuesioner pendaki yang dijabarkan pada Lampiran A, yaitu: hasil pencarian yang banyak dan beragam sehingga membutuhkan waktu lama untuk menelusuri dan membandingkan setiap informasi, selain itu banyaknya informasi yang tersedia justru membingungkan pendaki.



Gambar 1.1 Ilustrasi pencarian informasi jalur pendakian di internet

Permasalahan serupa dengan permasalahan yang telah dipaparkan di atas pernah dibahas oleh Handaya dan Lestari (2011). Penelitian Handaya dan Lestari (2011) berupaya menyediakan solusi untuk masalah tersebut dengan mengembangkan suatu sistem informasi pemandu pendakian yang dapat menyajikan informasi jalur pendakian gunung dalam bentuk teks dan gambar. Namun, sistem yang telah dikembangkan Handaya dan Lestari (2011) belum mampu menjawab permasalahan yang dihadapi pendaki yaitu untuk mendapatkan informasi yang tepat dalam waktu yang singkat. Sistem Handaya dan Lestari (2011) tidak menyediakan fungsi pencarian sehingga pengguna tidak dapat berinteraksi dengan sistem tentang informasi yang inginkan, pengguna masih harus mencari informasi secara manual ke dalam konten informasi yang disajikan sistem.

Sehubungan dengan kelemahan pada penelitian Handaya dan Lestari (2011), Azhari dan Scholihah (2006) memaparkan bahwa kendala menggunakan

pendekatan model data relasional adalah kurang dinamis untuk mendukung penyimpanan informasi yang lebih bermakna secara semantik, sehingga untuk melakukan pencarian dengan berbagai variasi pengetahuan terhadap data akan menjadi sangat terbatas, dengan menggunakan basis pengetahuan teknologi semantik (ontologi) model data yang digunakan untuk kasus-kasus pencarian informasi dapat dideskripsikan secara lebih semantik dan dapat digunakan berdasarkan persepsi pengguna. Rahutomo (2009) memaparkan pencarian dengan basis semantik berarti pencarian informasi berdasarkan makna yang terkait dengan kata kunci penelusuran, pencarian semantik adalah mencari konten yang sesuai dengan konteks yang diinginkan pengguna.

Menurut Pollock (2009) pencarian semantik terbagi ke dalam dua jenis. Jenis pertama adalah pencarian semantik yang memberikan hasil berupa *link* navigasi, jenis kedua adalah pencarian dengan memasukkan frasa atau kalimat (bahasa alami) yang mewakili keinginan pengguna untuk mendapatkan informasi. Berdasarkan pemaparan Kaufmann dan Bernstein (2007); Damljanovic dan Bontcheva (2009) proses pencarian dengan penggunaan *query* bahasa alami menjadi yang paling *user-friendly* bagi pengguna dan lebih mempercepat pengguna dalam melakukan pencarian informasi.

Penelitian tentang pencarian semantik dengan *input* berupa *query* bahasa alami pernah dilakukan oleh Nurkhamid (2009), Bendi (2010), Andri (2011) dan Suryawan (2013). Meskipun keempat penelitian tersebut menggunakan teknologi semantik dengan pendekatan yang berbeda-beda namun dari hasil penelitian dapat dikatakan penggunaan teknologi semantik mampu memberikan kemudahan pada pengguna sistem dalam melakukan pencarian informasi dengan hasil pencarian yang cukup relevan berdasarkan keinginan pengguna. Mengacu pada keempat penelitian tersebut, penggunaan teknologi semantik dalam kasus pencarian cukup tepat untuk mengatasi kelemahan seperti pada sistem yang dikembangkan Handaya dan Lestari (2011), dengan pemanfaatan teknologi semantik pendaki dapat melakukan pencarian informasi dengan mudah dan mendapatkan informasi yang relevan.

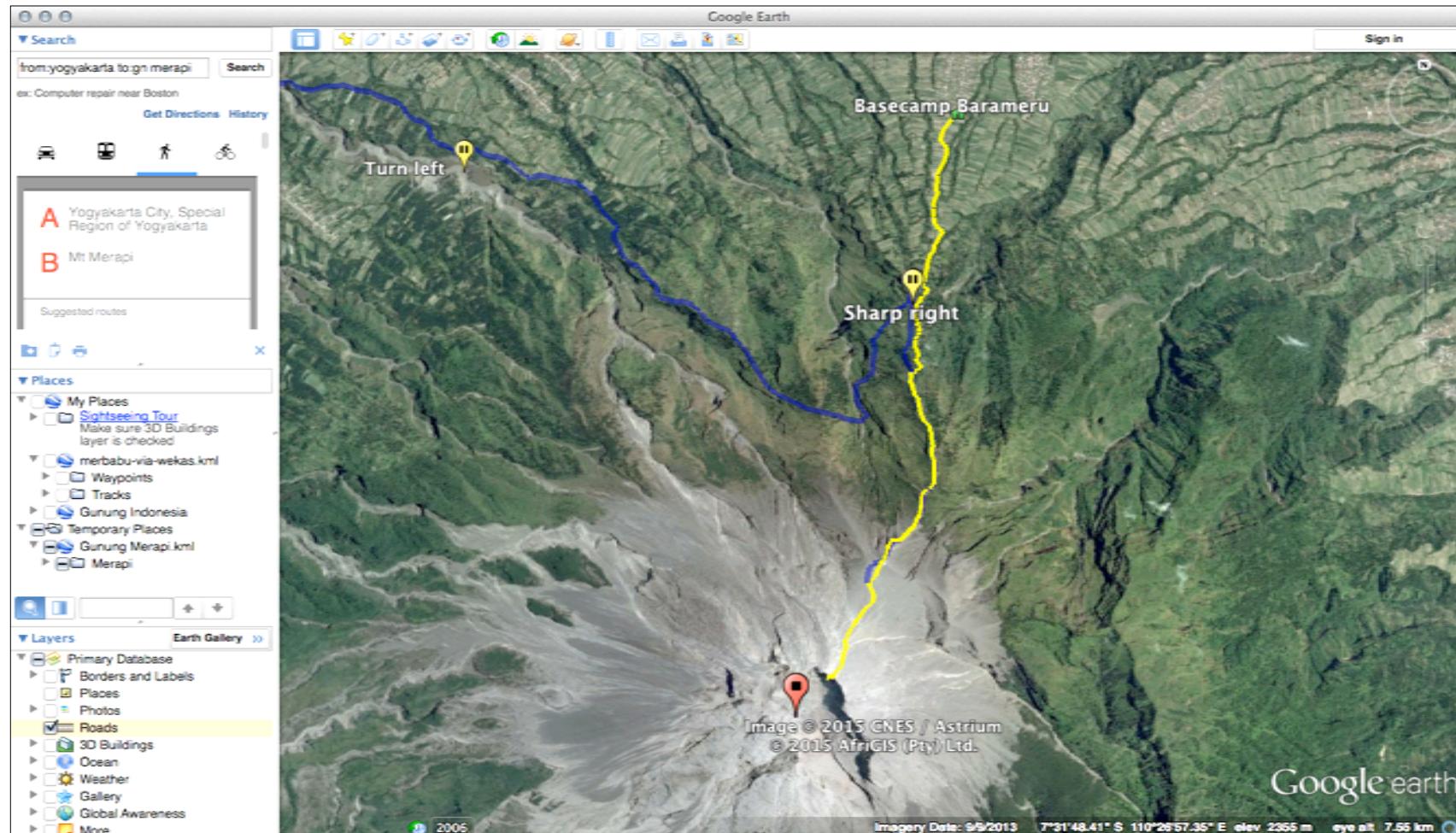
Kebutuhan lain dari para pendaki yaitu penyajian informasi jalur pendakian dalam bentuk peta. Penyajian informasi dalam bentuk peta sangat berguna bagi pendaki untuk dapat melihat informasi visualisasi entitas geografis dari jalur pendakian yang akan dilalui, sehingga memungkinkan pendaki untuk dapat menganalisa dan memperhitungkan banyak hal, misalnya: lokasi mendirikan tenda, bermalam dan lokasi sumber air.

Mengacu kepada penelitian Handaya dan Lestari (2011). Meskipun sistem pemandu pendakian yang telah dikembangkan mampu menyajikan informasi peta, akan tetapi peta yang disajikan masih berupa cetak berbentuk *file* citra kompresi (*JPEG*) bukan peta yang interaktif. Cuplikan contoh peta jalur pendakian gunung dalam bentuk *file* citra kompresi dapat dilihat pada Gambar 1.2.



Gambar 1.2 Informasi peta jalur pendakian berbentuk *file* citra kompresi (www.merbabu.com)

Informasi keruangan seperti pada Gambar 1.2 bukanlah informasi yang tepat untuk dijadikan acuan dalam pendakian. selain tidak akurat informasi pada peta tersebut membingungkan bagi pendaki. Alternatif yang dapat digunakan sebagai alat bantu dalam mencari informasi peta tentang jalur pendakian yaitu dengan memanfaatkan aplikasi berbasis peta seperti *Google Maps/Earth*.



Gambar 1.3 Perbandingan informasi jalur pendakian gunung merapi dari *Google Maps* dengan data hasil observasi (diakses tanggal 27 Februari 2015)

Google Maps/Earth merupakan bentuk layanan dari *Google* yang menawarkan teknologi pemetaan terkini yang dapat digunakan secara bebas dengan *platform opensource* Irwansyah dkk., (2011). Layanan *Google Maps* sangat mampu menyajikan peta interaktif yang akurat, akan tetapi aplikasi tersebut tidak menyediakan informasi yang khusus untuk pendakian gunung. Cuplikan perbandingan hasil pencarian informasi jalur pendakian menggunakan layanan *Google Maps* dengan informasi hasil observasi dapat dilihat pada Gambar 1.3. Garis berwarna biru pada Gambar 1.3 menunjukkan informasi yang dihasilkan oleh *Google Maps* dan garis berwarna kuning adalah informasi hasil observasi menggunakan alat *Global Positioning System (GPS)*.

Berdasarkan apa yang telah dipaparkan di atas, penelitian ini sejalan dengan penelitian Handaya dan Lestari (2011) yaitu berusaha untuk memberikan solusi untuk permasalahan pencarian informasi yang dihadapi pendaki. Namun dengan menggunakan pendekatan yang berbeda. Penelitian ini berusaha mengembangkan sebuah sistem pencarian informasi untuk jalur pendakian gunung menggunakan basis pengetahuan semantik. Kemampuan pencarian yang dikembangkan yaitu mampu untuk memahami *input query* bahasa alami dan mampu menggunakan pemahaman untuk mencari dan menghasilkan informasi dengan basis penyajian berupa peta interaktif. Fitur tambahan yang yang dikembangkan pada penelitian ini yaitu berupa fitur *spelling checker*.

1.2 Rumusan Masalah

Berdasarkan penjelasan pada latar belakang, rumusan-rumusan masalah yang dibahas pada penelitian ini, yaitu:

1. Bagaimana memproses dan memahami *input* bahasa alami menggunakan aturan tata bahasa Indonesia.
2. Bagaimana menggunakan hasil pemahaman *input* bahasa alami untuk proses pencarian informasi jalur pendakian gunung.
3. Bagaimana menyajikan informasi hasil pencarian ke dalam bentuk peta yang interaktif.

1.3 Batasan Masalah

Batasan masalah pada penelitian ini sebagai berikut:

1. Data jalur pendakian yang digunakan sebagai sampel uji coba yaitu data jalur pendakian gunung Merapi dan gunung Merbabu.
2. Informasi yang disajikan sistem yaitu objek pada jalur pendakian dan gunung meliputi *basecamp*, puncak, kawah, pos, sumber air dan *campground*. Atribut dari objek meliputi deskripsi, lokasi, ketinggian dan cuaca. Peta dasar menggunakan layanan *Google Maps API*. Data cuaca diambil melalui *web service* milik Badan Meteorologi, Klimatologi dan Geofisika (BMKG) dengan alamat <http://data.bmkg.go.id/>.
3. Sistem tidak menyajikan informasi transportasi kendaraan, kebutuhan peralatan perbekalan, waktu dan biaya.
4. Sistem hanya memproses *input* untuk informasi faktual yang tidak bersifat konfirmatif.
5. Sistem memproses *input* yang mengandung nomina, verba, preposisi, numeralia, artikula (yang), pronomina, partikel (kah) dan adverbia (saja).
6. Aturan tata bahasa Indonesia yang direpresentasikan ke dalam aturan gramatikal mengacu pada tata bahasa Indonesia yang dikemukakan oleh Alwi dkk., (2003).

1.4 Tujuan Penelitian

Tujuan dari penelitian ini adalah memberikan solusi untuk permasalahan pencarian informasi pendakian gunung, yaitu dengan mengembangkan sebuah sistem yang mampu menggunakan pengetahuan berbasis teknologi semantik untuk memahami *query* bahasa alami Indonesia dan mampu menghasilkan informasi yang relevan dengan penyajian informasi berupa peta interaktif.

1.5 Manfaat Penelitian

Manfaat dari penelitian ini, antara lain :

1. Untuk penggiat olahraga *mountain climbing*, yaitu :
 - Membantu mendapatkan informasi jalur pendakian dengan mudah, pendaki cukup menggunakan *query* bahasa alami untuk mendapatkan

informasi yang diinginkan, dan dengan penyajian peta interaktif pendaki dimudahkan untuk mengetahui lokasi suatu objek pendakian.

- Membantu mempercepat proses pengumpulan informasi, karena pendaki cukup menggunakan satu aplikasi *web* tanpa harus mengunjungi banyak *website* untuk mendapatkan informasi jalur pendakian.
 - Mengurangi kompleksitas ketika melakukan pencarian informasi karena dengan penggunaan *thesaurus* pada proses pencarian informasi pendaki tidak perlu melakukan pencarian berulang-ulang menggunakan kata kunci yang berbeda namun memiliki kaitan semantik yang sama.
2. Manfaat penelitian ini untuk penelitian sejenis dibidang teknologi semantik *web*, yaitu ontologi Bahasa yang dikembangkan pada penelitian ini dapat digunakan kembali untuk kasus-kasus pencarian yang melibatkan domain pencarian yang berbeda, misalnya: domain informasi wisata dan informasi rumah sakit.

1.6 Metodologi Penelitian

Metodologi yang digunakan dalam penelitian ini, adalah sebagai berikut:

1. Studi Literatur

Mengumpulkan dan mempelajari literatur yang membahas tentang metode pengembangan ontologi, implementasi pengetahuan ke dalam ontologi, metode pemrosesan bahasa alami, tata bahasa Indonesia untuk analisis sintaksis dan semantik, serta mengkaji metode-metode penggunaan data spasial pada ontologi.

2. Pengumpulan data

Mengumpulkan data-data yang akan digunakan, antara lain:

- Data-data jalur pendakian di gunung Merapi dan Merbabu yang dikumpulkan dengan cara observasi.
- Data-data yang berhubungan dengan bidang linguistik berupa kosakata, aturan tata bahasa dan bentuk kalimat yang biasa digunakan pendaki ketika mengumpulkan informasi. Data diperoleh dengan penyebaran kuesioner kepada para pendaki.

3. Analisis dan Perancangan

Analisis dan perancangan dilakukan secara bertahap, yaitu: merancang model dari sistem dengan menggunakan *Unified Modelling Language (UML)*, merancang basis pengetahuan (*knowledge-base*) yang digunakan sistem, dan merancang antarmuka sistem.

4. Implementasi Sistem

Pada tahap ini hasil dari tahap analisis dan perancangan akan diimplementasi.

5. Pengujian dan Pembahasan

Pada tahap ini sistem diujikan menggunakan sejumlah *input* yang dikumpulkan dari responden. Pengujian bertujuan untuk melihat apakah sistem mampu memahami *input* menggunakan kaidah tata bahasa Indonesia, apakah sistem mampu mendeteksi *input* yang tidak sesuai dengan kaidah tata bahasa Indonesia, apakah sistem mampu menyajikan informasi beserta dengan penyajian peta interaktif yang sesuai dengan *input*. Hasil pengujian kemudian akan dievaluasi.

6. Kesimpulan dan Saran

Pada tahap ini, hasil penelitian dan pengujian akan dirangkum dan dijadikan suatu kesimpulan. Kekurangan yang pada sistem akan dijadikan sebagai saran untuk penelitian selanjutnya.

1.7 Sistematika Penulisan

Penelitian ini terbagi ke dalam tujuh bab, yaitu sebagai berikut:

1. Bab I Pendahuluan

Membahas mengenai latar belakang, rumusan masalah, tujuan penelitian, metodologi penelitian, dan sistematika penulisan.

2. Bab II Tinjauan Pustaka

Membahas tentang penelitian-penelitian yang berhubungan dengan penelitian ini dan perbedaan penelitian ini dengan penelitian-penelitian yang telah dilakukan sebelumnya.

3. Bab III Landasan Teori

Berisi teori-teori yang digunakan sebagai dasar dalam penelitian ini, yang terdiri dari teori tata bahasa Indonesia, teori tentang pemrosesan bahasa alami (*natural language processing*), serta teori tentang pembentukan pengetahuan dan semantik *web*.

4. Bab IV Analisis dan Perancangan Sistem

Membahas mengenai analisa dan arsitektur sistem, perancangan sistem, perancangan ontologi dan perancangan antarmuka.

5. Bab V Implementasi

Membahas mengenai implementasi pengetahuan ke dalam ontologi, implementasi sistem dan implementasi antarmuka.

6. Bab VI Hasil dan Pembahasan

Menjelaskan hasil-hasil dari penelitian yang telah dilakukan, melakukan pengujian dan melakukan pembahasan.

7. Bab VII Kesimpulan dan Saran

Menjelaskan tentang kesimpulan dari penelitian yang telah dilakukan dan memuat saran untuk pengembangan pada penelitian selanjutnya.

BAB II

TINJAUAN PUSTAKA

Penelitian yang membahas masalah pencarian informasi pendakian gunung sebelumnya pernah dilakukan oleh Handaya dan Lestari (2011). Handaya dan Lestari (2011) mengembangkan sebuah sistem pemandu pendakian gunung berbasis *web* memanfaatkan basis data relasional untuk menyimpan data-data jalur pendakian. Kelemahan dari sistem yang dikembangkan Handaya dan Lestari (2011) yaitu tidak memiliki kemampuan untuk melakukan pencarian informasi sebagai bentuk komunikasi yang interaktif antara pengguna dengan sistem. Sistem menyajikan informasi secara searah dengan melakukan *query* ke dalam basis data relasional sehingga pengguna masih harus menelusuri dan membaca satu persatu konten informasi yang disajikan oleh sistem. Kelemahan lain dari penelitian Handaya dan Lestari (2011) yaitu pada penyajian informasi peta yang menggunakan peta cetak berupa *file* gambar kompresi (JPEG) sehingga tidak memungkinkan bagi pendaki untuk dapat melihat secara akurat informasi tentang lokasi suatu objek pendakian.

Azhari dan Sholichah (2006) menjelaskan bahwa kelemahan dari penggunaan basis data relasional seperti pada penelitian Handaya dan Lestari (2011) untuk proses pencarian informasi dapat diatasi dengan menggunakan pemodelan ontologi (basis pengetahuan). Fokus penelitian Azhari dan Scholichah (2006) membentuk informasi jadwal penerbangan pesawat menjadi suatu basis pengetahuan (ontologi), sehingga pencarian informasi dapat dibentuk berdasarkan persepsi pengguna dan informasi dapat dideskripsikan secara lebih semantik. Berdasarkan hasil pengujian model ontologi yang telah dibentuk dapat digunakan untuk menyajikan informasi berdasarkan apa yang maksudkan pengguna menggunakan bentuk *query* pencarian yang berbeda. Penelitian serupa Azhari dan Scholichah (2006) yang membahas tentang pemodelan ontologi juga dilakukan oleh Amborowati (2007) untuk domain informasi jadwal kereta api dan Wicaksono dkk., (2010) untuk jalur bus Trans Jogja. Penelitian yang dilakukan oleh Azhari dan

Sholichah (2006) juga Amborowati (2007) dan Wicaksono dkk., (2010) meskipun fokus pada pemodelan ontologi namun pendekatan yang digunakan dalam mengembangkan ontologi dapat dijadikan sebagai acuan dalam merepresentasikan suatu domain pengetahuan ke dalam ontologi.

Fazingga dan Lukasiewicz (2010) menyatakan bahwa tujuan utama dari *semantic search* adalah peningkatan kemampuan sistem dalam mengolah *query* bahasa alami, seperti penelitian yang dilakukan oleh Nurkhamid (2009). Nurkhamid (2009) mengembangkan sebuah sistem pencarian semantik untuk domain perpustakaan. Pengetahuan perpustakaan yang digunakan oleh Nurkhamid (2009) didefinisikan ke dalam ontologi termasuk kosakata (*vocabulary*), padanan kata (*thesaurus*), klasifikasi kata (taksonomi) dan daftar istilah (*glossaries*) sehingga hasil pencarian dapat dikembangkan berdasarkan kesamaan makna dengan kata kunci. Penelitian Nurhamid (2009) mampu memproses *input* bahasa alami berupa kata kunci dan melakukan pencarian dengan *thesaurus* kata. Namun demikian, kelemahan dari penelitian yang dilakukan oleh Nurkhamid (2009) yaitu tidak dapat memproses *input* berupa kalimat karena pengecekan hanya didasarkan pada pencocokan kata kunci.

Andri (2011) mengembangkan kembali penelitian yang telah dilakukan oleh Nurkhamid (2009). Andri (2011) memodifikasi ontologi yang telah dibentuk oleh Nurkhamid (2009) dan menambahkan operasi *stemming* untuk mengembalikan suatu kata ke akar katanya. Andri (2011) menggunakan operasi *stemming* yang telah dikembangkan oleh Tala (2003). Penelitian Andri (2011) mampu memproses *input* bahasa alami berupa kalimat perintah sederhana dan penggunaan operasi *stemming* dapat meningkatkan akurasi hasil pencarian, akan tetapi penelitian Andri (2011) belum mampu untuk memproses kalimat yang kompleks karena pemeriksaan yang dilakukan hanya didasarkan pada pola-pola kata yang sudah ditentukan (*surface structure*), tanpa melakukan analisis struktur dalam (*deep structure*) menggunakan aturan tata bahasa Indonesia. Kelemahan lain pada penelitian Andri (2011) yaitu proses pencarian yang didasarkan pada ekstraksi kata kunci yang terdapat pada *input*, bukan didasarkan pada pemahaman terhadap keseluruhan *input*.

Pendalaman dari pencarian semantik yaitu berupa sistem *question answering* (QA) yang memanfaatkan ontologi untuk memproses *query* bahasa alami hingga menghasilkan jawaban yang spesifik, seperti penelitian yang dilakukan oleh Bendi (2010). Sistem yang dikembangkan oleh Bendi (2010) mampu memproses kalimat pertanyaan dengan melakukan analisis struktur luar/struktur lahir (*surface structure*). Penelitian Bendi (2010) melakukan pengecekan kalimat menggunakan pola-pola yang sudah ditentukan dan membuang kata yang tidak dibutuhkan dengan menggunakan daftar *stopword* untuk menghasilkan kata kunci (*keyword*), kemudian mentranslasikan kata kunci ke dalam bentuk *query* SPARQL. Jawaban diperoleh dengan cara mengeksekusi *query* SPARQL. Kelemahan dari sistem yang dikembangkan Bendi (2010) yaitu tidak dapat melakukan analisis struktur dalam/struktur batin (*deep structure*). Sistem hanya memproses kalimat yang sederhana berdasarkan pola-pola kalimat yang sudah ditentukan dengan proses pencarian informasi didasarkan pada kata kunci, Kelemahan lain yaitu tidak dapat mendeteksi kalimat tanya menggunakan aturan tata bahasa Indonesia.

Djajasudarma (1999) memaparkan bahwa suatu kata dapat bermakna lebih dari satu, dapat mengacu pada benda yang berbeda sesuai dengan lingkungan pemakainya. Oleh karena itu proses pencarian yang didasarkan pada ekstraksi kata kunci saja seperti pada penelitian Nurkhamid (2009), Bendi (2010) dan Andri (2011) belum cukup untuk menginterpretasikan keinginan pengguna apabila pengguna memberikan *input* kepada sistem berupa rentetan kata (kalimat) yang memiliki kesatuan makna yang kompleks. Peningkatan kualitas pemrosesan *query* bahasa alami untuk memproses kalimat yang kompleks dengan menggunakan analisis struktur dalam (*deep structure*) dan mampu memahami kalimat tanya berdasarkan satuan makna pada kalimat bukan hanya didasarkan pada ekstraksi kata kunci yaitu seperti penelitian yang dilakukan oleh Suryawan (2013).

Suryawan (2013) melengkapi kelemahan-kelemahan yang terdapat pada penelitian sebelumnya dengan memisahkan pengetahuan yang digunakan untuk memproses bahasa alami dan pengetahuan yang digunakan untuk menghasilkan jawaban. Dengan memisahkan pengetahuan berdasarkan domain permasalahan

maka masing-masing pengetahuan dapat digunakan untuk menangani pemrosesan yang spesifik. Penelitian Suryawan (2013) mampu memberikan hasil pencarian yang relevan berdasarkan makna kalimat dan mampu mendeteksi kalimat menggunakan aturan tata bahasa Indonesia. Namun demikian kelemahan dari penelitian yang dikembangkan Suryawan (2013) yaitu hanya memproses *input* berupa kalimat tanya saja.

Berdasarkan penelitian untuk kasus-kasus pencarian yang telah dipaparkan di atas, penelitian ini berusaha untuk mengembangkan suatu sistem pencarian menggunakan teknologi semantik untuk menyelesaikan permasalahan yang telah dijabarkan pada Subbab 1.1. Hasil penelitian diharapkan dapat menggabungkan kelebihan-kelebihan dari penelitian yang dilakukan Nurkhamid (2009), Bendi (2010), Andri (2011) dan Suryawan (2013). Fitur tambahan yang dikembangkan pada penelitian ini berdasarkan saran-saran dari penelitian Bendi (2010) dan Andri (2011) yaitu fitur pengecekan kesalahan pengetikan ejaan (*spelling checker*).

Penelitian lain yang berhubungan dengan pembahasan penelitian ini yaitu penelitian yang memanfaakan *platform Google Maps* untuk penyajian informasi, seperti penelitian yang dilakukan Yunita (2011). Fokus penelitian Yunita (2011) yaitu pada penggunaan *Semantic Web Rule Language* (SWRL) untuk menentukan paket perjalanan wisata dengan membentuk hubungan objek-objek wisata ke dalam bentuk graf yang direpresentasikan ke dalam ontologi. Meskipun sama-sama menggunakan *platform Google Maps* dalam menyajikan informasi, namun perbedaan penelitian Yunita (2011) dengan penelitian ini yaitu pada pendekatan yang digunakan untuk mengintegrasikan data spasial ke dalam ontologi. Yunita (2011) menggunakan *data property latitude* dan *longitude* didalam ontologi untuk mendefinisikan lokasi suatu objek wisata yang akan ditampilkan pada peta. Sedangkan penelitian ini menggunakan KML untuk menyimpan data-data spasial yang diintegrasikan dengan ontologi, sehingga data-data spasial dapat digunakan untuk proses *parsing* berdasarkan hasil pencarian dan dapat digunakan secara lebih interaktif. Rangkuman perbandingan penelitian ini dengan penelitian sebelumnya dapat dilihat pada Tabel 2.1.

Tabel 2.1 Rangkuman perbandingan penelitian

No.	Peneliti	Fokus Penelitian	Representasi Data	Pendekatan yang digunakan	Hasil Penelitian
1	Handaya dan Lestari (2011)	Pengembangan Sistem informasi untuk domain informasi pendakian gunung	Basis data relasional	Penyajian informasi dengan query database SQL	Sistem informasi yang dapat menyajikan informasi jalur pendakian gunung dalam bentuk teks dan gambar
2	Azhari dan Sholichah (2006)	Pengembangan model ontologi untuk jadwal penerbangan	Ontologi	Representasi pengetahuan	Ontologi jadwal penerbangan
3	Amborowati (2007)	Pengembangan model ontologi untuk jadwal kereta api	Ontologi	Representasi pengetahuan	Ontologi jadwal kereta api
4	Nurkhamid (2009)	Pengembangan Sistem pencarian untuk domain perpustakaan	Ontologi	<ul style="list-style-type: none"> • Representasi pengetahuan • Pencocokan kata kunci dengan <i>thesaurus</i> • Ekstraksi informasi semantik dengan query SPARQL 	<ul style="list-style-type: none"> • Ontologi perpustakaan • Sistem yang dapat menyajikan informasi perpustakaan, fasilitas pencarian menggunakan kata kunci dan mampu menggunakan <i>thesaurus</i> kata
5	Bendi (2010)	Pengembangan Sistem QA untuk domain informasi perfilman	Ontologi	<ul style="list-style-type: none"> • Representasi pengetahuan • Analisa berdasarkan kata kunci dan pola kalimat • <i>Tokenizer</i> dengan ekstraksi kata kunci menggunakan <i>stopword</i> • Ekstraksi informasi semantik dengan query SPARQL 	<ul style="list-style-type: none"> • Ontologi Perfilman • Sistem yang dapat memberikan informasi perfilman, pencarian informasi menggunakan kalimat tanya tunggal
7	Andri (2011)	Pengembangan Sistem pencarian untuk domain perpustakaan	Ontologi	<ul style="list-style-type: none"> • Representasi pengetahuan • Analisa sintaksis berdasarkan kata kunci dan pola kalimat • <i>Tokenizer</i> dengan ekstraksi kata kunci menggunakan operasi <i>stemming</i> dan <i>stopword</i> 	<ul style="list-style-type: none"> • Ontologi Perpustakaan • Sistem yang dapat memberikan informasi perpustakaan, pencarian informasi menggunakan kalimat perintah tunggal

Tabel 2.1 – (Lanjutan)

No.	Peneliti	Fokus Penelitian	Representasi Data	Pendekatan yang digunakan	Hasil Penelitian
7	Yunita (2011)	Sistem perencanaan perjalanan wisata	Ontologi	<ul style="list-style-type: none"> • Representasi pengetahuan • Penggunaan SWRL untuk membentuk graf yang menghubungkan antar lokasi objek-objek wisata 	Sistem yang membantu merencanakan perjalanan pariwisata berdasarkan kriteria-kriteria tertentu, dengan penyajian informasi ke dalam bentuk peta
8	Suryawan (2013)	Pengembangan Sistem QA untuk domain perpustakaan	Ontologi	<ul style="list-style-type: none"> • Representasi pengetahuan • <i>Preprocessing</i> dan pembentukan urutan kata • <i>Parsing</i> menggunakan aturan tata bahasa Indonesia • Proses translasi pertanyaan menjadi <i>query SPARQL</i> dengan analisis struktur dalam (<i>deep structure</i>) • Ekstraksi informasi semantik menggunakan <i>query SPARQL</i> dengan <i>rule to rule hypothesis</i> 	<ul style="list-style-type: none"> • Ontolingua • Ontopustaka • Sistem yang dapat menyajikan informasi perpustakaan, melakukan pencarian dengan <i>input</i> kalimat tanya, mampu memahami kalimat dengan kaidah tata bahasa Indonesia dan mendeteksi kalimat yang tidak sesuai dengan kaidah tata bahasa Indonesia, menggunakan <i>thesaurus</i> kata dalam pencarian
9	Admojo (2015)	Pengembangan Sistem pencarian untuk domain informasi jalur pendakian gunung	Ontologi dan KML	<ul style="list-style-type: none"> • Representasi pengetahuan • <i>Preprocessing</i> dan pembentukan urutan kata • <i>Parsing</i> menggunakan aturan tata bahasa Indonesia dengan analisis struktur luar (<i>surface structure</i>) • Proses pembentukan <i>query SPARQL</i> dengan analisis struktur dalam (<i>deep structure</i>) • Ekstraksi informasi semantik menggunakan <i>query SPARQL</i> • Pemrosesan data spasial menggunakan KML Parser 	<ul style="list-style-type: none"> • Ontologi bahasa • Ontologi <i>Mountaineering</i> • Sistem yang dapat menyajikan informasi jalur pendakian gunung dengan penyajian informasi berupa peta interaktif, sistem yang mampu melakukan pencarian dengan <i>input</i> berupa kata, frasa, klausa atau kalimat, mampu memahami kalimat dengan kaidah tata bahasa indonesia dan mendeteksi kalimat yang tidak sesuai dengan kaidah tata bahasa indonesia dan menggunakan <i>thesaurus</i> kata dalam pencarian

BAB III

LANDASAN TEORI

3.1 Bahasa

Menurut Dardjowidjojo (2010), bahasa adalah sistem simbol lisan yang arbitrer yang dipakai oleh anggota suatu masyarakat bahasa untuk berkomunikasi dan berinteraksi antar sesamanya dengan berlandaskan pada budaya yang mereka miliki bersama. Menurut Chaer (2006), bahasa adalah suatu lambang berupa bunyi, bersifat arbitrer, digunakan oleh suatu masyarakat tutur untuk bekerja sama, berkomunikasi, dan mengidentifikasi diri.

Sebagai suatu sistem, bahasa memiliki aturan, kaidah, atau pola-pola tertentu, baik dalam tata bunyi, tata bentuk kata, maupun tata kalimat. Bahasa disusun oleh tiga komponen, yaitu: sintaksis, fonologi, dan semantik. Komponen sintaksis adalah komponen yang menangani ihwal yang berkaitan dengan kata, frasa, dan kalimat. Komponen fonologi adalah komponen yang menangani ihwal yang berkaitan dengan bunyi. Komponen semantik adalah komponen yang membahas ihwal makna (Dardjowidjojo, 2010).

3.1.1 Kata

Kata merupakan perwujudan bahasa sehingga bahasa tidak akan ada jika kata tidak ada. Setiap kata mengandung konsep makna dan mempunyai peran dalam pelaksanaan bahasa (Chaer, 2006). Alwi dkk., (2003) mengelompokkan kata ke dalam empat kategori sintaktis utama yaitu: verba, nomina, adjektiva, dan adverbia.

Alwi dkk., (2003) juga memperkenalkan kelompok kata lain yang dinamakan kata tugas yang terdiri dari beberapa kelompok yang lebih kecil yaitu: preposisi, konjungtor, dan partikel.

Verba (kata kerja)

Secara umum, verba berfungsi sebagai predikat atau sebagai inti predikat di dalam suatu kalimat (Alwi dkk., 2003). Dilihat dari perilaku semantiknya, verba memiliki makna inheren yang terkandung di dalam verba itu sendiri. Verba dapat

mengandung makna inheren perbuatan (aksi), proses, atau keadaan yang bukan sifat atau kuantitas (Alwi dkk., 2003).

Adjektiva (kata sifat)

Adjektiva adalah kata yang memberikan keterangan yang lebih khusus tentang sesuatu yang dinyatakan oleh nomina dalam kalimat (Alwi dkk., 2003). Menurut Alwi dkk., (2003), adjektiva dari segi perilaku semantiknya terbagi menjadi dua tipe pokok, yaitu:

1. Adjektiva bertaraf yang mengungkapkan suatu kualitas, adjektiva bertaraf dibagi atas (1) adjektiva pemerl sifat, (2) adjektiva ukuran, (3) adjektiva warna, (4) adjektiva waktu, (5) adjektiva jarak, (6) adjektiva sikap batin, dan (7) adjektiva cerapan.
2. Adjektiva tak bertaraf yang mengungkapkan keanggotaan dalam suatu golongan.

Sedangkan dari segi sintaksisnya, adjektiva dapat berperilaku sebagai fungsi atribut, fungsi predikat, dan fungsi adverbial atau keterangan.

Adverbia (kata keterangan)

Adverbia adalah kata-kata yang digunakan untuk memberi penjelasan pada kalimat atau bagian kalimat lain, yang sifatnya tidak menerangkan keadaan atau sifat (Chaer, 2006). Adverbia perlu dibedakan antara adverbia dalam tataran frasa dengan adverbia dalam tataran klausa. Dalam tataran frasa, adverbia adalah kata yang menjelaskan verba, adjektiva, atau adverbia lain. Dalam tataran klausa, adverbia mewatasi atau menjelaskan fungsi-fungsi sintaktis (Alwi dkk., 2003).

Nomina (kata benda)

Dari segi bentuk morfologisnya, nomina dapat dibagi ke dalam nomina dasar dan nomina turunan. Nomina dasar adalah nomina yang terdiri atas satu morfem. Nomina turunan adalah nomina yang diturunkan melalui proses afiksasi, perulangan, atau pemajemukan. Dari sisi semantiknya, kata benda mengacu pada manusia, binatang, benda, dan konsep atau pengertian. Secara sintaktis, nomina mempunyai ciri-ciri sebagai berikut (Alwi dkk., 2003):

1. Dalam kalimat yang predikatnya berupa verba, nomina cenderung

- menduduki fungsi subjek, objek, atau pelengkap.
2. Nomina tidak dapat diingkarkan dengan kata tidak.
 3. Nomina umumnya dapat diikuti dengan adjektiva, baik secara langsung maupun dengan dihubungkan oleh kata yang.

Pronomina

Pronomina adalah kata yang dipakai untuk mengacu kepada nomina lain. Bahasa Indonesia mengenal tiga macam pronomina, yaitu (Alwi dkk., 2003):

1. Pronomina persona yaitu pronomina yang digunakan untuk mengacu pada orang. Pronomina persona dapat mengacu pada diri sendiri (pronomina persona pertama), mengacu pada orang yang diajak bicara (pronomina persona kedua), atau mengacu pada orang yang dibicarakan (pronomina persona ketiga).
2. Pronomina penunjuk, Bahasa Indonesia memiliki tiga macam pronomina penunjuk, yaitu: pronomina penunjuk umum, pronomina penunjuk tempat, dan pronomina penunjuk ihwal.
3. Pronomina penanya yaitu pronomina yang dipakai sebagai pemarkah pertanyaan. Dari segi maknanya, yang ditanyakan dapat mengenai orang, barang, atau pilihan. Ditinjau dari bentuknya, kata penanya didasari oleh dua bentuk yaitu: apa dan mana. Unsur dasar tersebut dapat dikembangkan menjadi bentuk lain seperti apa, siapa, mengapa, kenapa, kapan, (ke) berapa, dimana, kemana, dari mana, bagaimana, dan bilamana.

Numeralia (kata bilangan)

Numeralia adalah kata yang digunakan untuk menghitung banyaknya maupun (orang, binatang, atau barang) dan konsep. Bahasa Indoensia mengenal dua macam numeralia, yaitu: nomeralia pokok (numeralia kardinal) dan numeralia tingkat (numeralia ordinal) (Alwi dkk., 2003).

Preposisi (kata depan)

Preposisi dalam bahasa Indonesia berfungsi sebagai penanda hubungan tempat, peruntukan, sebab, kesertaan atau cara, pelaku, waktu, ihwal (peristiwa), dan milik. Dilihat dari perilaku sintaksisnya, preposisi berada di depan nomina,

adjektiva, atau adverbia. Dilihat dari segi bentuknya, preposisi dapat dibagi menjadi preposisi tunggal dan preposisi majemuk (Alwi dkk., 2003).

Konjungtor (kata sambung)

Konjungtor adalah kata tugas yang menghubungkan dua satuan bahasa yang sederajat: kata dengan kata, frasa dengan frasa, atau klausa dengan klausa. Dilihat dari perilaku sintaksisnya dalam kalimat, konjungktor dapat dibagi menjadi empat kelompok, yaitu: konjungtor koordinatif, konjungtor korelatif, konjungtor subordinatif, dan konjungtor antar kalimat (Alwi dkk., 2003).

Interjeksi (kata seru)

Interjeksi adalah kata tugas yang digunakan untuk mengungkapkan rasa hati pembicara. Interjeksi digunakan untuk memperkuat rasa hati pembicara misalnya rasa kagum, sedih, heran dan jijik. Interjeksi umumnya ditempatkan di awal kalimat dan pada penulisannya diikuti oleh tanda koma. Secara struktural, interjeksi tidak bertalian dengan unsur kalimat yang lain (Alwi dkk., 2003).

Artikula

Artikula adalah kata tugas yang membatasi makna nomina. Bahasa Indonesia membagi artikula ke dalam tiga kelompok, yaitu: artikula yang bersifat gelar, artikula yang mengacu ke makna kelompok, dan artikula yang menominalkan (Alwi dkk., 2003).

Partikel penegas

Partikel penegas adalah morfem-morfem yang digunakan untuk menegaskan (Chaer, 2006). Partikel penegas meliputi kata yang tidak tertakluk pada perubahan bentuk dan hanya berfungsi untuk menampilkan unsur yang diiringinya. Bahasa Indonesia memiliki empat partikel penegas, yaitu: *-lah*, *-kah*, *-tah* dan *-pun* (Alwi dkk., 2003).

3.1.2 Frasa

Frasa adalah gabungan dua buah kata atau lebih yang merupakan satu kesatuan. Tujuan dari penggabungan dua kata atau lebih menjadi satu kesatuan adalah untuk menampung konsep makna yang lebih khas atau lebih tertentu, yang

tidak dapat diwujudkan dengan sebuah kata saja (Chaer, 2006). Secara sintaksis, frasa dapat dikelompokkan ke dalam frasa verbal, frasa nominal, frasa pronominal, frasa numeralia, dan frasa preposisional.

Frasa Verbal

Frasa verbal adalah satuan bahasa bukan klausa yang terbentuk dari dua kata atau lebih dengan verba sebagai intinya (Alwi dkk., 2003). Satuan bahasa yang dapat mendampingi verba dalam frasa verbal dapat berupa frasa nominal, klausa pemerlengkapan, dan atau frasa preposisional (Lapolika, 1990).

Frasa Nominal

Frasa nominal dapat dibentuk dari nomina yang didahului oleh frasa numeral dan atau diikuti oleh frasa nominal, frasa verbal, frasa preposisional, klausa, dan/ atau penentu. Frasa nominal juga dapat dibentuk dari klausa (sebagai pemerlengkapan) yang dapat didahului oleh nomina (Lapolika, 1990).

Menurut Alwi dkk., (2003), frasa nominal dapat dibentuk dari nomina dengan memperluas nomina tersebut ke kiri dan/ atau ke kanan. Perluasan ke kiri dilakukan dengan meletakkan frasa numeralia di depan nomina. Perluasan ke kanan dapat memiliki beberapa macam bentuk dengan mengikuti kaidah sebagai berikut (Alwi dkk., 2003):

1. Suatu inti dapat diikuti oleh satu nomina lain atau lebih. Rangkaian tersebut dapat ditutup oleh pronomina persona dan pronomina penunjuk ini atau itu.
2. Suatu inti dapat diikuti oleh adjektiva, pronomina atau frasa pemilikan, dan kemudian ditutup dengan pronomina penunjuk ini atau itu.
3. Jika suatu nomina diikuti oleh adjektiva dan tidak terdapat pewatas lain yang mengikutinya, kata yang dapat disisipkan di antara nomina dan adjektiva. Akan tetapi kata "yang" dan "adjektiva" harus dipindahkan ke belakang jika di dalam frasa tersebut terdapat pronomina.
4. Suatu inti dapat diikuti verba tertentu yang pada hakikatnya dapat dipisahkan oleh kata "yang", "untuk", atau unsur tertentu.
5. Suatu inti dapat diperluas dengan aposisi, yaitu frasa nominal yang mempunyai acuan yang sama dengan nomina yang diterangkannya.

6. Suatu inti dapat diperluas dengan pewatas belakang, yakni klausa yang dimulai dengan kata "yang".
7. Suatu inti dapat diperluas oleh frasa preposisional. Frasa preposisional yang menjadi pewatas nomina tersebut merupakan bagian dari frasa nominal sehingga tidak dapat dipindahkan ke tempat lain.

Frasa Pronominal

Pronomina dapat diperluas menjadi frasa pronominal. Menurut Alwi dkk., (2003), perluasan pronomina menjadi frasa pronominal dapat dilakukan dengan menambahkan numeralia kolektif, menambahkan kata penunjuk, menambahkan kata sendiri, menambahkan klausa yang diawali oleh kata "yang", atau menambahkan frasa nominal yang memiliki fungsi apositif.

Frasa Numeralia

Menurut Alwi dkk., (2003), numeralia dapat diperluas menjadi frasa numeralia. Frasa numeralia dapat dibentuk dengan menambahkan kata penggolong di belakang numeralia.

Frasa Preposisional

Frasa preposisional dibentuk dari preposisi dan frasa nominal. Meskipun frasa preposisional mempunyai makna yang beragam, struktur frasa preposisional relatif seragam (Lapoliwa, 1990).

3.1.3 Klausa

Menurut Alwi dkk., (2003) istilah klausa dipakai untuk merujuk pada deretan kata yang paling tidak memiliki subjek dan predikat. Klausa merupakan deretan kata tanpa intonasi atau tanda baca tertentu, suatu klausa bisa menjadi kalimat yang berbeda-beda tergantung pada intonasi atau tanda baca yang dipakai.

3.1.4 Kalimat

Alwi dkk., (2003) memaparkan bahwa kalimat adalah rentetan kata yang disusun sesuai dengan kaidah yang berlaku. Tiap kata dalam kalimat mempunyai tiga klasifikasi, yaitu: kategori sintaksis, fungsi sintaksis, peran semantik. Klasifikasi tersaji pada Gambar 3.1.

Kalimat	Ibu	memarahi	adi
	↓	↓	↓
Kategori Sintaktis	Nomina	Verba	Nomina
Fungsi Sintaktis	Subjek	Predikat	Objek
Peran Semantis	Pelaku	Perbuatan	Sasaran

Gambar 3.1 Klasifikasi kata dalam sebuah kalimat

Menurut Alwi dkk., (2003) kalimat merupakan konstruksi sintaksis terbesar yang terdiri dari dua kata atau lebih. Penggabungan dua kata atau lebih dalam satu kalimat menuntut adanya keserasian diantara unsur-unsur tersebut, baik dari segi makna maupun dari segi bentuk. Kalimat dapat dikelompokkan berdasarkan jumlah klausa yang menyusun kalimat. Berdasarkan jumlah klausanya, kalimat dapat dibedakan menjadi (Alwi dkk., 2003):

1. Kalimat tunggal

Kalimat tunggal adalah kalimat yang terdiri atas satu klausa, sehingga konstituen untuk unsur subjek dan predikat hanya ada satu atau merupakan satu kesatuan. Kalimat tunggal dapat mengandung unsur manasuka seperti keterangan tempat, waktu, dan alat.

2. Kalimat majemuk

Kalimat majemuk disusun oleh lebih dari satu klausa. Klausa-klausa yang terdapat di dalam kalimat majemuk bertingkat dapat dihubungkan dengan dua cara yaitu:

- Koordinasi adalah menghubungkan dua klausa atau lebih yang memiliki kedudukan konstituen yang sama. Disebut juga dengan kalimat majemuk setara.
- Subordinasi adalah menghubungkan dua klausa atau lebih yang kedudukan konstituennya tidak sama. Hubungan yang dibangun dengan menggunakan subordinasi dapat bersifat melengkapi (komplementatif) atau bersifat mewatasi atau menerangkan (atributif). Disebut juga dengan kalimat majemuk bertingkat.

Berdasarkan bentuk atau kategori sintaksisnya, kalimat dapat dibagi ke dalam empat kelompok, yaitu (Alwi dkk., 2003):

1. Kalimat berita (kalimat deklaratif)

Kalimat deklaratif umumnya digunakan untuk menyampaikan pernyataan sehingga isinya merupakan berita bagi pendengar atau pembacanya.

2. Kalimat perintah (kalimat imperatif)

Dardjowidjojo dkk., (1988) mendefinisikan kalimat imperatif sebagai kalimat yang maknanya memberikan perintah untuk melakukan sesuatu.

3. Kalimat tanya (kalimat interogatif)

Dardjowidjojo dkk., (1988) mendefinisikan kalimat interogatif sebagai kalimat yang menanyakan sesuatu atau seseorang. Kalimat interogatif secara formal ditandai oleh kata tanya seperti apa, siapa, berapa, kapan, dan bagaimana, dengan atau tanpa partikel *-kah* sebagai penegas.

4. Kalimat seru (kalimat eksklamatif)

Kalimat seru atau kalimat eksklamatif juga dikenal dengan sebutan kalimat interjeksi. Kalimat seru dapat digunakan untuk menyatakan perasaan kagum atau heran (Alwi dkk., 2003).

Kalimat dapat juga dikelompokkan berdasarkan kelengkapan unsur-unsur yang menyusun kalimat. Berdasarkan kelengkapan unsurnya, yaitu kalimat kalimat lengkap (kalimat major) dan kalimat tak lengkap (kalimat minor). Kalimat lengkap adalah kalimat yang memiliki unsur subjek dan predikat. Kalimat tak lengkap adalah kalimat yang tidak memiliki unsur subjek dan/ atau predikat (Alwi dkk., 2003).

3.2 NLP (*Natural Language Processing*)

Liddy (2001) menyatakan bahwa NLP adalah pengembangan berbagai teknik komputasi untuk menganalisa dan menampilkan teks dalam bahasa alami pada satu atau lebih tingkat analisis linguistik untuk mencapai tujuan manusia dalam hal bahasa yaitu menyelesaikan berbagai tugas dan aplikasi.

Menurut Kao dan Poteet (2007) pemrosesan bahasa alami (NLP) adalah usaha untuk mendapatkan representasi makna dari *free text*. Suryawan (2013)

memaparkan pemrosesan bahasa alami biasanya menggunakan konsep linguistik (seperti nomina, verba, dan lain-lain) dan struktur gramatikal. NLP menggunakan berbagai representasi pengetahuan untuk mengatasi masalah-masalah yang ditemukan, representasi pengetahuan yang digunakan di dalam NLP misalnya kosakata, makna kata, properti gramatikal dan aturan gramatikal. Menurut Kaplan dalam Suprihadi (2005) pemrosesan bahasa terdiri dari:

1. Penganalisis leksikal (*scanner*), menerima *string-string* yang ditulis dalam sebuah bahasa dan mengidentifikasi *substring-substring* elemen bahasa tersebut, *substring* hasil analisis leksikal disebut *token* yang dimasukkan ke bagian parser.
2. Parser, bertanggung jawab membentuk struktur yang merepresentasikan *statement-statement* dalam bahasa tersebut.
3. Pembangkit kode, menghasilkan kode dari hasil parser untuk menentukan jawaban akhir *query*.

Menurut Maier dan Warren dalam Hartati dkk., (2008) komponen pemrosesan bahasa alami terdiri dari :

1. *Scanner*

Berfungsi melakukan analisis leksikal, mengidentifikasi semua besaran yang membangun suatu bahasa pada sumber. *Scanner* menerima *input* berupa rangkaian karakter kemudian memilah dan mengelompokkannya menjadi satuan leksik yang disebut *token*, *token* akan menjadi *input* parser. Tugas *scanner* dapat meliputi: membaca kode sumber dengan meruntut karakter demi karakter, mengenali besaran leksik dan mentransformasikan menjadi sebuah *token*, menentukan jenis *token*, mengirimkan *token*, membuang/mengabaikan *blank* dan komentar, menangani kesalahan, menangani simbol.

2. Parser

Mencari berbagai kemungkinan aturan yang dapat digunakan untuk memecah kalimat. Sehingga pemecahan kalimat dapat dipandang sebagai masalah pencarian, *parsing* merupakan analisis sintaks yang berguna untuk memeriksa urutan kemunculan *token*. Barr dan Feigenbaum (1981) memandang *parser* sebagai *recursive pattern matcher* yang melakukan pencarian untuk

memetakan *string* kata-kata ke dalam himpunan pola sintaksis yang bermakna. Menurut Barr dan Feigenbaum (1981) dalam perancangan parser perlu memperhatikan empat hal, yaitu: (1) Keseragaman, (2) Pengetahuan dari banyak sumber, (3) Presisi, (4) Tipe dan hasil.

3. *Translator, Optimizer* dan *Evaluator*

Translator menerima hasil dari proses *parsing*, *translator* berfungsi menghasilkan kode antara sebelum memasuki proses evaluasi untuk menentukan jawaban akhir *query*. Komponen *optimizer* bersifat opsional, berfungsi untuk mengoptimalkan kode agar lebih efisien. Komponen terakhir *evaluator*, berfungsi untuk mengevaluasi daftar tujuan (*goal list*) untuk menentukan jawaban akhir.

Natural Language Processing merupakan kemampuan suatu komputer untuk memproses bahasa, baik lisan maupun tulisan yang digunakan oleh manusia dalam komunikasi sehari-hari. Tujuan dari bidang NLP adalah melakukan proses pembuatan model komputasi dari bahasa, sehingga dapat terjadi interaksi antara manusia dan komputer dengan perantara bahasa alami.

Beberapa komponen dalam pemrosesan bahasa alami tidak bersifat kaku, pada beberapa masalah mungkin hanya mengambil beberapa bagian dari pendekatan tersebut, mungkin ada yang melakukan tambahan proses sesuai dengan karakter bahasa yang digunakan dan sistem yang dibangun.

3.3 Ontologi

Istilah ontologi berasal dari salah satu bidang ilmu filsafat yaitu ilmu mengenai ekstensi alamiah (*The study of the nature of existence*) yang merupakan suatu cabang metafisika yang bersangkutan dengan mengidentifikasi segala hal yang benar-benar ada dan bagaimana mendeskripsikannya (Antoniou dan van Harmellen, 2008). Menurut Aristoteles dalam Sarno dkk., (2012) ontologi adalah studi tentang keberadaan, yakni suatu sistem klasifikasi untuk menjelaskan keberadaan dunia nyata.

Ontologi di dalam kajian ilmu komputer merupakan bagian terpenting dari teknologi semantik *web*, ontologi adalah model dari dunia nyata yang dibentuk

dari serangkaian konsep yang menjelaskan wilayah pengetahuan tertentu.

Ontologi dapat digambarkan sebagai suatu struktur hirarkis yang mengandung definisi kelas, hubungan antar entitas, karakteristik atau properti dan tata aturan yang berlaku disuatu bidang pengetahuan, artinya dengan menggunakan model ontologi suatu data yang disimpan dapat dipandang sebagai bentuk pengetahuan yang kongkrit karena memiliki arti/ makna, saling terkait dan memiliki keterhubungan dengan data yang lain (Azhari dan Scholihah, 2006).

Berdasarkan beberapa pemaparan dalam Sarno dkk., (2012) ditinjau dari sisi *computer science* ontologi memiliki beberapa arti, sebagai berikut:

1. Ontologi merupakan definisi dari pengertian dasar dan relasi kosakata dari suatu domain permasalahan (Neches dkk.,1991).
2. Ontologi merupakan sebuah spesifikasi eksplisit dari konseptualisme (Gruber, 1993).
3. Sebuah ontologi mendefinisikan konsep hubungan, dan perbedaan lain yang relevan untuk pemodelan domain permasalahan (Gruber, 2009).
4. Spesifikasi berbentuk definisi kosakata representasional (kelas, hubungan dan sebagainya) yang memberikan makna untuk kendala kosakata dan formal tentang penggunaan koherennya (Gruber, 2009).
5. Ontologi adalah representasi simbolis tentang pengetahuan obyek, kelas obyek, *property* obyek dan relasi antar obyek untuk merepresentasikan suatu pengetahuan tentang domain aplikasi (Lim dkk., 2011).

Salah satu tujuan dibuatnya ontologi semantik menurut Sarno dkk., (2012) yaitu untuk meningkatkan otomatisasi pemroses teks dengan menyediakan representasi konsep yang ada di dunia secara *language independent* dan *meaning based*. Ontologi juga bermanfaat untuk meningkatkan akurasi dalam proses pencarian informasi di *web*. Dalam *web* semantik untuk merepresentasikan basis pengetahuan dan sumber daya *web*, ontologi menghubungkan simbol-simbol yang dipahami manusia dengan bentuknya yang dapat diproses oleh mesin, dengan demikian ontologi menjadi jembatan antara manusia dan mesin (Davies dkk., 2006). Berdasarkan pemaparan Breitman dkk., (2007) meskipun terjadi perbedaan definisi mengenai sebuah ontologi, terdapat kesamaan yakni sebuah ontologi

memiliki *vocabulary* dari *term-term* dan bagaimana *term-term* tersebut saling berelasi, *term* merujuk pada konsep-konsep dalam sebuah domain.

3.3.1 Komponen ontologi

Ontologi memiliki beberapa komponen penyusun, yaitu kelas (*class*), *instance*, *property* dan relasi (Muller, 2008). Menurut Pirro dkk., (2010) komponen ontologi dapat dikembangkan lagi menjadi konsep, relasi, fungsi, aksiom, dan *instance*. Sarno dkk., (2012) menjabarkan secara umum komponen ontologi sebagai berikut:

1. Konsep (*Concept*) dan Kelas (*Class*)

Konsep merepresentasikan sekumpulan atau satu kelas dari entitas atau *things* dalam suatu domain. Konsep juga dikenal sebagai *classes*, *object* dan *categories*. Konsep dapat berupa deskripsi dari tugas, fungsi, aksi, strategi dan sebagainya. Konsep merupakan himpunan yang abstrak dari suatu obyek, misalnya kelas “bunga” memiliki *instance* “mawar”.

2. Hubungan/Relasi (*Relation*)

Relasi digunakan untuk menjelaskan hubungan antara dua obyek. Relasi menggambarkan sebuah tipe interaksi diantara konsep-konsep pada sebuah domain, misalnya relasi *hasMother*, *hasFather*.

3. Atribut (*Attribute*)

Digunakan untuk menjelaskan properti obyek, atribut terdiri dari sebuah penanda (*tag*) dan nilai yang berkaitan, nilai dapat berupa berbagai tipe, misalnya kelas “bunga” memiliki atribut “warna” dan “aroma”.

4. Fungsi (*Function*)

Sebuah relasi khusus yang menghubungkan elemen ke-n untuk elemen ke n-1 sebagai pendahulunya, misalnya *Mother-of*.

5. Aksioma (*Axiom*)

Sebuah model kalimat yang selalu bernilai *true*. Aksiom digunakan untuk membatasi nilai kelas atau *instance*, misalnya jika seorang mahasiswa mengikuti 2 mata kuliah, mata kuliah A dan mata kuliah B, maka mahasiswa tersebut haruslah mahasiswa di semester kedua.

6. *Instance*

Instance bisa juga disebut sebagai individual, yaitu *ground level* dari ontologi (Muller, 2008). *Instance* merupakan sesuatu yang direpresentasikan oleh konsep, sebagai contoh *instance* dari kelas “bunga” adalah “mawar”.

3.3.2 Metode pengembangan ontologi

Noy dan McGuinness (2001) menjabarkan metode pengembangan ontologi yang terdiri dari tujuh tahapan, yang kemudian disempurnakan oleh Antoniou dan van Harmelen (2008) menjadi delapan tahapan, yaitu:

1. Menentukan domain dan ruang lingkup (*scope*) dari ontologi

Pengetahuan atau informasi tentang suatu domain yang akan direpresentasikan ke dalam ontologi bergantung dari bagaimana ontologi tersebut akan digunakan dan siapa yang akan menggunakan ontologi tersebut. Kebutuhan menentukan bentuk pengetahuan yang akan direpresentasikan ke dalam ontologi, oleh karena itu sebuah ontologi dapat direpresentasikan secara berbeda-beda meskipun untuk sebuah domain yang sama. Noy dan McGuinness (2001) menjabarkan beberapa pertanyaan yang dapat digunakan untuk menentukan domain dan ruang lingkup ontologi yang akan dikembangkan, yaitu:

- Domain apa yang akan dibahas oleh ontologi?
- Ontologi akan digunakan untuk apa?
- Pertanyaan seperti apa saja yang dapat di jawab oleh Informasi yang terdapat di dalam ontologi?
- Siapa yang akan menggunakan dan memelihara ontologi?

Gruninger dan Fox dalam Noy dan McGuinness (2001) menambahkan cara untuk menentukan lingkup ontologi adalah dengan membuat pertanyaan kompetensi (*competency questions*), berupa daftar pertanyaan yang lebih spesifik tentang lingkup yang akan dibahas dalam ontologi dan ontologi yang akan dikembangkan harus memiliki pengetahuan untuk menjawab pertanyaan tersebut. Pertanyaan tersebut juga dapat digunakan pada saat melakukan pengujian dan evaluasi terhadap ontologi.

2. Mempertimbangkan penggunaan ontologi yang telah ada

Noy dan McGuinness (2001) menjabarkan bahwa dalam mengembangkan

suatu ontologi perlu mempertimbangkan penggunaan ontologi lain yang telah ada, sehingga pengembangan ontologi tidak perlu dilakukan dari awal. Terdapat kemungkinan bahwa domain ontologi yang akan dibangun sudah pernah dibentuk sebelumnya, ontologi yang telah tersedia dapat digunakan sebagai dasar dalam pengembangan ontologi baik itu berdasarkan penelitian sebelumnya ataupun ontologi yang telah dikembangkan oleh lembaga atau konsorsium, misalnya FOAF dan SKOS.

3. Menyebutkan istilah-istilah penting di dalam ontologi

Menuliskan semua istilah yang akan digunakan di dalam ontologi. Dengan adanya definisi dan penjelasan istilah yang digunakan akan memudahkan dalam memahami ontologi yang dibangun, sebagai contoh membuat daftar nama *class* dan *property* yang akan digunakan.

4. Mendefinisikan *class* dan hirarkinya

Uschold dan Gruninger dalam Noy dan McGuinness (2001) mengemukakan tiga pendekatan dalam mendefinisikan hirarkhi *class*, yaitu:

- *top down*, dimana proses pengembangan dimulai dari konsep yang paling umum, kemudian dilanjutkan dengan konsep yang spesifik.
- *bottom up*, dimana proses pengembangan dimulai dari *class* yang paling spesifik, kemudian dilanjutkan dengan mengelompokkan *class-class* tersebut menjadi konsep yang lebih umum.
- kombinasi, gabungan dari pendekatan *top down* dan *bottom up*.

5. Mendefinisikan *property*

Setelah *class-class* terbentuk tahap selanjutnya yaitu mendefinisikan struktur internal dari *class*, dapat berupa atribut-atribut yang menjadi karakteristik *class* tersebut. Contohnya dengan membuat *statement* mengenai hubungan antar *class* dan *property* yang digunakan termasuk domain dan range dari *property*.

6. Mendefinisikan *facet*

Property dapat memiliki *facet* yang berbeda untuk mendeskripsikan tipe nilai (*value*), nilai yang dapat diterima, jumlah kardinalitas, dan fitur lain dari nilai *property*. Antoniou dan van Harmelen (2004) mengemukakan

bahwa *facet* yang umumnya digunakan terdiri dari:

- Kardinalitas, yaitu menyatakan banyaknya nilai *property* berbeda yang dapat atau harus dimiliki oleh *property*.
- Tipe nilai menjelaskan tipe nilai yang digunakan didalam *property*.
- Domain dan *range* dari *property* menjelaskan karakteristik relasional dari *property*, yaitu: simetris, transitif, invers dari *property*, dan nilai fungsional (*function value*).

7. Mendefinisikan Individu (*instances*) dari *class*

Noy dan McGuinness (2001) memaparkan dalam pendefinisian *instance* dilakukan dengan langkah-langkah sebagai berikut: (1) pilih sebuah *class*, (2) buat *instance* individual dari *class* tersebut, (3) isi nilai *property* dari *instance*.

8. Memeriksa anomali

Informasi atau data yang tidak konsisten dapat muncul di dalam ontologi, misalnya definisi domain dan *range* yang tidak konsisten pada *property* transitif, *property* simetris, atau pada *property* invers, atau dapat juga inkonsistensi kardinalitas dari nilai *property*.

3.4 Semantik Web

Secara umum dapat dikatakan *web semantik* merupakan sebuah teknologi yang menjadikan *web* menjadi lebih pintar untuk dapat memahami maksud atau keinginan pengguna dan dapat memberikan informasi yang relevan dan spesifik.

McGuinness (2004) menyatakan *web semantik* merupakan suatu visi untuk *web* masa depan, dimana informasi di dalam *web* diberikan arti secara eksplisit, sehingga menjadi lebih mudah diproses oleh mesin secara otomatis dan lebih mudah menyatukan informasi yang tersedia di *web*.

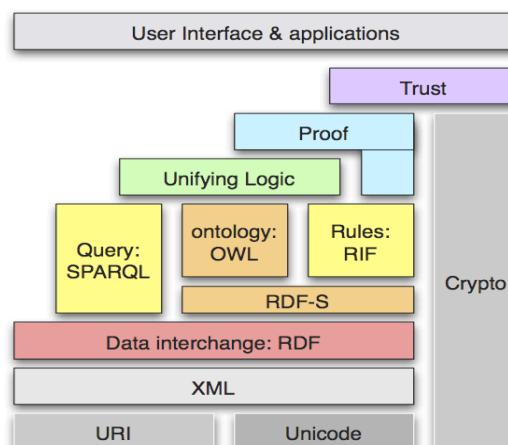
Menurut Berners-Lee dkk., (2001) *web semantik* adalah perluasan dari *web* yang mendukung *database* dalam bentuk yang dapat dipahami oleh mesin. pada dasarnya *web semantik* bertujuan agar *web* konten yang di ekspresikan di dalam bahasa alami yang dimengerti manusia, dapat juga dimengerti, diinterpretasi dan digunakan oleh perangkat lunak. Melalui *web semantik* berbagai

perangkat lunak akan mampu mencari, membagi, dan mengintegrasikan informasi dengan cara yang lebih mudah.

Kata semantik dapat berarti makna atau sesuatu yang berhubungan dengan ilmu yang mempelajari makna dan perubahan makna. Berners-Lee (1998) mengemukakan bahwa dalam konteks *web* semantik kata semantik menunjukkan bahwa makna dari suatu data yang terdapat dalam *web* dapat dipahami bukan hanya oleh manusia namun juga oleh mesin (*machine understandable*).

3.4.1 Arsitektur *web* semantik

Arsitektur *web* semantik terdiri dari beberapa *layer* seperti yang disajikan dalam Gambar 3.2.



Gambar 3.2 Arsitektur *web* semantik (Berners-Lee, 2006)

Arsitektur *web* semantik pada Gambar 3.2 dibagi menjadi 3 *layer*. *Layer* Bawah terdiri dari :

1. XML adalah sebuah *markup language* yang memungkinkan penciptaan dokumen-dokumen yang terdiri dari susunan data yang terstruktur. XML digunakan untuk membangun dokumen *web* terstruktur dengan kosakata yang didefinisikan sendiri oleh pengguna.
2. URI merupakan penamaan yang unik untuk identifikasi sumber dari *web* semantik.
3. *Unicode* berfungsi untuk merepresentasikan dan memanipulasi teks ke dalam banyak bahasa, menjembatani dokumen dalam bahasa manusia yang berbeda sehingga dapat merepresentasikannya.

Layer tengah terdiri dari :

1. RDF (*Resource Description Framework*) merupakan model data dasar yang digunakan untuk menuliskan *statement* sederhana mengenai objek *web* (*resource*). RDF adalah sebuah *framework* untuk membuat pernyataan dalam sebuah bentuk *triple* dan memungkinkan untuk merepresentasikan informasi dari sebuah *source* dalam bentuk *graph*.
2. RDFS (*RDF Schema*) menyediakan permodelan sederhana (*modelling primitive*) untuk mengatur objek *web* ke dalam hirarkhi (Antoniou dan van Harmelen, 2004). RDFS menyediakan dasar-dasar *vocabulary* untuk RDF, dengan RDFS memungkinkan untuk membuat hirarki dari *class* dan *property*-nya.
3. OWL (*Web Ontology Language*) memperluas RDFS dengan menambahkan konsep dan relasi objek yang lebih kompleks untuk mendeskripsikan semantik dari *statement* RDFS. Dengan OWL memungkinkan untuk menambahkan sebuah *constraint*, seperti *cardinality*, batasan nilai, karakteristik dari *property* dan memberikan kekuatan *reasoning* pada *web* semantik.
4. SPARQL adalah bahasa *query* yang digunakan untuk *query* data berbentuk RDF (termasuk statement RDF dan OWL). Bahasa *query* diperlukan untuk merujuk informasi dari aplikasi *web* semantik (DuCharme, 2011).

Sedangkan *layer* atas terdiri dari:

1. RIF (*Rule Interchange Format*) format yang digunakan untuk mendefinisikan pertukaran aturan diantara *rule system*, yaitu diantara *web rule engine*. Misalnya untuk mendeskripsikan relasi yang tidak dapat dideskripsikan secara logika pada OWL.
2. Kriptografi untuk memastikan dan memverifikasi bahwa *semantik web statement* berasal dari sumber yang terpercaya. Misalnya dengan menggunakan “*digital signature*” dari pernyataan RDF.
3. *User Interface*, merupakan *layer* terakhir yang memungkinkan manusia untuk menggunakan aplikasi *web* semantik.

3.4.2 XML

Extensible Markup Language (XML) merupakan sebuah bahasa *markup* yang didesain untuk mendeskripsikan dokumen secara terstruktur. Berbeda dengan HTML, XML memungkinkan penggunanya untuk mendefinisikan sendiri *tag-tag* yang diinginkan (Breitman dkk., 2007). Selain itu XML memungkinkan representasi informasi dalam bentuk yang mudah dipahami oleh manusia sekaligus dapat diproses oleh mesin. Kelebihan lainnya, XML memisahkan antara konten (*content*) dan cara konten tersebut dipresentasikan (*presentation*) (Antoniou & van Harmelen, 2008). Contoh dokumen XML disajikan pada gambar 3.3.

```
<?xml version="1.0" encoding="UTF-16"?>
<!DOCTYPE email SYSTEM "email.dtd">
<email>
    <head>
        <from name="Michael Maher"
              address="michaelmaher@cs.gu.edu.au"/>
        <to name="Grigoris Antoniou"
              address="grigoris@cs.unibremen.de"/>
        <subject>Where is your draft?</subject>
    </head>

    <body>
        Grigoris, where is the draft of the
        paper
        You promised me last week?
    </body>
</email>
```

Gambar 3.3 Contoh dokumen XML (Antoniou dan van Harmellen, 2008)

XML *Schema* merupakan bahasa yang digunakan untuk mendefinisikan sekumpulan aturan (*schema*) atau struktur yang harus diikuti oleh dokumen XML. Struktur dari dokumen XML yang dibuat harus sesuai dengan *schema* yang telah didefinisikan untuk menjaga validitas dokumen tersebut. Contoh XML *schema* tersaji pada gambar 3.4

Sebuah dokumen XML dapat menggunakan beberapa *schema*. Jika *schema-schema* tersebut mendefinisikan sebuah tipe elemen dengan cara yang berbeda, dibutuhkan suatu cara untuk mengetahui *schema* mana yang digunakan untuk mendefinisikan elemen tersebut dalam dokumen XML, yaitu dengan menggunakan XML *namespace*.

```

<element name="email" type="emailType"/>
<complexType name="emailType">
    <sequence>
        <element name="head" type="headType"/>
        <element name="body" type="bodyType"/>
    </sequence>
</complexType>

<complexType name="headType">
    <sequence>
        <element name="from" type="nameAddress"/>
        <element name="to" type="nameAddress"
            minOccurs="1" maxOccurs="unbounded"/>
        <element name="cc" type="nameAddress"
            minOccurs="0" maxOccurs="unbounded"/>
        <element name="subject" type="string"/>
    </sequence>
</complexType>

<complexType name="nameAddress">
    <attribute name="name" type="string" use="optional"/>
    <attribute name="address" type="string" use="required"/>
</complexType>

```

Gambar 3.4 Contoh dokumen XML *Schema* (Antoniou dan van Harmellen, 2008)

XML *namespace* merupakan sebuah koleksi yang berisi nama-nama elemen beserta tipe dan atributnya pada sebuah *schema* (Dykes & Tittel, 2005). Sebuah *namespace* didefinisikan dalam bentuk *xmlns:prefix="location"* seperti tersaji pada gambar 3.5.

```

<?xml version="1.0" encoding="UTF-16"?>
<vu:instructors
    xmlns:vu="http://www.vu.com/empDTD"
    xmlns:gu="http://www.gu.au/empDTD"
    xmlns:uky="http://www.uky.edu/empDTD">
    <uky:faculty
        uky:title="assistant professor"
        uky:name="John Smith"
        uky:department="Computer Science"/>
    <academicStaff
        title="lecturer"
        name="Mate Jones"
        school="Information Technology"/>
</vu:instructors>

```

Gambar 3.5 Contoh dokumen XML *Namespace* (Antoniou dan van Harmellen, 2008)

3.4.3 RDF (*Resource Description Framework*)

Resource Description Framework (RDF) merupakan model data untuk obyek (*resource*) dan relasi antar obyek. Menurut Breitman dkk., (2007) RDF merupakan sebuah model data yang sederhana dan fleksibel untuk mendeskripsikan hubungan antara sumber daya-sumber daya *web* dalam bentuk

RDF *statement*. RDF mendukung interoperabilitas antar aplikasi yang melakukan pertukaran informasi dan bersifat *machine-understandable* di *web*.

Antoniou dan van Harmellen (2008) menyatakan konsep dasar dari RDF adalah *resources*, *properties* dan *statements*. Lassila dan Swick (1999) menyatakan model data RDF terdiri atas tiga objek tipe :

1. *Resource*

Segala sesuatu yang digambarkan dengan RDF disebut *resource*. *Resource* bisa berupa keseluruhan atau bagian dari sebuah halaman *web*. *Resource* ini biasanya diberi nama menggunakan URI (*Uniform Resource Identifier*). URI bersifat bisa diperluas maka URI bisa digunakan sebagai pengenal bagi berbagai macam entitas.

2. *Property*

Property merupakan aspek atau karakteristik, atribut, serta relasi khusus yang digunakan untuk menggambarkan sebuah *resource*. Setiap *property* memiliki arti khusus, mendefinisikan nilai yang mungkin, tipe *resource* yang digambarkan dan relasinya dengan properti lain.

3. Pernyataan (*Statement*)

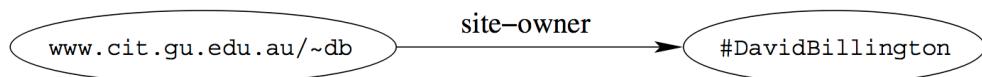
Suatu *resource* bersama dengan *property* dan nilai dari suatu *property* untuk *resource* membentuk suatu RDF *statement*. RDF *Statement* dinyatakan dalam *triple* yaitu: (S) subjek, (P) predikat dan (O) objek. RDF *statement* dapat dituliskan ke dalam tiga bentuk notasi (Antoniou dan van Harmellen, 2008 ; Breitman dkk., 2007) yaitu:

- a. RDF *triple*, notasi ini mentranslasikan RDF *statement* menjadi karakter *string* secara langsung. Notasi untuk himpunan RDF *statement* terdiri dari RDF *triple*, bagian yang mengidentifikasi dalam *statement* dapat disebut subjek, karakteristik (*property*) dari subjek disebut sebagai predikat, sedangkan nilai dari *property* disebut sebagai objek. Masing-masing merepresentasikan RDF *statement* di dalam himpunan tersebut, yang dirangkaikan tanpa memperhatikan urutan RDF *triple*. Contoh RDF *statement* yang dinyatakan dengan notasi RDF *triple* disajikan dalam Gambar 3.6.

```
(http://www.cit.gu.edu.au/~db,
http://www.mydomain.org/site-owner, #DavidBillington).
```

Gambar 3.6 Contoh RDF *statement* yang dinyatakan dengan notasi RDF *triple* (Antoniou dan van Harmelen, 2008)

- b. RDF *graph*, notasi ini mentranslasikan RDF *statement* menjadi *graph*, dimana *node* merepresentasikan subjek atau objek, dan busur berarah merepresentasikan *property*. Notasi untuk himpunan RDF *statement* berupa *labeled graph*, Contoh RDF *statement* yang dinyatakan dengan notasi RDF *graph* disajikan dalam Gambar 3.7.



Gambar 3.7 Contoh RDF *statement* yang dinyatakan dengan notasi RDF *graph* (Antoniou dan van Harmelen, 2008)

- c. RDF/XML menyediakan notasi XML untuk RDF *statement*, dan merupakan notasi yang disarankan dalam konteks pertukaran data diantara agen perangkat lunak (*software agent*). Menurut Antoniou dan van Harmelen (2008) penggunaan *tool graf* dalam representasi RDF adalah cara yang tepat untuk dapat dipahami oleh manusia (*human-understanding*), namun visi *web semantik* membutuhkan representasi yang dapat diakses dan diproses oleh mesin (*machine-precessable*). Oleh karena itu, alternatif representasi yang lain adalah dengan menggunakan pernyataan (*statement*) dalam format XML. Sebuah dokumen RDF diwakili oleh elemen XML dengan tag *rdf:RDF*. Contoh RDF *statement* yang dinyatakan dalam notasi RDF/XML disajikan dalam Gambar 3.8.

```

<?xml version="1.0" encoding="UTF-16"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:mydomain="http://www.mydomain.org/my-rdf-ns">

    <rdf:Description rdf:about="http://www.cit.gu.edu.au/~db">
        <mydomain:site-owner rdf:resource="#DavidBillington"/>
    </rdf:Description>
</rdf:RDF>
  
```

Gambar 3.8 Contoh RDF *statement* yang dinyatakan dalam notasi RDF/XML (Antoniou dan van Harmelen, 2008)

3.4.4 RDFS (RDF Schema)

RDFS atau RDF *Schema* adalah kosakata untuk menjelaskan *property* dan *class* dari sumber RDF. RDF secara fleksibel dapat menguraikan *resource* dengan *class-class*, *property-property*, dan nilai yang ada di *property* akan tetapi RDF tidak menyediakan mekanisme untuk mendefinisikan *class* serta *property* yang spesifik untuk aplikasi, RDF memerlukan suatu cara untuk menggambarkan spesifikasi *class* dan *property* yaitu dengan menggunakan RDFS, dapat dikatakan RDFS merupakan perluasan dari RDF. Contoh penggunaan RDFS tersaji pada Gambar 3.9.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:myCamera="http://www.liyangyu.com/camera#"
    xml:base="http://www.liyangyu.com/camera#"
    >
    <rdfs:Class rdf:about="http://www.liyangyu.com/camera#Camera">
        </rdfs:Class>
        <rdfs:Class rdf:about="http://www.liyangyu.com/camera#Lens">
            </rdfs:Class>
        <rdfs:Class rdf:about="http://www.liyangyu.com/camera#Body">
            </rdfs:Class>
        <rdfs:Class rdf:about="http://www.liyangyu.com/camera#ValueRange">
            </rdfs:Class>
    </rdfs:Class>
</rdf:RDF>
```

Gambar 3.9 Contoh definisi *class* dengan menggunakan kosakata RDFS yang berbasis notasi RDF/XML (Yu, 2010)

Menurut Yu (2010) RDFS adalah suatu bentuk bahasa yang bisa digunakan untuk mendefinisikan kosakata, yang kemudian dapat digunakan untuk membentuk struktur dokumen RDF. Breitman dkk., (2007) menyatakan RDFS dapat dibagi menjadi kelompok berdasarkan kegunaannya, sebagai berikut:

1. *Class*

Resource dapat dibagi menjadi kelompok-kelompok yang disebut dengan *class*. Anggota dari *class* disebut dengan *instance* dari *class*. *Class* di dalam RDFS itu sendiri merupakan suatu *resource*. *Class* diidentifikasi dengan menggunakan URIref dan dideskripsikan dengan menggunakan *property* RDF. *Property* *rdf:type* digunakan untuk menyatakan bahwa suatu *resource* adalah *instance* dari suatu *class*, *property* RDFS yang dapat digunakan untuk mendefinisikan *class*. yaitu: *rdfs:Resource*, *rdfs:Class*, *rdfs:Literal*, *rdfs:Datatype*.

2. Properties

Mencakup istilah RDFS yang dapat digunakan untuk menentukan *properties*. *Properties* adalah *instance* dari *class rdfs:Property*, yaitu: *rdfs:range*, *rdfs:domain*, *rdfs:subClassOf*, *rdfs:subPropertyOf*, *rdfs:label* dan *rdfs:comment*. *Property rdfs:domain* digunakan untuk menyatakan domain dari *class* dan *property rdfs:range* digunakan untuk menyatakan bahwa nilai dari properti tersebut adalah *instance* dari *class* tertentu atau *instance* dari *XML Schema datatype* (Breitman dkk., 2007). *Property rdfs:subPropertyOf* digunakan untuk menyatakan suatu *property* adalah subproperti dari *property* lainnya (Brickley dan Guha, 2004).

3. Individu

Individu dapat dikatakan *instance* dari *class*. *Instance* dari suatu *class* adalah *resource* yang nilai dari properti *rdf:type*-nya adalah *class* tersebut. Sebuah *resource* dapat menjadi *instance* dari beberapa *class* sekaligus (Breitman dkk., 2007).

3.4.5 OWL (*Ontology Web Language*)

OWL merupakan jawaban atas kebutuhan akan bahasa pemodelan ontologi yang lebih *powerfull* dari RDF dan RDFS (Antoniou dan van Harmellen, 2004). Menurut Yu (2010) tujuan dari OWL sama seperti RDFS yaitu untuk mendefinisikan ontologi termasuk *class*, *property*, *relationship* untuk domain aplikasi yang spesifik, namun dibandingkan dengan RDFS OWL menyediakan kemampuan untuk mengekspresikan *relationship* yang lebih kaya dan kompleks, sehingga dapat membangun aplikasi dengan kemampuan *reasoning* yang lebih kuat.

OWL didefinisikan sebagai kosakata seperti halnya RDF dan RDFS, akan tetapi OWL memiliki semantik yang lebih kaya. Suatu ontologi di dalam OWL merupakan himpunan *triple* RDF yang menggunakan kosakata OWL. Tujuan dari penggunaan OWL adalah agar mesin dapat menginterpretasikan *content* dari *Web* (Breitman dkk., 2007).

OWL digunakan untuk menambahkan beberapa kosakata yang mendeskripsikan *properties* dan *class*, antara lain: relasi antara *class* (misalnya

disjointness), kardinalitas, *equality*, berbagai tipe dari *property*, karakteristik dari *property*. Perbedaan OWL dan RDF adalah sifat kardinalitas dan adanya validasi.

OWL menyediakan tiga sub bahasa yang berbeda tingkatan bahasa-nya yang dirancang untuk berbagai kebutuhan tertentu dari pengguna (Breitman dkk., 2007 ; Antoniou & van Harmelen., 2008), antara lain:

1. OWL *Lite*

OWL *Lite* menyediakan pendefinisian klasifikasi hirarkhi *class* dan *properties* dengan batasan-batasan (*constraints*) sederhana (McGuinness dan van Harmellen, 2008). Menurut Antoniou dan van Harmellen (2008) bahwa OWL *Lite* harus memenuhi syarat sebagai ontologi OWL DL serta mengikuti ketentuan sebagai berikut:

- a. Tidak mengandung *owl:oneOf*, *owl:unionOf*, *owl:complementOf*, dan *owl:hasValue*.
- b. *Statement* kardinalitas hanya dapat memiliki value 0 dan 1.
- c. *Statement owl:equivalentClass* tidak dapat diimplementasikan pada *class* anonim.

2. OWL DL (*Description Logic*)

Suryawan (2013) menjabarkan OWL DL ditujukan untuk mendapatkan ekspresivitas maksimum dengan tetap mempertahankan *computational completeness* (dapat dilakukan komputasi untuk memperoleh kesimpulan) dan *decidability* (komputasi dapat selesai dalam waktu tertentu). OWL DL dapat menggunakan kosakata dari OWL *Full*, akan tetapi penggunaannya harus memenuhi ketentuan tertentu (McGuinness dan van Harmellen, 2004).

Antoniou dan van Harmellen (2008) menyatakan bahwa untuk mempermudah pengaturan komputasional dari *Description Logic* maka OWL DL harus memenuhi ketentuan sebagai berikut:

- a. Partisi kosakata, dimana suatu *resource* hanya dapat menjadi salah satu diantara *class*, tipe data, *datatype property*, *object property*, individu, nilai data, atau bagian dari kosakata terdefinisi (*built-in vocabulary*).
- b. Partisi dari *resource* harus dinyatakan secara eksplisit.
- c. *Property* *owl:FunctionalProperty*, *owl:SymmetricProperty*

owl:inverseFunctionalProperty, dan *owl:inverseOf* tidak dapat digunakan untuk menspesifikasikan *datatype property* karena *object property* dan *datatype property* bersifat *disjoint*.

- d. Batasan kardinalitas tidak dapat diterapkan pada *transitive property* beserta seluruh subpropertinya.
- e. Pembatasan pada *class* anonim, dimana *class* anonim dapat menjadi domain atau *range* pada *property* *owl:equivalentClass* atau *owl:disjointWith*, dan hanya dapat menjadi *range* pada properti *rdfs:subClassOf*.

Kelebihan dari OWL DL adalah memungkinkan untuk mendapatkan dukungan penalaran secara efisien. Kekurangan dari OWL DL adalah kehilangan kompatibilitas penuh dengan RDF. Dokumen RDF harus dikembangkan dengan cara tertentu dan dibatasi dengan cara tertentu sebelum menjadi dokumen OWL DL yang legal. Semua dokumen OWL DL yang legal merupakan dokumen RDF yang legal (Antoniou dan van Harmellen, 2008).

3. OWL *Full*

OWL *Full* mencakup semua kosakata OWL (Antoniou dan van Harmellen., 2008). OWL *Full* ditujukan untuk mendapatkan ekspresivitas maksimum dan kebebasan sintaks dari RDF dengan mengabaikan jaminan kemampuan komputasional (Suryawan 2014). OWL *Full* memungkinkan penambahan makna dari kosakata yang belum digambarkan, baik kosakata RDF maupun kosakata OWL (McGuinness dan van Harmellen, 2008).

Kelebihan dari OWL *Full* adalah sepenuhnya kompatibel dengan RDF, baik secara sintaksis maupun secara semantik. Kekurangan dari OWL *Full* adalah ekspresivitas dari OWL *Full* yang sangat *powerfull* membuat dukungan penalaran (*reasoning*) menjadi sangat sulit untuk dilakukan (Antoniou dan van Harmellen, 2008).

3.4.6 SPARQL

SPARQL adalah bahasa *query* yang digunakan untuk mencari data pada basis pengetahuan ontologi yang dinyatakan dalam RDF. SPARQL digunakan

untuk melakukan *query* isi data RDF dan juga menyediakan protokol yang dapat digunakan jika ingin melakukan *query* dari *remote* data RDF (Yu, 2010).

SPARQL dapat digunakan mencari data pada sumber data yang beragam, baik data yang disimpan dengan menggunakan RDF atau data yang dipandang sebagai RDF melalui *middleware* (Prud'hommeaux dkk., 2008). Menurut Yu (2010) kegunaan dari SPARQL antara lain, yaitu:

1. Untuk melakukan *query* ke dalam RDF dan mendapatkan informasi yang spesifik.
2. Untuk melakukan *query* ke dalam *remote server* RDF dan mendapatkan *streamming* hasil balik.
3. Untuk menjalankan *query* otomatis terhadap Data RDF dan menghasilkan bentuk laporan.
4. Memungkinkan untuk pengembangan aplikasi pada *high level*, yaitu, aplikasi dapat bekerja dengan hasil *query* SPARQL, tidak secara langsung dengan RDF *statement*.

SPARQL memiliki empat bentuk *query* (Yu, 2010), yaitu:

1. SELECT

SELECT adalah bentuk yang paling banyak digunakan karena sebagian besar dari fitur-fiturnya dapat digunakan oleh bentuk-bentuk *query* lainnya. *Query* SPARQL dalam bentuk SELECT digunakan untuk memperoleh seluruh atau sebagian variabel yang sesuai dengan *statement* yang dinyatakan oleh *query*. *Output* yang dikembalikan oleh *query* SPARQL dalam bentuk SELECT adalah variabel dan nilai dari variabel tersebut secara langsung (Prud'hommeaux dkk., 2008). Contoh dari *query* SPARQL dalam bentuk SELECT disajikan dalam Gambar 3.10.

```
base <http://danbri.org/foaf.rdf>
prefix foaf: <http://xmlns.com/foaf/0.1/>

select *
from <http://danbri.org/foaf.rdf>
where {
    <#danbri> ?property ?value
}
```

Gambar 3.10 Contoh *query* SPARQL dalam bentuk SELECT (Yu, 2010)

2. CONSTRUCT

Query SPARQL dalam bentuk CONSTRUCT digunakan untuk memperoleh RDF *graph*. *Graph* RDF yang dihasilkan dengan cara mengambil masing-masing solusi dalam urutan solusi, mengganti variabel yang terdapat di dalam *graph template*, dan mengkombinasikan *triple-triple* hasil penggantian variabel ke dalam sebuah *graph* RDF dengan menggunakan operasi penggabungan (Prud'hommeaux dkk., 2008). Contoh dari *query SPARQL* dalam bentuk CONSTRUCT disajikan dalam Gambar 3.11.

```

prefix foaf: <http://xmlns.com/foaf/0.1/>

construct {
    ?person a foaf:Person.
    ?person foaf:name ?name.
    ?person foaf:mbox ?email.
}
from <http://danbri.org/foaf.rdf>

where {
    ?person a foaf:Person.
    ?person foaf:name ?name.
    ?person foaf:mbox ?email.
}

```

Gambar 3.11 Contoh *query SPARQL*, CONSTRUCT (Yu, 2010)

3. DESCRIBE

Query SPARQL dalam bentuk DESCRIBE digunakan untuk memperoleh sebuah *graph* RDF yang memuat data RDF mengenai suatu *resource*. Data mengenai suatu *resource* yang dikembalikan tidak dinyatakan secara eksplisit di dalam *query SPARQL*, sehingga pembuat *query* tidak harus mengetahui struktur dari RDF yang menjadi sumber data (Prud'hommeaux dkk., 2008). Contoh dari *query SPARQL* dalam bentuk DESCRIBE disajikan dalam Gambar 3.12.

```

prefix foaf: <http://xmlns.com/foaf/0.1/>

describe ?x
from <http://danbri.org/foaf.rdf>
where {
    ?x foaf:mbox <mailto:timbl@w3.org>.
}

```

Gambar 3.12 Contoh *query SPARQL*, DESCRIBE (Yu, 2010)

4. ASK

Query SPARQL dalam bentuk ASK diidentifikasi dengan *keyword* ASK, dan prosesor *query* hanya mengembalikan nilai benar atau salah, tergantung pada apakah pola grafik tertentu memiliki kecocokan dalam *dataset* atau tidak (Yu, 2010). ASK juga digunakan untuk menguji apakah suatu pola *query* memiliki solusi atau tidak. *Query* SPARQL dalam bentuk ASK tidak mengembalikan solusi dari *query*, *query* ini hanya mengembalikan informasi apakah *query* tersebut memiliki solusi atau tidak (Prud'hommeaux dkk., 2008). Contoh dari *query* SPARQL dalam bentuk ASK disajikan dalam Gambar 3.13.

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
ask
from <http://danbri.org/foaf.rdf>
where
{
    ?x foaf:mbox <mailto:danbri@danbri.org> .
}
```

Gambar 3.13 Contoh dari *query* SPARQL dalam bentuk ASK (Yu, 2010)

3.5 Peta

Peta adalah representasi dari permukaan bumi atau bagian dari permukaan bumi pada kertas atau media lainnya. Informasi yang terdapat pada peta dapat berupa letak ataupun bentuk geografis dari suatu lokasi tertentu (Clark, 1990).

Dilihat dari sifatnya, terdapat dua macam peta yaitu peta topografi dan peta tematik. Peta topografi adalah peta yang berisi informasi mengenai bentuk permukaan bumi, dapat berupa gambaran unsur-unsur alam, seperti sungai, laut, gunung ataupun berupa gambaran unsur-unsur buatan manusia, seperti perumahan serta pelabuhan. Sedangkan peta tematik merupakan peta yang memiliki suatu tema tertentu, atau menggabungkan beberapa unsur-unsur tertentu yang memiliki kesamaan. Contohnya adalah peta jaringan (jaringan telekomunikasi, jaringan listrik, jaringan irigasi), peta ketinggian (kontur, *Digital Terrain Model / Digital Elevation Model*), serta peta tata guna lahan (*land use*) seperti sawah, hutan, kebun, ladang.

Peta merupakan gambaran wilayah geografis bagian permukaan bumi yang dapat disajikan dalam berbagai cara yang berbeda, mulai dari peta konvensional yang tercetak hingga peta digital yang tampil di layar komputer (peta digital). Bentuk peta digital yang paling sederhana adalah memindahkan media peta yang sebelumnya kertas menjadi gambar pada komputer, misal JPEG tanpa adanya *database* dengan kemampuan interaktif.

Peta interaktif adalah peta digital yang memungkinkan pengguna untuk memasukkan elemen-elemen lain seperti bunyi, gambar dan data-data spasial. Interaktif pada peta sendiri dapat didefinisikan suatu kejadian aksi dan reaksi antara *user* dan peta, dimana pengguna tidak hanya diam atau bersifat pasif namun tersedia fasilitas bagi pengguna untuk memilih atau mengatur arah aplikasi peta sesuai dengan keinginannya.

Peta interaktif lebih serba guna dan dinamis karena bisa menunjukkan banyak view yang berbeda dengan subjek yang sama. Peta ini juga memungkinkan perubahan skala, animasi gabungan, gambar, suara, dan bisa terhubung ke sumber informasi tambahan melalui internet. Peta dapat di-*update* ke peta tematik baru dan bisa menambahkan detail informasi geografi lainnya (Denny dan Agtrisari, 2003).

Denny dan Agtrisari (2003) memaparkan terdapat tiga informasi umum yang dapat dimasukkan pada peta digital, yaitu :

- Informasi geografis, menyediakan informasi mengenai posisi dan bentuk-bentuk dari fitur geografis yang spesifik
- Informasi atribut, menyediakan informasi non-grafis tambahan mengenai tiap-tiap fitur
- Informasi tampilan, menjabarkan informasi mengenai bagaimana tampilan fitur pada layar

3.6 KML (*Keyhole Markup Language*)

Anonim (2013) menjelaskan KML adalah format file standar internasional yang dikelola oleh *Open Geospatial Consortium* (OGC). KML digunakan untuk menampilkan data geografis pada *Earth Browser*, seperti *Google Earth* dan

Google Maps. File KML memiliki fitur-fitur untuk mendeskripsikan data geografis seperti menentukan letak/ lokasi (*placemark*), lapisan tanah (*ground overlays*), dan jalur/ jalan (*path*), bentuk bidang (*polygons*).

KML menggunakan struktur berbasis *tag* dengan elemen bersarang (*nested*) dan atribut yang didasarkan pada standar XML. Semua *tag* bersifat *case-sensitive* dan harus mengikuti aturan yang telah ditentukan. *Tag* standar yang dapat digunakan, yaitu:

1. *Placemark*

Placemark adalah salah satu fitur yang paling umum digunakan, *Placemark* digunakan untuk menandai posisi di permukaan bumi, penanda dapat berupa *icon-icon* yang dapat disesuaikan. Penggunaan *placemark* tersaji pada Gambar 3.14.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>Simple placemark</name>
    <description>placemark example</description>
    <Point>
      <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

Gambar 3.14 Contoh penggunaan *placemark*

Placemark ditandai dengan *tag* *<Placemark>* di dalamnya dapat ditambahkan elemen seperti *<name>* untuk memberikan label, *<Point>* yang menentukan lokasi *placemark* berdasarkan titik koordinat.

2. *Ground Overlay*

Ground overlay digunakan untuk meletakkan gambar (*image*) ke dalam peta. Penggunaan *ground overlay* disajikan pada gambar 3.15. Elemen *<Folder>* pada Gambar 3.15 digunakan untuk mengelompokkan isi dan label data *overlay*, elemen *<Icon>* berisi *link* yang merujuk file dengan format gambar yang digunakan pada *overlay*, *<LatLonBox>* mengatur posisi dari gambar *overlay*. *Ground overlay* juga mendukung penggunaan gambar dengan format JPEG, BMP, GIF, TIFF, TGA, dan PNG.

```

<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Folder>
    <name>Ground Overlays</name>
    <description>Examples of ground overlays</description>
    <GroundOverlay>
      <Icon>
        <href>
          http://developers.google.com/kml/documentation/images/etna.jpg
        </href>
      </Icon>
      <LatLonBox>
        <north>37.91904192681665</north>
        <south>37.46543388598137</south>
        <east>15.35832653742206</east>
        <west>14.60128369746704</west>
        <rotation>-0.1556640799496235</rotation>
      </LatLonBox>
    </GroundOverlay>
  </Folder>
</kml>

```

Gambar 3.15 Contoh penggunaan *Ground Overlay*

3. Path/Polyline

KML mendukung pembuatan *path* yang dapat digunakan untuk menggambarkan jalan, jalur atau rute. Di dalam file KML *path* didefinisikan oleh elemen *<LineString>*. Contoh penggunaan *path* tersaji pada Gambar 3.16.

```

<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
    <name>Paths</name>
    <description>Examples of paths</description>
    <Placemark>
      <LineString>
        <altitudeMode>absolute</altitudeMode>
        <coordinates> -112.2550785337791,36.07954952145647,2357
                      -112.2549277039738,36.08117083492122,2357
                      -112.2552505069063,36.08260761307279,2357
                      -112.2564540158376,36.08395660588506,2357
                      -112.2580238976449,36.08511401044813,2357
                      -112.2595218489022,36.08584355239394,2357
        </coordinates>
      </LineString>
    </Placemark>
  </Document>
</kml>

```

Gambar 3.16 Contoh penggunaan *path/polyline*

4. *Polygon*

Polygon digunakan untuk membuat berbagai bentuk atau pola keruangan sederhana. Contoh *polygon* tersaji pada Gambar 3.17.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>polygon example</name>
    <Polygon>
      <altitudeMode>relativeToGround</altitudeMode>
      <LinearRing>
        <coordinates>
          -77.05788457660967,38.87253259892824,100
          -77.05465973756702,38.87291016281703,100
          -77.05315536854791,38.87053267794386,100
          -77.05552622493516,38.868757801256,100
          -77.05844056290393,38.86996206506943,100
          -77.05788457660967,38.87253259892824,100
        </coordinates>
      </LinearRing>
    </Polygon>
  </Placemark>
</kml>
```

Gambar 3.17 Contoh penggunaan *polygon*

BAB IV

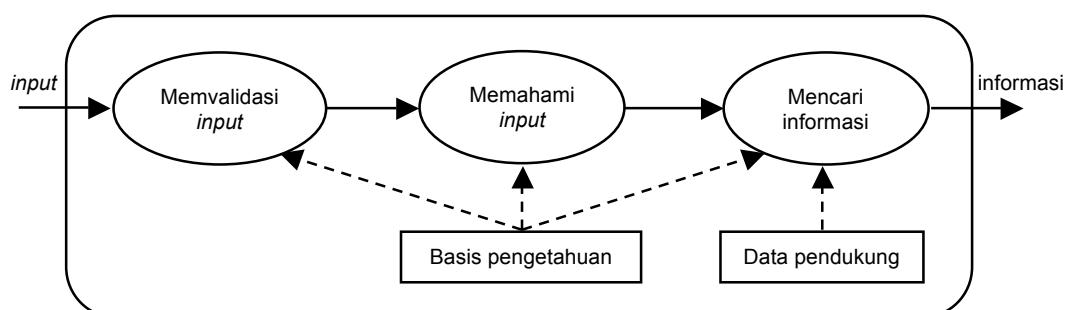
ANALISIS DAN RANCANGAN SISTEM

4.1 Gambaran Umum Sistem

Penelitian ini membahas tentang pengembangan sistem pencarian informasi berbasis teknologi semantik untuk domain jalur pendakian gunung. Sistem menerima *input* berupa *query* bahasa alami yaitu bahasa Indonesia. Sistem memproses *input* untuk dipahami dan menggunakan hasil pemahaman untuk melakukan pencarian informasi. Hasil pencarian informasi kemudian disajikan kepada pengguna.

Secara umum, proses yang dikerjakan sistem hingga menghasilkan informasi dibagi menjadi tiga tahapan, yaitu: memvalidasi *input*, memahami *input* dan mencari informasi. Setiap tahapan proses yang dikerjakan sistem menggunakan basis pengetahuan yang direpresentasikan ke dalam 2 ontologi, yaitu ontologi Bahasa untuk merepresentasikan pengetahuan dibidang linguistik dan ontologi *Mountaineering* untuk merepresentasikan pengetahuan jalur pendakian gunung.

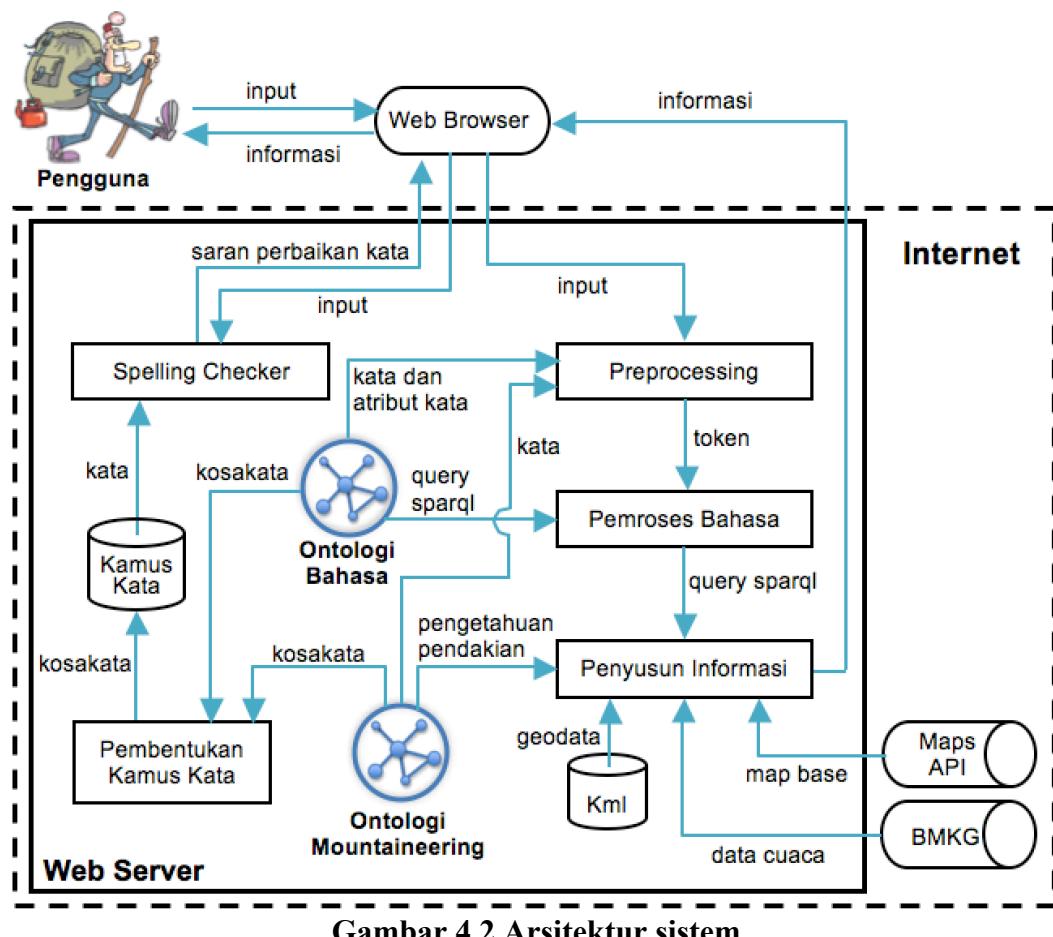
Data pendukung digunakan pada tahap pencarian informasi untuk menyajikan informasi akhir kepada pengguna. Data pendukung yang digunakan yaitu data geografis, data peta dan data cuaca. Tahapan-tahapan yang dilakukan oleh sistem hingga menghasilkan informasi diilustrasikan pada Gambar 4.1.



Gambar 4.1 Ilustrasi tahapan kerja sistem

4.2 Desain Sistem

Sistem yang dikembangkan pada penelitian ini merupakan sistem berbasis *web*. Pengguna dapat melakukan pencarian informasi jalur pendakian gunung menggunakan *web browser* dengan *input* berupa bahasa alami yaitu bahasa Indonesia. Untuk memproses *input* hingga menghasilkan informasi, sistem terbagi menjadi tiga komponen utama yaitu: *Preprocessing*, Pemroses Bahasa dan Penyusun Informasi. Komponen-komponen sistem dapat dilihat pada rancangan arsitektur yang dijabarkan pada Gambar 4.2.

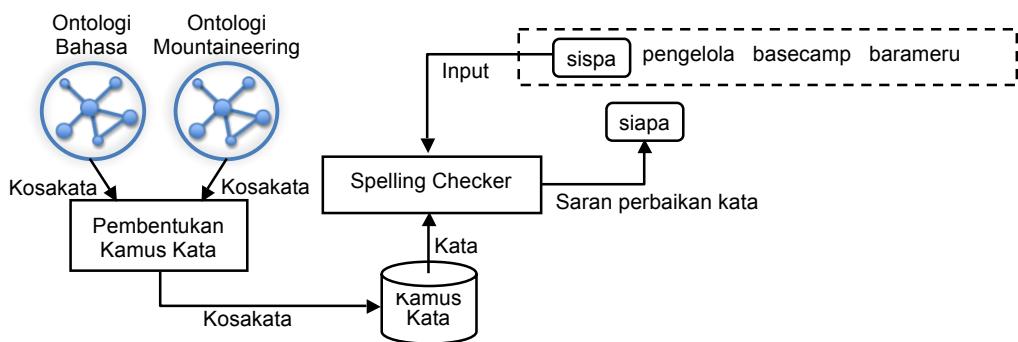


Gambar 4.2 Arsitektur sistem

4.2.1 Penerimaan *input*

Alur kerja sistem dimulai ketika pengguna memasukkan *input* melalui *web browser*. Selama proses *input* berlangsung komponen *Spelling Checker* bertugas mendeteksi dan melakukan pengecekan pada input, proses yang dilakukan oleh *Spelling Checker* yaitu membandingkan susunan abjad pada kata input dengan

susunan abjad pada kata yang ada di dalam kamus kata. Penggunaan *Spelling Checker* bertujuan untuk mengurangi terjadinya kesalahan pengetikan oleh pengguna. Apabila terdapat kata yang ejaannya berbeda maka komponen *Spelling Checker* akan mengirimkan *sugestion* (saran perbaikan) kata. Komponen *Spelling Checker* pada penelitian ini merupakan fitur tambahan, oleh karena itu proses di dalam komponen *Spelling Checker* tidak dibahas lebih mendalam. Ilustrasi proses *Spelling Checker* disajikan pada Gambar 4.3.

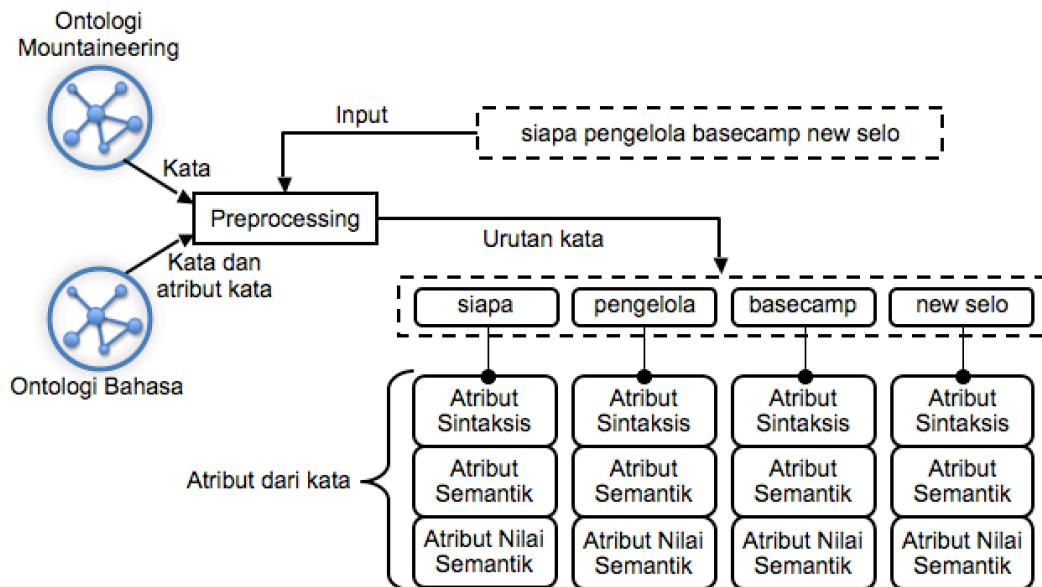


Gambar 4.3 Ilustrasi proses *Spelling Checker*

4.2.2 *Preprocessing*

Input diterima komponen *Preprocessing* ketika *input* telah di-submit oleh pengguna. *Preprocessing* bertugas melakukan validasi dan membentuk *input* menjadi urutan kata. Proses validasi dilakukan dengan mengecek setiap kata input dengan kata yang terdapat di dalam ontologi (ontologi Bahasa dan ontologi *Mountaineering*). *Input* dinyatakan valid apabila setiap kata pada *input* dapat dikenali (ada di dalam ontologi). Setelah *input* dinyatakan valid, *input* akan dibentuk menjadi urutan kata. Ilustrasi dari komponen *Preprocessing* disajikan pada Gambar 4.4.

Satuan kata pada urutan kata dapat mengandung satu kata atau lebih. Misalnya pada kata "new" dan "selo" seperti yang diilustrasikan pada Gambar 4.4, berdasarkan pengetahuan pada ontologi *Mountaineering*, kata "new selo" merupakan nama untuk satu buah objek pendakian dan tidak dapat dipisahkan. Oleh karena itu, sistem membakukan kata *input* "new" dan "selo" menjadi "new selo".



Gambar 4.4 Ilustrasi *Preprocessing*

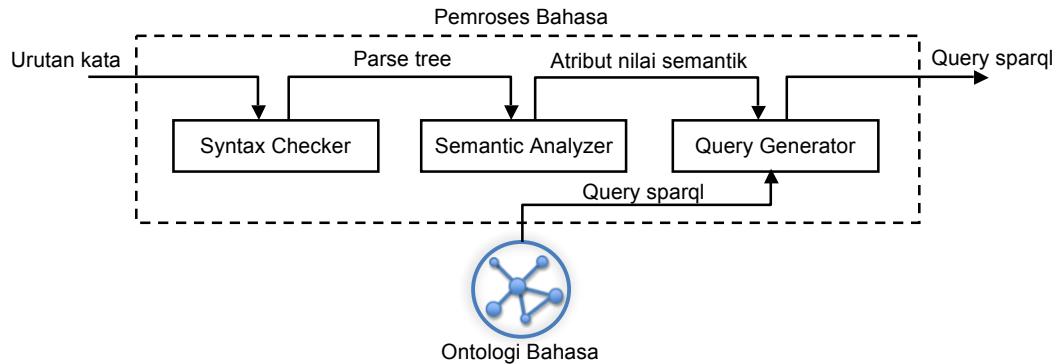
Proses pembentukan *input* menjadi urutan kata juga melibatkan pengambilan atribut tiap kata dari ontologi (ontologi Bahasa). Atribut tersebut yaitu Atribut Sintaksis yang digunakan untuk proses pengecekan sintaksis, Atribut Semantik yang digunakan untuk pengecekan semantik dan Atribut Nilai Semantik yang digunakan untuk menghasilkan *query* SPARQL. Selengkapnya tentang Atribut Sintaksis, Atribut Semantik dan Atribut Nilai Semantik dijabarkan pada Sub Bab 4.3.1.

4.2.3 Pemroses Bahasa

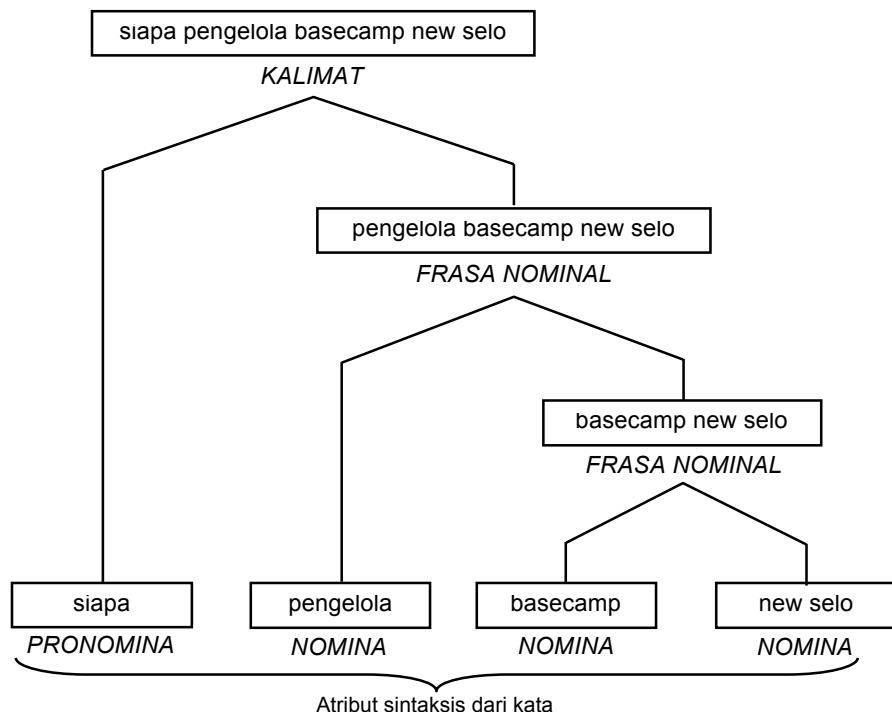
Tugas dari komponen Pemroses Bahasa yaitu memproses urutan kata hasil *Preprocessing* untuk dipahami. Proses pemahaman didasarkan pada pengecekan urutan kata secara sintaksis dan semantik menggunakan aturan tata bahasa Indonesia. Pemroses Bahasa terdiri dari beberapa sub-komponen yaitu *Syntax Checker*, *Semantic Analyzer* dan *Query Generator*. Ilustrasi dari komponen Pemroses Bahasa dapat dilihat pada Gambar 4.5.

Urutan kata yang dihasilkan oleh komponen *Preprocessing* diterima oleh *Syntax Checker*. *Syntax Checker* bertugas melakukan pengecekan sintaksis dengan cara mem-parsing urutan kata ke dalam *parse tree*. Proses pembentukan *parse tree* dilakukan dengan mencocokan Atribut Sintaksis dari tiap kata

menggunakan aturan-aturan tata bahasa Indonesia. Ilustrasi pembentukan *parse tree* dapat dilihat pada Gambar 4.6.



Gambar 4.5 Ilustrasi komponen Pemroses Bahasa

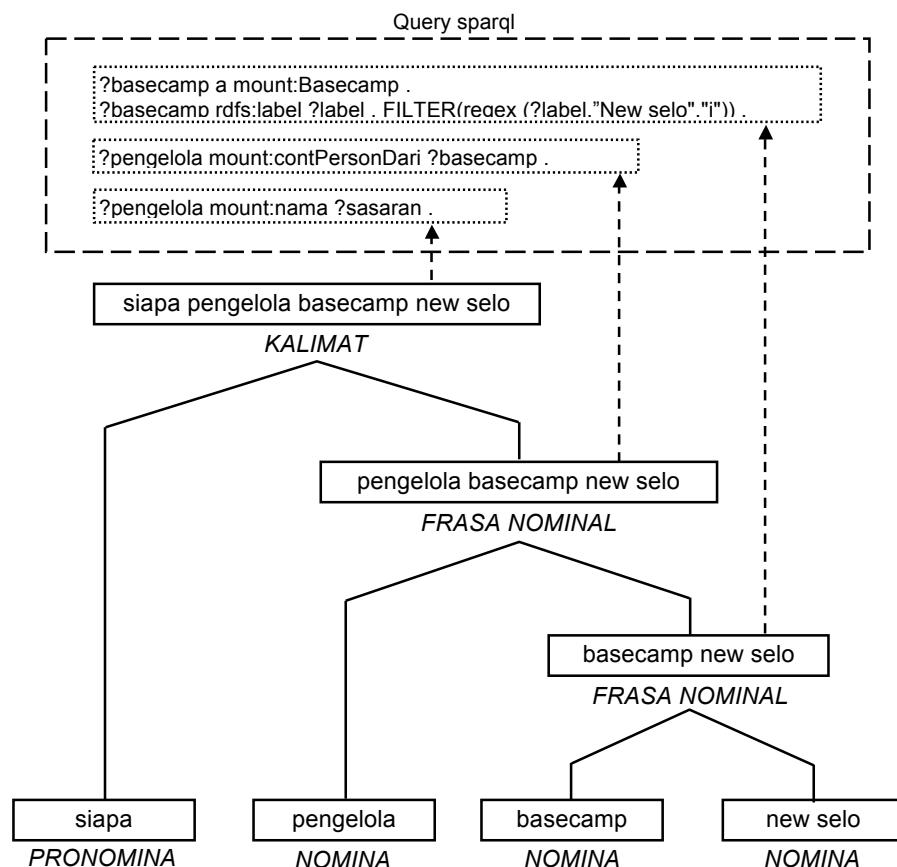


Gambar 4.6 Ilustrasi pembentukan *parse tree*

Apabila urutan kata dapat diterima sebagai penyusun *parse tree* maka proses akan dilanjutkan oleh *Semantic Analyzer*. *Semantic Analyzer* bertugas melakukan pengecekan semantik dengan cara menelusuri *node-node* pembentuk *parse tree* kemudian mencocokkan Atribut Semantik dari tiap *node* menggunakan aturan-aturan tata bahasa. Apabila *node* pembentuk *parse tree*

dapat diterima secara semantik, maka *Semantic Analyzer* akan mengirimkan Atribut Nilai Semantik kepada *Query Generator*.

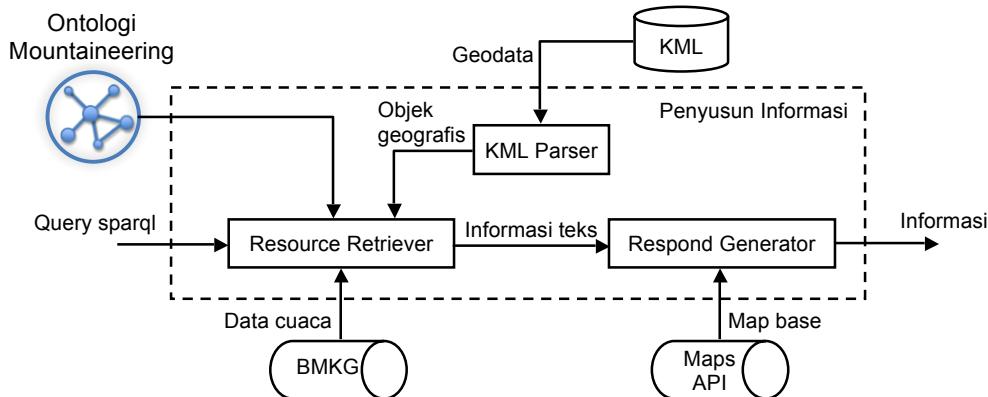
Query Generator bertugas mengubah Atribut Nilai Semantik yang dihasilkan oleh *Semantic Analyzer* menjadi *statement-statement* dalam bentuk *query SPARQL*. *Statement-statement query* SPARQL yang dihasilkan oleh *Query Generator* merupakan hasil akhir dari komponen Pemroses Bahasa yang kemudian dikirimkan kepada komponen Penyusun Informasi. Ilustrasi pembentukan *statement-statement query* SPARQL dijabarkan dalam Gambar 4.7.



Gambar 4.7 Ilustrasi pembentukan *query SPARQL*

4.2.4 Penyusun Informasi

Komponen Penyusun Informasi bertugas mencari, menyusun dan menyajikan informasi. Penyusun Informasi terdiri dari beberapa sub-komponen yaitu *Resource Retriever*, *KML Parser* dan *Respond Generator*. Ilustrasi komponen Penyusun Informasi disajikan pada Gambar 4.8.



Gambar 4.8 Ilustrasi komponen Penyusun Informasi

Resource Retriever bertugas melakukan proses pencarian informasi dengan mengeksekusi *query* SPARQL yang dihasilkan oleh Pemroses Bahasa. *Resource Retriever* juga bertugas mengumpulkan informasi pendukung seperti informasi cuaca yang diambil dari *web service* milik BMKG dan informasi geografis yang diambil dari berkas KML. Proses pencarian informasi geografis melibatkan komponen *KML Parser* yang bertugas mem-*parsing* geodata dari dalam berkas KML menjadi objek-objek geografis.

Hasil akhir dari *Resource Retriever* berupa informasi teks yang diterima oleh *Respond Generator*. *Respond Generator* bertugas memanggil peta dasar menggunakan layanan Google Maps API kemudian menyajikan informasi kepada pengguna. *Respond Generator* juga bertugas memberikan informasi berupa pesan kesalahan jika *input* pengguna tidak dapat diproses oleh sistem.

4.3 Perancangan Ontologi

Sistem yang dikembangkan pada penelitian ini menggunakan dua buah ontologi untuk merepresentasikan pengetahuan dunia (*world knowledge*), yaitu: pengetahuan bahasa yang direpresentasikan ke dalam ontologi Bahasa dan pengetahuan jalur pendakian gunung yang direpresentasikan ke dalam ontologi *Mountaineering*.

4.3.1 Rancangan ontologi Bahasa

Pengetahuan bahasa yang digunakan dalam penelitian ini berkaitan dengan pengetahuan bidang linguistik yakni meliputi pengetahuan tentang kata, hubungan kata, kategori kata, fungsi sintaksis, dan perilaku semantik dari satuan bahasa.

Pengetahuan sintaksis dan pengetahuan semantik digunakan pada proses pemahaman *input* hingga menghasilkan *query* SPARQL didasarkan pada tata bahasa Indonesia baku yang dikemukakan oleh Alwi dkk., (2003). Pengetahuan sintaksis dan pengetahuan semantik dinyatakan dalam aturan-aturan gramatikal yang direpresentasikan menggunakan *Unification Based Grammar* berdasarkan formalisme penulisan gramatikal yang telah dikemukakan oleh (Suryawan, 2013; Gazdar dan Mellish, 1989). Aturan gramatikal yang digunakan pada penelitian ini dapat dilihat pada Lampiran B.

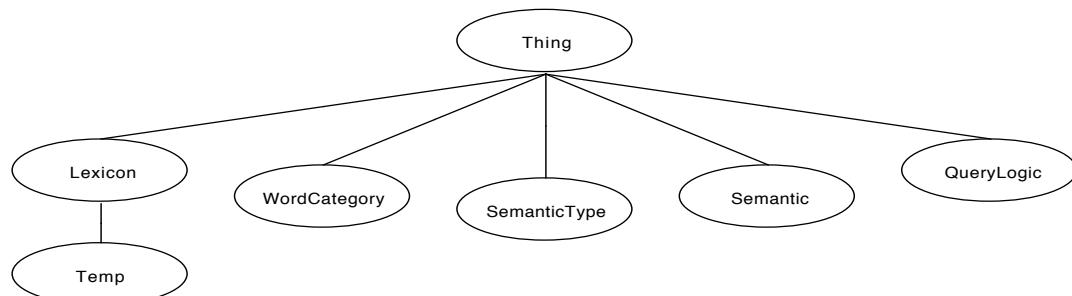
Penggunaan *Unification Based Grammar* untuk merepresentasikan aturan gramatikal memungkinkan sistem untuk melakukan pemeriksaan satuan bahasa yang tidak dapat diterima secara semantik. Misalkan pada kalimat “*siapakah merapi*”, terdiri dari kata “*siapa*” pronomina dan kata “*merapi*” nomina yang secara sintaksis kedua kata tersebut dapat diterima sebagai penyusun kalimat. Berdasarkan pengetahuan dunia kata “*siapa*” merupakan kata untuk menanyakan nomina orang atau insan dan menurut pengetahuan dunia yang dimiliki sistem kata “*merapi*” bukanlah nomina orang atau insan melainkan nomina benda, sehingga kalimat “*siapa merapi*” tidak dapat diterima secara semantik.

Orang menggunakan kalimat berdasarkan pengetahuan dunia yang diketahuinya (Alwi, 2003). Meskipun ontologi Bahasa yang dikembangkan pada penelitian ini dirancang agar dapat *reusable* (dapat digunakan dengan domain selain pendakian gunung). Namun, penggunaan kosakata pada penelitian ini dititikberatkan pada kebutuhan domain penelitian yaitu pendakian gunung. Misalkan penggunaan kata “*basecamp*” pada kalimat “*siapa pengelola basecamp New Selo*”. Kata “*basecamp*” bukanlah kata bahasa Indonesia dan jika diartikan dapat menghasilkan beberapa persepsi, yaitu “tempat kumpul” dan “markas”. Jika kedua persepsi tersebut digunakan sebagai pengganti kata “*basecamp*” maka akan menghasilkan makna yang berbeda, contohnya pada

kalimat “siapa pengelola tempat kumpul New Selo” atau “siapa pengelola markas New Selo”.

Pengetahuan pada ontologi Bahasa dideskripsikan ke dalam konsep-konsep yang disusun berdasarkan suatu klasifikasi dan dikelompokkan ke dalam *class-class* yang sama. Konsep yang digunakan pada ontologi Bahasa didasarkan pada konsep bahasa yang telah dideskripsikan oleh Suryawan (2013), yang dikembangkan sesuai dengan kebutuhan sistem pada penelitian ini.

Class yang dibentuk dalam ontologi Bahasa yaitu *Lexicon*, *WordCategory*, *Semantic*, *SemanticType*, *QueryLogic*. Setiap *class* berada dalam domain yang sama, namun menyimpan konsep pengetahuan yang berbeda-beda. Hierarki antar *class* dideskripsikan pada Gambar 4.9, sedangkan relasi antar kelas dideskripsikan pada Tabel 4.1.



Gambar 4.9 Class-class pada ontologi Bahasa

Tabel 4.1 Relasi antar class pada ontologi Bahasa

No	Class	Relasi
1	<i>Lexicon</i>	<i>Disjoint</i> dengan <i>WordCategory</i> , <i>SemanticType</i> , <i>Semantic</i> dan <i>QueryLogic</i>
2	<i>WordCategory</i>	<i>Disjoint</i> dengan <i>Lexicon</i> , <i>SemanticType</i> , <i>Semantic</i> dan <i>QueryLogic</i>
3	<i>SemanticType</i>	<i>Disjoint</i> dengan <i>WordCategory</i> , <i>Lexicon</i> , <i>Semantic</i> dan <i>QueryLogic</i>
4	<i>Semantic</i>	<i>Disjoint</i> dengan <i>WordCategory</i> , <i>Lexicon</i> , <i>SemanticType</i> dan <i>QueryLogic</i>
5	<i>QueryLogic</i>	<i>Disjoint</i> dengan <i>WordCategory</i> , <i>Lexicon</i> , <i>SemanticType</i> dan <i>Semantic</i>

Class Lexicon adalah representasi dari konsep kata yang merupakan satuan bahasa terkecil. Setiap individual yang terdapat di dalam *class Lexicon* memiliki kategori, peran dan fungsi di dalam kalimat yang dideskripsikan menggunakan

property, yaitu: *word*, *category*, *semanticValue*, *type*, *argCat0*, *argCat1*, *argCat2*, *stripCat*, *argType0*, *argType1*, *argType2*, *stripType*. *Property* yang digunakan untuk mendeskripsikan kata dijabarkan pada Tabel 4.2.

Tabel 4.2 *Property* untuk mendeskripsikan konsep kata

No	Property	Range	Keterangan
1	<i>word</i>	<i>Xsd:String</i>	Deskripsi struktur lahir kata
2	<i>category</i>	<i>WordCategory</i>	Deskripsi kategori sintaksis kata
3	<i>argCat0</i>	<i>WordCategory</i>	
4	<i>argCat1</i>	<i>WordCategory</i>	
5	<i>argCat2</i>	<i>WordCategory</i>	
6	<i>stripCat</i>	<i>WordCategory</i>	
7	<i>semanticValue</i>	<i>Semantic</i>	Deskripsi nilai semantik
8	<i>type</i>	<i>SemanticType</i>	Deskripsi tipe semantik
9	<i>argType0</i>	<i>SemanticType</i>	
10	<i>argType1</i>	<i>SemanticType</i>	
11	<i>argType2</i>	<i>SemanticType</i>	
12	<i>stripType</i>	<i>SemanticType</i>	

Property yang dijabarkan pada Tabel 4.2 memiliki fungsi, yaitu: *Property word* digunakan untuk mendeskripsikan struktur lahir suatu kata. *Property category* digunakan untuk mendeskripsikan kategori sintaksis yang dimiliki kata. *Property type* digunakan untuk mendeskripsikan tipe semantik yang dimiliki kata. *Property semanticValue* digunakan untuk mendeskripsikan nilai semantik kata.

Property argCat0, *argCat1*, *argCat2* dan *stripCat* digunakan untuk mendeskripsikan hubungan perilaku sintaksis suatu kata, frasa, klausa, kalimat. *Property argType0*, *argType1*, *argType2* dan *stripType* digunakan untuk mendeskripsikan hubungan perilaku semantik suatu kata, frasa, klausa, kalimat. Selengkapnya tentang konsep kata dapat dilihat pada Sub Bab 5.1. Selengkapnya tentang implementasi dari perilaku sintaksis dan perilaku semantik menggunakan aturan gramatisal dapat dilihat pada Sub Sub Bab 5.3.3.

Class SemanticType merupakan realisasi dari konsep tentang perilaku semantik yang dimiliki oleh suatu kata. Konsep tentang perilaku semantik tidak dideskripsikan dengan menggunakan *property* apapun.

Class Semantic merupakan realisasi dari konsep tentang nilai semantik dari suatu kata atau suatu satuan bahasa. Konsep tentang nilai semantik dari suatu

kata atau suatu satuan bahasa tidak dideskripsikan dengan menggunakan *property* apapun.

Class WordCategory merupakan realisasi dari konsep tentang kategori sintaksis. Konsep tentang kategori sintaksis tidak dideskripsikan dengan menggunakan *property* apapun.

Class QueryLogic merupakan realisasi dari konsep informasi semantik dari suatu satuan bahasa. Konsep tentang makna dari suatu satuan bahasa dideskripsikan dengan menggunakan dua buah *property*, yaitu: *parA* dan *qpart*. *Property* yang digunakan untuk mendeskripsikan konsep tentang informasi semantik atau makna dari suatu satuan bahasa dijabarkan dalam Tabel 4.3.

Tabel 4.3 Property untuk mendeskripsikan informasi semantik

No	Property	Range	Keterangan
1	<i>parA</i>	<i>Sematik</i>	Deskripsi nilai semantik
2	<i>qpart</i>	<i>Xsd:String</i>	Deskripsi informasi semantik dalam bentuk <i>statement query</i> SPARQL

Property parA yang dijabarkan pada Tabel 4.3 digunakan untuk menyatakan asosiasi dari penyusun satuan bahasa yang dideskripsikan dalam nilai semantik suatu kata. *Property qpart* digunakan untuk menyatakan informasi semantik atau makna dari suatu satuan bahasa dalam bentuk *statement query* SPARQL.

4.3.2 Rancangan ontologi *Mountaineering*

Ontologi *Mountaineering* yang dikembangkan pada penelitian ini merupakan representasi dari pengetahuan jalur pendakian dan gunung yang berhubungan dengan olahraga *mountaineering*. Penelitian ini mengasumsikan ontologi yang merepresentasikan pengetahuan jalur pendakian dan gunung yang berhubungan dengan olahraga *mountaineering* belum pernah didefinisikan sebelumnya, sehingga ontologi *Mountaineering* dikembangkan dari awal mengikuti tahapan *ontologi development* yang telah dijabarkan oleh Noy dan Macguines (2001).

Pengetahuan jalur pendakian gunung mencakup pengetahuan tentang gunung, jalur pendakian dan titik penting yang ada di jalur pendakian. Konsep pengetahuan jalur pendakian gunung dideskripsikan ke dalam enam *class* utama, yaitu: *Mountain*,

Crater, *ClimbingTracks*, *ClimbingPoints*, *AdministrativeRegions* dan *Person*. Hubungan antar *class* pada ontologi *Mountaineering* dijabarkan pada Tabel 4.4.

Class Mountain merupakan penjabaran dari konsep gunung, *class Mountain* tidak memiliki *subclass*. *Class Mountain* dideskripsikan dengan *property*, yaitu: *locatedIn*, *locationOf*, *hasCaretaker*, *hasPeak*, *hasCrater*, *hasTrack*, *label*, *mapFile*, *pmId*, *description*, *name* dan *elevation*. *Property* yang digunakan untuk mendeskripsikan konsep gunung yang dijabarkan pada Tabel 4.5.

Tabel 4.4 Hubungan antar *class* utama pada ontologi *Mountaineering*

No	Class	Keterangan
1	<i>Mountain</i>	<i>Disjoint</i> dengan <i>Class AdministrativeRegions</i> , <i>ClimbingTracks</i> , <i>ClimbingPoints</i> , <i>Crater</i> dan <i>Person</i>
2	<i>Crater</i>	<i>Disjoint</i> dengan <i>Class Mountain</i> , <i>AdministrativeRegions</i> , <i>ClimbingTracks</i> , <i>ClimbingPoints</i> dan <i>Person</i>
3	<i>ClimbingTracks</i>	<i>Disjoint</i> dengan <i>Class Mountain</i> , <i>AdministrativeRegions</i> , <i>ClimbingPoints</i> , <i>Crater</i> dan <i>Person</i>
4	<i>ClimbingPoints</i>	<i>Disjoint</i> dengan <i>Class Mountain</i> , <i>AdministrativeRegions</i> , <i>ClimbingTracks</i> , <i>Crater</i> dan <i>Person</i>
5	<i>AdministrativeRegions</i>	<i>Disjoint</i> dengan <i>Class Mountain</i> , <i>ClimbingTracks</i> , <i>ClimbingPoints</i> , <i>Crater</i> dan <i>Person</i>
6	<i>Person</i>	<i>Disjoint</i> dengan <i>Class Mountain</i> , <i>AdministrativeRegions</i> , <i>ClimbingTracks</i> , <i>ClimbingPoints</i> dan <i>Crater</i>

Tabel 4.5 Property untuk mendeskripsikan konsep gunung

No	Property	Range	Keterangan
1	<i>locatedIn</i>	<i>AdministrativeRegions</i>	Lokasi gunung
2	<i>locationOf</i>	<i>Mountain</i> , <i>ClimbingTracks</i> , <i>ClimbingPoints</i> , <i>Crater</i>	Lokasi objek pendakian yang terletak di gunung
3	<i>hasCaretaker</i>	<i>Person</i>	Juru kunci gunung
4	<i>hasPeak</i>	<i>Peak</i>	Puncak gunung
5	<i>hasCrater</i>	<i>Crater</i>	Kawah yang terletak di gunung
6	<i>hasTrack</i>	<i>ClimbingTracks</i>	Jalur pendakian yang terletak di gunung
7	<i>label</i>	<i>Xsd:String</i>	Label
8	<i>mapFile</i>	<i>Xsd:String</i>	Berkas KML
9	<i>pmId</i>	<i>Xsd:String</i>	Identitas unik gunung pada berkas KML
10	<i>description</i>	<i>Xsd:String</i>	Penjelasan tentang gunung
11	<i>name</i>	<i>Xsd:String</i>	Nama gunung
12	<i>elevation</i>	<i>Xsd:Integer</i>	Ketinggian gunung

Class Crater merupakan penjabaran dari konsep kawah yang terdapat di suatu gunung. *Class Crater* dideskripsikan dengan *property*, yaitu : *craterOf*, *locatedIn*, *description*, *label*, *mapFile*, *pmId*. *Property* yang digunakan untuk mendeskripsikan konsep kawah dijabarkan pada Tabel 4.6.

Tabel 4.6 Property untuk mendeskripsikan konsep kawah

No	Property	Range	Keterangan
1	<i>craterOf</i>	Mountain	Kawah dari suatu gunung
2	<i>locatedIn</i>	<i>ClimbingTracks</i> , Mountain	Lokasi kawah
3	<i>description</i>	Xsd:String	Tentang kawah
4	<i>label</i>	Xsd:String	Label
5	<i>mapFile</i>	Xsd:String	Berkas KML
6	<i>name</i>	Xsd:String	Nama kawah
7	<i>pmId</i>	Xsd:String	Identitas unik kawah pada berkas KML

Class ClimbingTracks merupakan penjabaran dari konsep jalur pendakian yang terdapat di gunung. Satu gunung dapat memiliki lebih dari satu jalur pendakian. *Class ClimbingTracks* tidak memiliki relasi *subclass*. *Class ClimbingTracks* dideskripsikan dengan menggunakan *property*, yaitu: *hasCriticalPoint*, *locationOf*, *locatedIn*, *hasPost*, *hasBasecamp*, *hasWaterSources*, *hasCampground*, *hasShortCut*, *name*, *label*, *status*, *mapFile*, *pmId*, *description*, *distance*. *Property* yang digunakan untuk mendeskripsikan *class ClimbingTracks* dijabarkan pada Tabel 4.7.

Class ClimbingPoints merupakan penjabaran dari konsep titik-titik penting yang ada pada jalur pendakian. *Class ClimbingPoints* memiliki relasi *subclass* yaitu: *Basecamp*, *Post*, *Campground*, *WaterSources*, *Peak* dan *UniquePoint*. Relasi *subclass* yang terdapat pada *class ClimbingPoints* dijabarkan pada Tabel 4.8.

Pembentukan *subclass* pada *class ClimbingPoints* dimaksudkan untuk mendeskripsikan konsep dari titik-titik yang ada di jalur pendakian secara lebih spesifik. Hal itu dikarenakan setiap titik pendakian yang terdapat pada suatu jalur pendakian gunung dapat memiliki makna yang berbeda-beda, contohnya *class Basecamp* dan *class Post*, *Basecamp* biasanya merupakan titik awal dari suatu pendakian dan *Post* merupakan titik atau poin yang terdapat di jalur pendakian

dan biasanya terdapat area dimana pendaki dapat beristirahat dan mendirikan tenda.

Tabel 4.7 Property untuk mendeskripsikan konsep jalur pendakian

No	Property	Range	Keterangan
1	<i>hasCriticalPoint</i>	<i>ClimbingPoints</i>	Titik pendakian yang terletak di jalur pendakian
2	<i>locationOf</i>	<i>ClimbingPoints, Crater</i>	Jalur pendakian sebagai lokasi dari titik pendakian
3	<i>locatedIn</i>	<i>Mountain</i>	Lokasi jalur pendakian
4	<i>hasPost</i>	<i>Post</i>	Pos yang terdapat di jalur pendakian
5	<i>hasBasecamp</i>	<i>Basecamp</i>	<i>Basecamp</i> dari jalur pendakian
6	<i>hasWaterSources</i>	<i>WaterSources</i>	Sumber air yang terdapat di jalur pendakian
7	<i>hasCampground</i>	<i>Campground</i>	<i>Campground</i> yang terdapat di jalur pendakian
8	<i>hasShortCut</i>	<i>ClimbingTracks</i>	Jalur pintas yang terdapat pada jalur pendakian
9	<i>name</i>	<i>Xsd:String</i>	Nama jalur
10	<i>label</i>	<i>Xsd:String</i>	Label
11	<i>status</i>	<i>Xsd:String</i>	Status jalur pendakian
12	<i>mapFile</i>	<i>Xsd:String</i>	Lokasi data KML
13	<i>pmlId</i>	<i>Xsd:String</i>	Identitas unik jalur pada data KML
15	<i>description</i>	<i>Xsd:String</i>	Penjelasan tentang jalur
16	<i>distance</i>	<i>Xsd:Integer</i>	Panjang jalur

Tabel 4.8 Subclass dari class *ClimbingPoints*

No	Subclass	Keterangan
1	<i>Basecamp</i>	Subclass dari Class <i>ClimbingPoints</i> yang menjabarkan konsep <i>basecamp</i>
2	<i>Post</i>	Subclass dari Class <i>ClimbingPoints</i> yang menjabarkan konsep pos
3	<i>Campground</i>	Subclass dari Class <i>ClimbingPoints</i> yang menjabarkan konsep tempat mendirikan tenda dan bermalam
4	<i>WaterSources</i>	Subclass dari Class <i>ClimbingPoints</i> yang menjabarkan konsep sumber air yang ada di jalur pendakian
5	<i>Peak</i>	Subclass dari Class <i>ClimbingPoints</i> yang menjabarkan konsep puncak yang dicapai melalui jalur pendakian
6	<i>UniquePoint</i>	Subclass dari Class <i>ClimbingPoints</i> yang menjabarkan konsep titik-titik unik dan tidak dapat dikategorikan

Class Basecamp merupakan penjabaran dari konsep *basecamp* yang terdapat pada jalur pendakian. *Class Basecamp* dideskripsikan dengan *property*, yaitu: *criticalPointOf*, *hasContactPerson*, *locatedIn*, *basecampOf*, *name*, *phone*,

description, elevation, label, mapFile, pmId. *Property* yang digunakan untuk mendeskripsikan konsep *basecamp* dijabarkan pada Tabel 4.9.

Tabel 4.9 *Property* untuk mendeskripsikan konsep *basecamp*

No	Property	Range	Keterangan
1	<i>criticalPointOf</i>	<i>ClimbingTracks</i>	Titik penting dari jalur pendakian
2	<i>hasContactPerson</i>	<i>Person</i>	Pengelola <i>basecamp</i>
3	<i>locatedIn</i>	<i>ClimbingTracks, Mountain, AdministrativeRegions</i>	Lokasi <i>basecamp</i>
4	<i>basecampOf</i>	<i>ClimbingTracks</i>	<i>Basecamp</i> dari jalur pendakian
5	<i>name</i>	<i>Xsd:String</i>	Nama <i>basecamp</i>
6	<i>phone</i>	<i>Xsd:String</i>	Telepon
7	<i>description</i>	<i>Xsd:String</i>	Penjelasan tentang <i>basecamp</i>
8	<i>elevation</i>	<i>Xsd:Integer</i>	Tingkat ketinggian <i>basecamp</i>
9	<i>label</i>	<i>Xsd:String</i>	Label
10	<i>mapFile</i>	<i>Xsd:String</i>	Berkas KML
11	<i>pmId</i>	<i>Xsd:String</i>	Identitas unik <i>basecamp</i> pada berkas KML
12	<i>address</i>	<i>Xsd:String</i>	Alamat <i>basecamp</i>

Class Post merupakan penjabaran dari konsep pos pendakian yang ada di jalur pendakian. *Class Post* dideskripsikan dengan *property*, yaitu: *postOf, locatedIn, criticalPointOf, description, name, mapFile, pmId, label, address, elevation.* *Property* yang digunakan untuk mendeskripsikan konsep pos dijabarkan pada Tabel 4.10.

Tabel 4.10 *Property* yang digunakan untuk mendeskripsikan konsep pos

No	Property	Range	Keterangan
1	<i>postOf</i>	<i>ClimbingTracks</i>	Pos dari jalur pendakian
2	<i>locatedIn</i>	<i>Mountain, ClimbingTracks</i>	Lokasi pos
2	<i>criticalPointOf</i>	<i>ClimbingTracks</i>	Titik penting dari jalur pendakian
3	<i>description</i>	<i>Xsd:String</i>	Penjelasan tentang pos
4	<i>name</i>	<i>Xsd:String</i>	Nama pos
5	<i>mapFile</i>	<i>Xsd:String</i>	Berkas KML
6	<i>pmId</i>	<i>Xsd:String</i>	Identitas unik pos pada berkas KML
7	<i>label</i>	<i>Xsd:String</i>	Label
8	<i>elevation</i>	<i>Xsd:Integer</i>	Tingkat ketinggian pos

Class Campground merupakan deskripsi dari konsep tempat mendirikan tenda dan tempat bermalam yang ada di jalur pendakian gunung. *Class Campground* dideskripsikan menggunakan *property*, yaitu: *campgroundOf,*

criticalPointOf, locatedIn, description, name, label, mapFile, pmId, elevation. *Property* yang digunakan untuk menjabarkan konsep *campground* dijabarkan pada Tabel 4.11.

Class WaterSources merupakan deskripsi dari konsep sumber air atau lokasi tempat mengambil air yang terdapat pada jalur pendakian. *Class WaterSources* dideskripsikan menggunakan *property*, yaitu: *locatedIn, waterSourcesOf, criticalPointOf, description, name, label, mapFile, pmId.* *Property* yang digunakan untuk mendeskripsikan konsep sumber air dijabarkan pada Tabel 4.12.

Tabel 4.11 *Property* untuk menjabarkan konsep *campground*

No	<i>Property</i>	<i>Range</i>	Keterangan
1	<i>campgroundOf</i>	<i>ClimbingTracks</i>	<i>Campground</i> dari jalur pendakian
2	<i>criticalPointOf</i>	<i>ClimbingTracks</i>	Titik penting dari jalur pendakian
3	<i>locatedIn</i>	<i>Mountain, ClimbingTracks</i>	Lokasi <i>campground</i>
4	<i>description</i>	<i>Xsd:String</i>	Penjelasan <i>campground</i>
5	<i>name</i>	<i>Xsd:String</i>	Nama <i>campground</i>
6	<i>label</i>	<i>Xsd:String</i>	Label
7	<i>mapFile</i>	<i>Xsd:String</i>	Berkas KML
8	<i>pmId</i>	<i>Xsd:String</i>	Identitas unik <i>campground</i> pada berkas KML
9	<i>elevation</i>	<i>Xsd:Integer</i>	Tingkat ketinggian <i>campground</i>

Tabel 4.12 *Property* untuk mendeskripsikan konsep sumber air

No	<i>Property</i>	<i>Range</i>	Keterangan
1	<i>locatedIn</i>	<i>ClimbingTracks</i>	Lokasi sumber air
2	<i>waterSourcesOf</i>	<i>Mountain, ClimbingTracks</i>	Sumber air dari jalur pendakian
3	<i>criticalPointOf</i>	<i>ClimbingTracks</i>	Titik penting dari jalur pendakian
4	<i>description</i>	<i>Xsd:String</i>	Penjelasan tentang sumber air
5	<i>name</i>	<i>Xsd.String</i>	Nama sumber air
6	<i>label</i>	<i>Xsd.String</i>	Label
7	<i>mapFile</i>	<i>Xsd.String</i>	Berkas KML
8	<i>pmId</i>	<i>Xsd.String</i>	Identitas unik sumber air pada berkas KML

Class Peak merupakan penjabaran dari konsep puncak gunung. Puncak gunung merupakan titik tertinggi di suatu gunung sekaligus menjadi tujuan akhir pendaki. Puncak dapat juga sebagai titik yang dilalui pada suatu jalur pendakian untuk menuju puncak yang lain, karena suatu gunung dapat memiliki lebih dari

satu puncak. *Class Peak* dideskripsikan dengan *property*, yaitu: *peakOf*, *criticalPointOf*, *locatedIn*, *description*, *name*, *label*, *mapFile*, *pmId*, *elevation*. *Property* yang digunakan untuk mendeskripsikan konsep puncak gunung dijabarkan pada Tabel 4.13.

Class UniquePoint menjabarkan konsep titik-titik yang bersifat unik dan tidak dapat dikelompokkan, karena itu *Class UniquePoint* tidak termasuk ke dalam *class Basecamp*, *Post*, *Crater*, *Peak*, *Campground* dan *WaterSources*. *Class UniquePoint* dideskripsikan dengan *property*, yaitu: *criticalPointOf*, *locatedIn*, *description*, *name*, *label*, *mapFile*, *pmId*, *elevation*. *Property* yang digunakan untuk mendeskripsikan titik-titik unik dijabarkan pada Tabel 4.14.

Tabel 4.13 Property yang digunakan untuk mendeskripsikan konsep puncak

No	Property	Range	Keterangan
1	<i>peakOf</i>	<i>Mountain</i>	Puncak dari suatu gunung
2	<i>criticalPointOf</i>	<i>ClimbingTracks</i>	Titik penting dari jalur pendakian
3	<i>locatedIn</i>	<i>Mountain</i> , <i>ClimbingTracks</i>	Lokasi puncak
4	<i>description</i>	<i>Xsd:String</i>	Penjelasan dari puncak
5	<i>name</i>	<i>Xsd:String</i>	Nama puncak
6	<i>label</i>	<i>Xsd:String</i>	Label
7	<i>mapFile</i>	<i>Xsd:String</i>	Berkas KML
8	<i>pmId</i>	<i>Xsd:String</i>	Identitas unik puncak pada berkas KML
9	<i>elevation</i>	<i>Xsd:Integer</i>	Tingkat ketinggian puncak

Tabel 4.14 Property untuk mendeskripsikan konsep titik unik

No	Property	Range	Keterangan
1	<i>criticalPointOf</i>	<i>ClimbingTracks</i>	Titik unik dari jalur pendakian
2	<i>locatedIn</i>	<i>Mountain</i> , <i>ClimbingTracks</i>	Lokasi titik unik
3	<i>description</i>	<i>Xsd:String</i>	Penjelasan tentang titik unik
4	<i>name</i>	<i>Xsd:String</i>	Nama titik unik
5	<i>label</i>	<i>Xsd:String</i>	Label
6	<i>mapFile</i>	<i>Xsd:String</i>	Berkas KML
7	<i>pmId</i>	<i>Xsd:String</i>	Identitas unik titik pada berkas KML
8	<i>elevation</i>	<i>Xsd:Integer</i>	Tingkat ketinggian titik unik

Class AdministrativeRegions merupakan penjabaran dari konsep wilayah administratif yang terdapat di Indonesia. Deskripsi kewilayahan yang dijabarkan pada penelitian ini merupakan wilayah administratif yang berhubungan dengan lokasi gunung, jalur pendakian dan titik pendakian. *Class AdministrativeRegions*

memiliki relasi *subclass*, yaitu: *Kabupaten* dan *Province*. Relasi *subclass* dari *class AdministrativeRegions* dijabarkan pada Tabel 4.15.

Class Kabupaten merupakan penjabaran dari konsep wilayah administratif kabupaten yang dideskripsikan dengan menggunakan *property*, yaitu: *locationOf*, *locatedIn*, *label* dan *name*. *Property* yang digunakan untuk mendeskripsikan konsep tentang kabupaten dijabarkan pada Tabel 4.16.

Class Province merupakan penjabaran dari konsep wilayah administratif provinsi yang dideskripsikan dengan menggunakan *property*, yaitu: *locationOf*, *name*, *label* dan *weatherTomorrow*. *Property* yang digunakan untuk mendeskripsikan konsep tentang provinsi dijabarkan pada Tabel 4.17.

Tabel 4.15 Subclass dari class *AdministrativeRegions*

No	Subclass	Keterangan
1	<i>Kabupaten</i>	<i>Subclass</i> dari <i>Class AdministrativeRegions</i> yang menjabarkan konsep dari kabupaten
2	<i>Province</i>	<i>Subclass</i> dari <i>Class AdministrativeRegions</i> yang menjabarkan konsep dari provinsi

Tabel 4.16 Property untuk mendeskripsikan konsep kabupaten

No	Property	Range	Keterangan
1	<i>locationOf</i>	<i>Mountain</i> , <i>ClimbingTracks</i> , <i>ClimbingPoints</i>	Lokasi dari gunung, jalur dan titik pendakian
2	<i>locatedIn</i>	<i>Province</i>	Lokasi kabupaten
3	<i>label</i>	<i>Xsd:String</i>	Label
4	<i>name</i>	<i>Xsd:String</i>	Nama kabupaten

Tabel 4.17 Property untuk mendeskripsikan konsep provinsi

No	Property	Range	Keterangan
1	<i>locationOf</i>	<i>Mountain</i> , <i>ClimbingTracks</i> , <i>ClimbingPoints</i>	Lokasi dari gunung, jalur dan titik pendakian
2	<i>name</i>	<i>Xsd:String</i>	Nama provinsi
3	<i>label</i>	<i>Xsd:String</i>	Label
4	<i>weatherTomorrow</i>	<i>Xsd:String</i>	Data cuaca

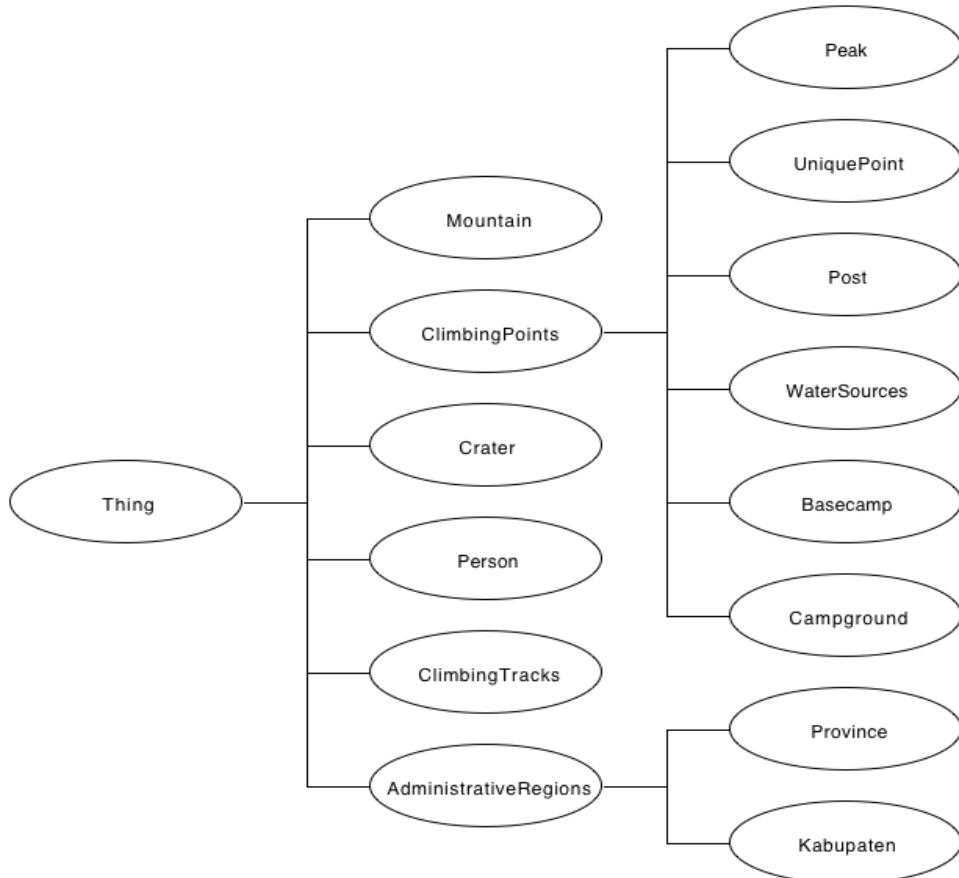
Class Person merupakan deskripsi dari konsep tentang orang yang berhubungan dengan gunung dan jalur pendakian, misalnya orang yang menjadi juru kunci suatu gunung dan orang yang bertanggung jawab terhadap suatu *basecamp* atau *contact person* yang bisa dihubungi pada suatu *basecamp*. *Class*

Person dideskripsikan menggunakan *property*, yaitu: *caretakerOf*, *contactPersonOf*, *description*, *name*, *phone*, *label*. *Property* yang digunakan untuk mendeskripsikan konsep orang disajikan pada Tabel 4.18.

Tabel 4.18 Property untuk mendeskripsikan konsep orang

No	Property	Range	Keterangan
1	<i>caretakerOf</i>	<i>Mountain</i>	Juru kunci dari gunung
2	<i>contactPersonOf</i>	<i>Basecamp</i>	Kontak person dari <i>basecamp</i>
3	<i>deskripsi</i>	<i>Xsd:String</i>	Nama orang
4	<i>nama</i>	<i>Xsd:String</i>	Nomor telepon
5	<i>label</i>	<i>Xsd:String</i>	Label

Hubungan dari tiap *class* dan *subclass* yang dideskripsikan pada ontologi *Mountaineering* dapat dilihat pada Gambar 4.10.

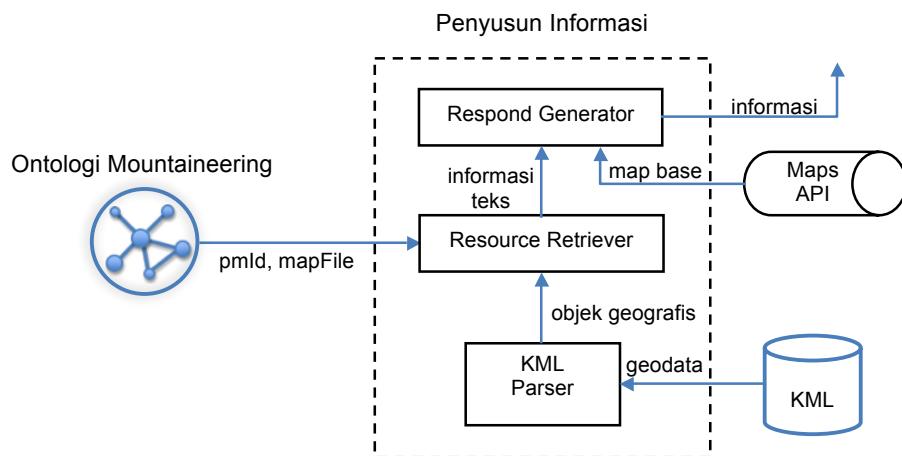


Gambar 4.10 Relasi class yang ada dalam ontologi *Mountaineering*

4.4 Perancangan Data KML

Kelebihan dari pencarian yang dikembangkan pada penelitian ini yaitu fitur untuk dapat melakukan pencarian ke dalam konten informasi geografis berupa peta. Peta dasar (*base map*) diambil dari layanan dari *Google Maps API* dengan konten data-data spasial yang direpresentasikan ke dalam berkas KML.

KML berfungsi menyediakan konten data-data keruangan (spasial) yang dihubungkan dengan ontologi *Mountaineering* menggunakan dua buah *property*, yaitu: *property mapFile* dan *pmId*. *Property mapFile* digunakan untuk menentukan berkas KML yang akan digunakan, sedangkan *property pmId* digunakan untuk memanggil identitas unik dari objek yang terdapat di dalam berkas KML. Ilustrasi pengambilan data dari dalam KML dapat dilihat pada Gambar 4.11.



Gambar 4.11 Ilustrasi pengambilan geodata dari KML

Penelitian ini menggunakan *placemark* dan *polyline* untuk menjabarkan objek-objek yang ada di gunung. *Placemark* digunakan untuk menjabarkan satu buah objek yang mewakili suatu *instance* pendakian yang terdapat di dalam ontologi *Mountaineering*, misalnya: gunung, *basecamp* dan pos. *Placemark* di dalam berkas KML dideskripsikan menggunakan struktur *tag*, yaitu: *<Name>*, *<ExtendedData>*, *<Description>*, *<Styleurl>* dan *<Point>*. Struktur *tag* yang digunakan untuk mendeskripsikan *placemark* selengkapnya dapat dilihat pada Tabel 4.19.

Tabel 4.19 Struktur tag yang digunakan untuk mendeskripsikan placemark

No	Nama tag	Keterangan
1	<i>Name</i>	Nama placemark
2	<i>ExtendedData</i>	Deskripsi identitas unik placemark
3	<i>Description</i>	Penjelasan dari placemark
4	<i>Styleurl</i>	Style dan icon dari placemark
5	<i>Point</i>	Titik koordinat dari placemark

Polyline berfungsi untuk menjabarkan objek dengan koordinat yang saling terhubung, pada penelitian ini *polyline* digunakan untuk menggambarkan jalur pendakian. *Polyline* dideskripsikan dengan struktur tag yaitu: <*Name*>, <*ExtendedData*>, <*Description*>, <*Styleurl*>, <*Linestring*>. Struktur tag untuk mendeskripsikan *polyline* selengkapnya dapat dilihat pada Tabel 4.20.

Tabel 4.20 Struktur tag yang digunakan dalam fitur polyline

No	Nama tag	Keterangan
1	<i>Name</i>	Nama polyline
2	<i>ExtendedData</i>	Deskripsi identitas unik polyline
3	<i>Description</i>	Penjelasan
4	<i>Styleurl</i>	style dan icon dari polyline
5	<i>Linestring</i>	Titik koordinat untuk membentuk polyline

Tag <*ExtendedData*> merupakan tag yang digunakan khusus untuk mendeskripsikan identitas unik dari suatu *instance* pada ontologi *Mountaineering* yang memiliki objek geografis di dalam berkas KML. Penggunaan tag <*ExtendedData*> dijabarkan pada Gambar 4.12.

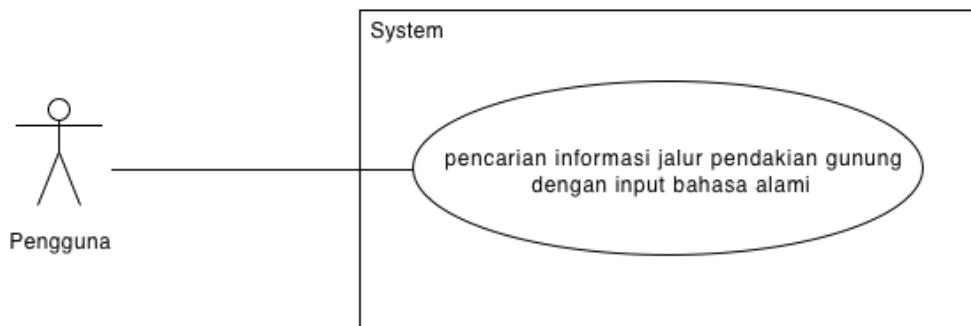
```
<ExtendedData>
  <Data name="pmId">
    <value>id unik suatu instance</value>
  </Data>
</ExtendedData>
```

Gambar 4.12 Penggunaan tag <ExtendedData>

4.5 Perancangan Sistem dengan UML

Perancangan sistem menggunakan UML dilakukan dengan memodelkan struktur dan *behavior* sistem. Struktur dari sistem dideskripsikan dengan menggunakan *usecase diagram*, sedangkan *behavior* sistem dideskripsikan dengan menggunakan *activity diagram*.

Aplikasi hanya menyediakan satu buah layanan yaitu pencarian informasi dan layanan yang disediakan sistem dapat digunakan secara langsung oleh pengguna. Aplikasi hanya mengenal satu jenis pengguna sehingga semua pengguna aplikasi memiliki hak akses yang sama. Layanan yang disediakan oleh aplikasi beserta pengguna layanan dideskripsikan dalam Gambar 4.13.



Gambar 4.13 Usecase diagram

Pemrosesan *request* dikerjakan oleh beberapa komponen, yaitu: *User Interface* (UI), *Spelling Checker*, *Controller*, *Preprocess*, *Parser*, *Semantic Analyser*, dan *Resource Retriever*. Tahapan pemrosesan *request* yang dideskripsikan dengan menggunakan *activity diagram* dijabarkan pada Gambar 4.14.

Pemrosesan dimulai ketika komponen UI menerima *input* dari pengguna. Selama *input* yang diterima UI belum di-*submit* oleh pengguna, UI mengirimkan *input* kepada komponen *Spelling Checker*. Komponen *Spelling Checker* bertugas memeriksa dan mendeteksi apabila pada *input* terdapat kesalahan ejaan. Proses pengecekan dilakukan dengan mencocokkan kata pada *input* dengan kata pada ontologi. Komponen *Spelling Checker* pada penelitian ini merupakan fitur tambahan sehingga tidak akan dibahas lebih lanjut, pada penelitian ini komponen *spelling checker* menggunakan *library Jazzy Spell Checker* yang dikembangkan oleh Idzelis (2005).

Input yang telah di-*submit* diterima oleh *Controller* yang kemudian diteruskan kepada *Preprocessor*. *Preprocessor* bertugas melakukan validasi pada *input*. *Input* yang dinyatakan valid kemudian dibentuk ke dalam urutan-urutan kata. Urutan-urutan kata yang dihasilkan oleh *Preprocessor* dikembalikan kepada

Controller yang kemudian diteruskan kepada *Parser*. *Parser* menerima urutan-urutan kata dan membentuk urutan-urutan kata menjadi *parse tree*. Proses pembentukan *parse tree* dilakukan dengan melakukan pengecekan sintaksis berdasarkan aturan-aturan gramatikal tata bahasa Indonesia.

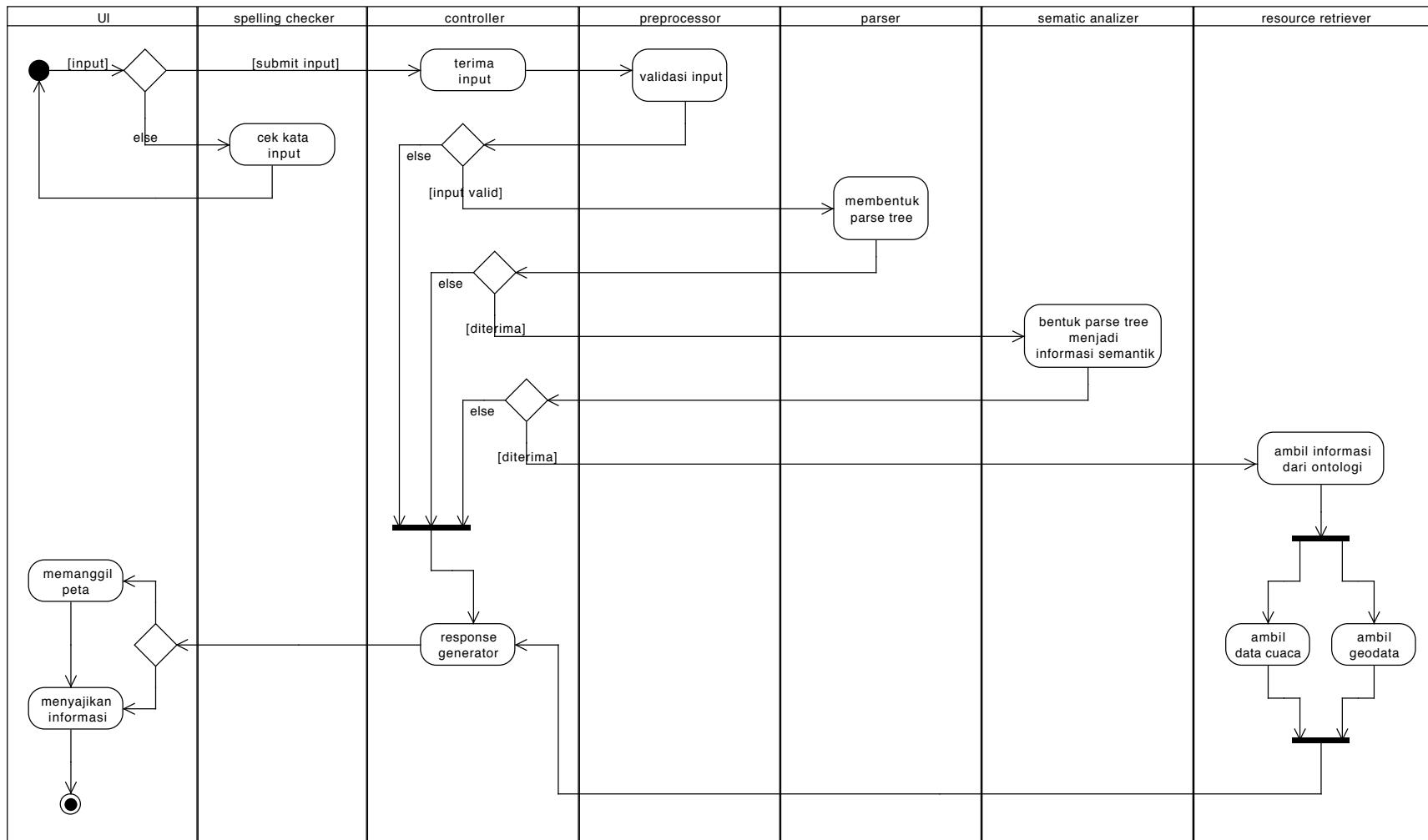
Apabila *parse tree* dapat diterima secara sintaksis sebagai representasi satuan bahasa, parser mengembalikan hasil pengecekan kepada *Controller*. Apabila urutan kata tidak dapat diterima secara aturan sintaksis maka *Controller* akan membuat pesan kesalahan yang dikirimkan kepada *Response Generator*.

Urutan kata yang dapat diterima berdasarkan aturan sintaksis diteruskan kepada *Semantic Analyzer*. *Semantic Analyzer* bertugas melakukan pengecekan semantik dari tiap *node* penyusun *Parse Tree*, *node* yang dapat diterima secara semantik kemudian dibentuk menjadi informasi semantik berupa potongan-potongan *query SPARQL*.

Controller menerima hasil dari *Semantic Analyzer* kemudian dikirimkan kepada *Resource Retriever*. *Resource Retriever* mengeksekusi *query* untuk mencari informasi ke dalam ontologi. Jika dibutuhkan *Resource Retriever* juga mengambil informasi dari luar ontologi yaitu *geodata* dari berkas KML dan data cuaca dari *web service BMKG*.

Resource Retriever mengirimkan hasil pencarian kepada *Response Generator*. *Response Generator* bertugas untuk membentuk informasi baik itu informasi yang merupakan hasil pencarian dari *Resource Retriever* ataupun pesan-pesan kesalahan dari *Preprocessor*, *Parser* dan *Semantic Analyzer*.

UI bertugas menyajikan informasi yang telah dibentuk oleh *Response Generator*. Apabila yang dikirimkan oleh *Response Generator* berupa pesan kesalahan yang tidak memerlukan informasi peta maka pesan langsung ditampilkan, namun apabila hasil yang dikirimkan oleh *Response Generator* berupa informasi yang memerlukan peta maka UI akan memanggil peta kemudian baru informasi disajikan.



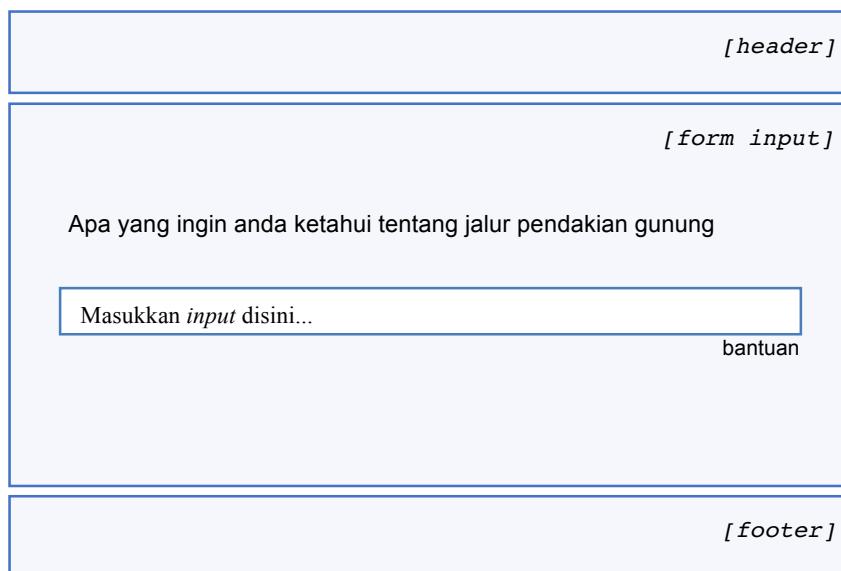
Gambar 4.14 Activity diagram

4.6 Perancangan Antarmuka

Aplikasi yang dikembangkan pada penelitian ini adalah aplikasi berbasis *web*, sehingga antarmuka yang digunakan dalam penelitian ini berupa halaman *web* yang dapat digunakan oleh pengguna untuk berinteraksi dengan sistem.

Sistem hanya menyediakan satu buah layanan, yaitu: pencarian informasi dengan *input* berupa kalimat bahasa Indonesia. Aplikasi menyediakan dua buah halaman, yaitu halaman *input* dan halaman *output*. Rancangan dari halaman *input* dijabarkan pada Gambar 4.15, sedangkan rancangan dari halaman *output* dijabarkan pada Gambar 4.16.

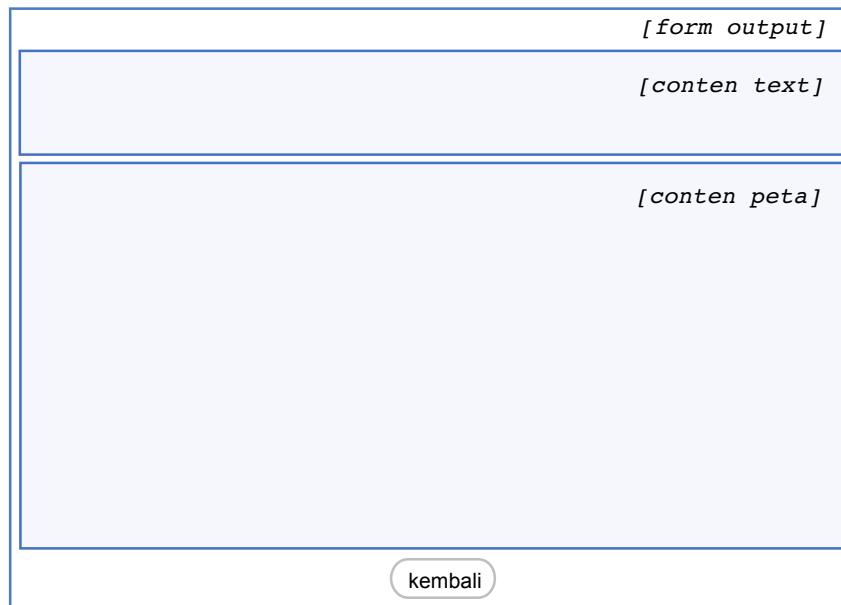
Rancangan Antarmuka *input* dibagi menjadi tiga bagian, yaitu: bagian *header*, bagian *form input*, dan bagian *footer*. Bagian *header* menjelaskan nama dan keterangan aplikasi. Bagian *form input* adalah bagian inti yang digunakan oleh pengguna untuk memberikan *input*. Bagian *footer* memuat informasi tambahan seperti tahun pembuatan aplikasi.



Gambar 4.15 Rancangan halaman *input*

Form yang terdapat dalam halaman *input* memuat label, sebuah *textbox*, dan sebuah *button*. Label digunakan untuk memberikan petunjuk untuk mengarahkan pengguna ketika memberikan *input*. *Textbox* digunakan untuk menampung *input* yang diberikan oleh pengguna, *input* dari pengguna akan diproses dengan menekan tombol *Enter* pada *Keyboard*. *Button* yang berlabel

bantuan adalah *link* yang digunakan untuk menampilkan jendela berisi panduan dalam melakukan pencarian.



Gambar 4.16 Rancangan halaman *output*

Rancangan antarmuka *output* merupakan sebuah *form* yang terdiri dari dua bagian, yaitu: *content text* dan *content peta*. Bagian *content text* merupakan bagian yang digunakan untuk menyajikan hasil dari pemrosesan *input*, hasil pemrosesan yang ditampilkan pada *content text* dapat berupa informasi hasil pencarian atau pesan kesalahan. *Content peta* digunakan untuk menampilkan hasil dari pemrosesan *input* berupa informasi dalam bentuk peta.

BAB V

IMPLEMENTASI

5.1 Implementasi Ontologi

Pengetahuan bahasa dan pengetahuan jalur pendakian yang digunakan pada penelitian ini diimplementasikan ke dalam ontologi menggunakan *tool Protege*. Pengetahuan bahasa diimplementasikan ke dalam ontologi Bahasa sedangkan pengetahuan jalur pendakian gunung diimplementasikan ke dalam ontologi *Mountaineering*.

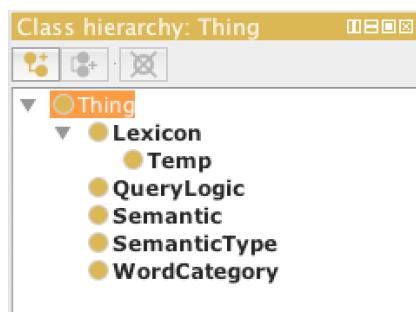
5.1.1 Implementasi ontologi Bahasa

Langkah awal dalam mengimplementasikan ontologi Bahasa yaitu mendefinisikan *Uniform Resource Identifier* (URI). URI untuk mendefinisikan ontologi Bahasa yaitu <http://www.semanticweb.org/ontologies/bahasa-v1>. Deskripsi URI dari ontologi Bahasa dijabarkan pada Gambar 5.1.



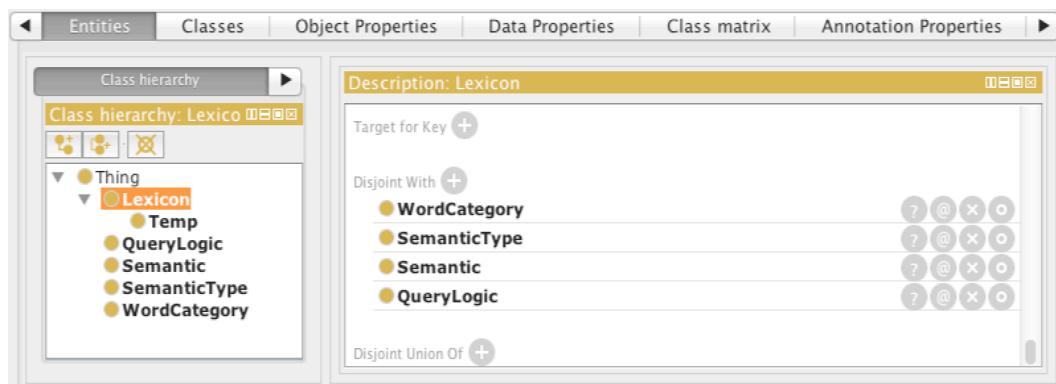
Gambar 5.1 Implementasi URI pada ontologi Bahasa

Setelah mendefinisikan URI langkah selanjutnya yaitu mendefinisikan *class*. Pendefinisan *class* pada *tool Protege* dilakukan menggunakan *tab classes*. *Class* yang didefinisikan ke dalam ontologi Bahasa yaitu: *Lexicon* [*Temp*], *WordCategory*, *Semantic*, *SemanticType*, *QueryLogic*. Implementasi *class* dari ontologi Bahasa dijabarkan pada Gambar 5.2.



Gambar 5.2 Implementasi *class* pada ontologi Bahasa

Setelah *class* didefinisikan langkah selanjutnya yaitu mendefinisikan relasi *disjoint* antar *class*. *Class* pada ontologi Bahasa yang memiliki relasi *disjoint* yaitu: *Class Lexicon disjoint* dengan *WordCategory*, *SemanticType*, *Semantic* dan *QueryLogic*. *Class WordCategory disjoint* dengan *Lexicon*, *SemanticType*, *Semantic* dan *QueryLogic*. *Class SemanticType disjoint* dengan *Lexicon*, *WordCategory*, *Semantic* dan *QueryLogic*. *Class Semantic disjoint* dengan *Lexicon*, *WordCategory*, *SemanticType* dan *QueryLogic*. *Class QueryLogic disjoint* dengan *Lexicon*, *WordCategory*, *SemanticType* dan *Semantic*. Implementasi relasi *disjoint* pada ontologi Bahasa menggunakan *tool Protege* dijabarkan pada Gambar 5.3.



Gambar 5.3 Implementasi *disjoint* pada ontologi Bahasa

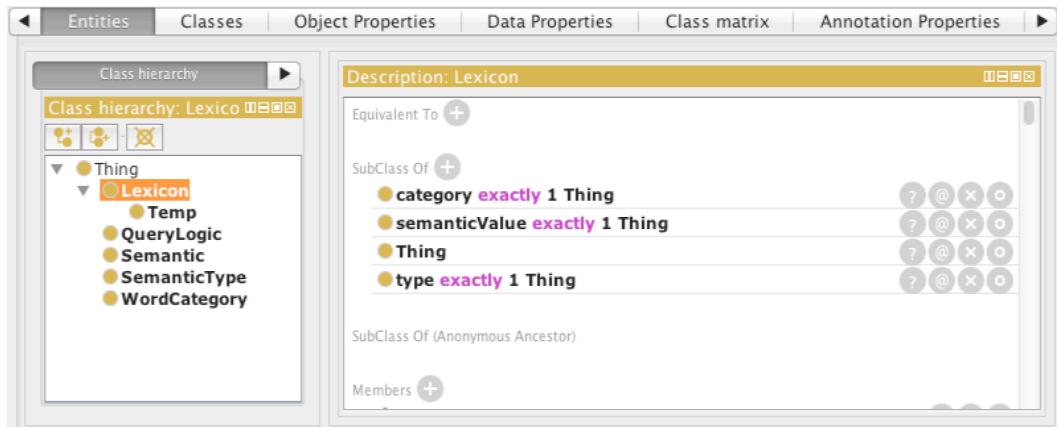
Tahap selanjutnya dalam implementasi ontologi yaitu mendefinisikan *Data property* dan *Object Property*. Pendefinisan *property* pada *tool Protege* dilakukan dengan menggunakan *tab properties*.

Data Property yang digunakan pada ontologi Bahasa yaitu: *word* dan *qpart*. *Object Property* yang digunakan pada ontologi Bahasa yaitu: *category*, *semanticValue*, *argCat0*, *argCat1*, *argCat2*, *stripCat*, *argType0*, *argType1*, *argType2*, *stripType*, *type* dan *parA*. Implementasi *property* dari ontologi Bahasa menggunakan *tool Protege* dijabarkan pada Gambar 5.4.



Gambar 5.4 Implementasi *property* pada ontologi Bahasa (a) *data property*. (b) *object property*

Tahapan selanjutnya setelah mendefinisikan *property* yaitu mendefinisikan kardinalitas *property* dari *class-class* yang terdapat di dalam ontologi. Pada *tool Protege*, kardinalitas diimplementasikan pada *tab classes*. Implementasi kardinalitas menggunakan *tool Protege* dijabarkan pada Gambar 5.5.



Gambar 5.5 Implementasi kardinalitas pada ontologi Bahasa

Tahapan selanjutnya yaitu mendefinisikan *instance* yang merupakan anggota dari *class* yang telah terbentuk. *Instance* didefinisikan menggunakan *tab Individuals* yang terdapat didalam *tool Protege*. Contoh *instance* pada ontologi Bahasa yaitu *instance gunung2* seperti yang dijabarkan pada Gambar 5.6. Implementasi dari *instance gunung2* menggunakan *tool Protege* dijabarkan pada Gambar 5.7. Semua *Instance* pada ontologi Bahasa diimplementasikan dengan cara yang sama.

Instance		Class
gunung2		Lexicon
Data Property	Nilai	Tipe
word	gunung	string
Object Property	Nilai	
category	nomina	
argCat0	nomina	
argCat1	frasa_nominal	
argCat2	-	
stripCat	-	
type	Tgunung2	
argType0	Tterdapat2, Tdimana11, Tcuaca2, Tinformasi2, Tlokasi2, Tpuncak31, Tkawah31, Ttemp32, Ttitik3, Tjurkun2, Tpos31, Tketinggian2, Tpeta, Tjurkunya2	
argType1	Ttemp21, Ttemp22	
argType2	-	
stripType	-	
semanticValue	Ngunung2	

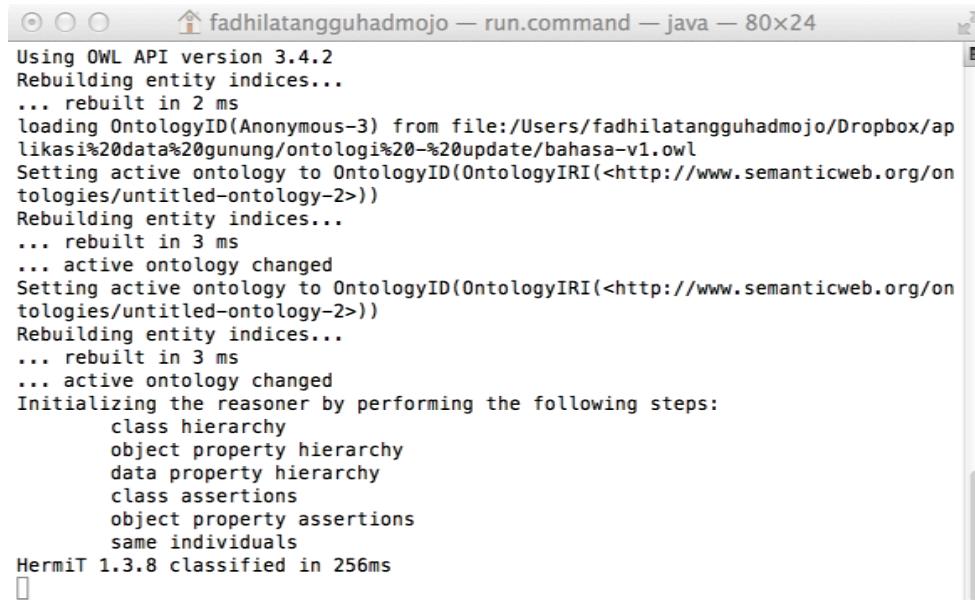
Gambar 5.6 Contoh deskripsi instance gunung2 pada class Lexicon

The screenshot shows the Protege interface with several panels:

- Top Bar:** Classes, Object Properties, Data Properties, Annotation Properties, Individuals, OntoGraf, SPARQL Query, Ontology Differences.
- Class hierarchy:** Shows the class Lexicon under the Thing category.
- Annotations:** For the individual gunung2, it shows the annotation `label` with the value `gunung`.
- Description:** Shows the type `Lexicon` and lists same individuals and different individuals.
- Property assertions:** Lists various properties and their values for the individual gunung2, such as `argType0`, `argCat1`, `type`, `semanticValue`, etc.
- Bottom Panel:** Shows the data property assertion `word "gunung"^^string`.

Gambar 5.7 Implementasi instance gunung2 dari class Lexicon pada ontologi Bahasa menggunakan tool Protege

Langkah terakhir dalam pengimplementasian ontologi adalah melakukan pemeriksaan inkonsistensi. Pemeriksaan inkonsistensi menggunakan *tool Protege* dilakukan melalui menu *Reasoner* → *Start reasoner*. Berdasarkan hasil pengecekan pada ontologi Bahasa yang dapat dilihat pada Gambar 5.8 menunjukkan bahwa pada ontologi Bahasa tidak terdapat inkonsistensi.



```

fadhilatangguhadmojo — run.command — java — 80x24
Using OWL API version 3.4.2
Rebuilding entity indices...
... rebuilt in 2 ms
loading OntologyID(Anonymous-3) from file:/Users/fadhilatangguhadmojo/Dropbox/aplikasi%20data%20gunung/ontologi%20-%20update/bahasa-v1.owl
Setting active ontology to OntologyID(OntologyIRI(<http://www.semanticweb.org/ontologies/untitled-ontology-2>))
Rebuilding entity indices...
... rebuilt in 3 ms
... active ontology changed
Setting active ontology to OntologyID(OntologyIRI(<http://www.semanticweb.org/ontologies/untitled-ontology-2>))
Rebuilding entity indices...
... rebuilt in 3 ms
... active ontology changed
Initializing the reasoner by performing the following steps:
    class hierarchy
    object property hierarchy
    data property hierarchy
    class assertions
    object property assertions
    same individuals
Hermit 1.3.8 classified in 256ms

```

Gambar 5.8 Hasil pemeriksaan inkonsistensi pada ontologi Bahasa

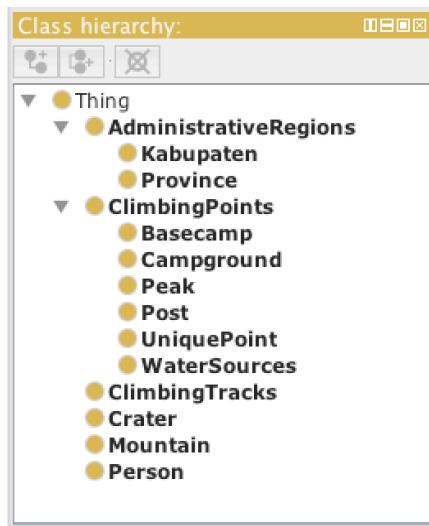
5.1.2 Implementasi ontologi *Mountaineering*

Tahapan implementasi pada ontologi *Mountaineering* sama seperti pada ontologi Bahasa. Langkah awal dalam mengimplementasikan ontologi *Mountaineering* yaitu mendefinisikan *Uniform Resources Identifier* (URI). URI yang digunakan untuk mendefinisikan ontologi *Mountaineering* adalah <http://www.semanticweb.org/ontologies/mountaineering>. Deskripsi URI dari ontologi *Mountaineering* dijabarkan pada Gambar 5.9.



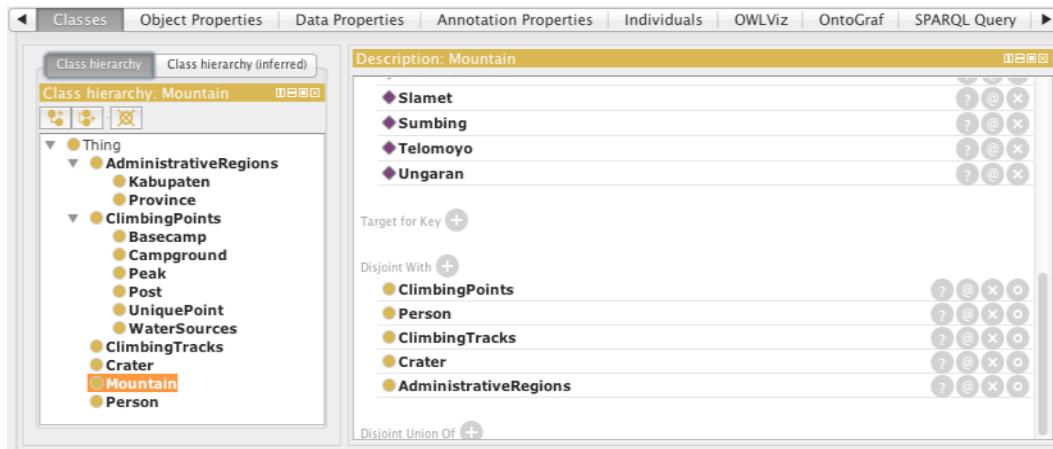
Gambar 5.9 Implementasi URI ontologi *Mountaineering*

Class yang didefinisikan didalam ontologi *Mountaineering* yaitu: *Mountain*, *ClimbingTracks*, *Crater*, *ClimbingPoints*, *Person* dan *AdministrativeRegions*. *Class ClimbingPoints* memiliki *subclass*, yaitu: *Basecamp*, *Campground*, *Post*, *Peak*, *WaterSources* dan *UniquePoint*. Implementasi dari *class* ontologi *Mountaineering* menggunakan *tool Protege* dijabarkan pada Gambar 5.10.



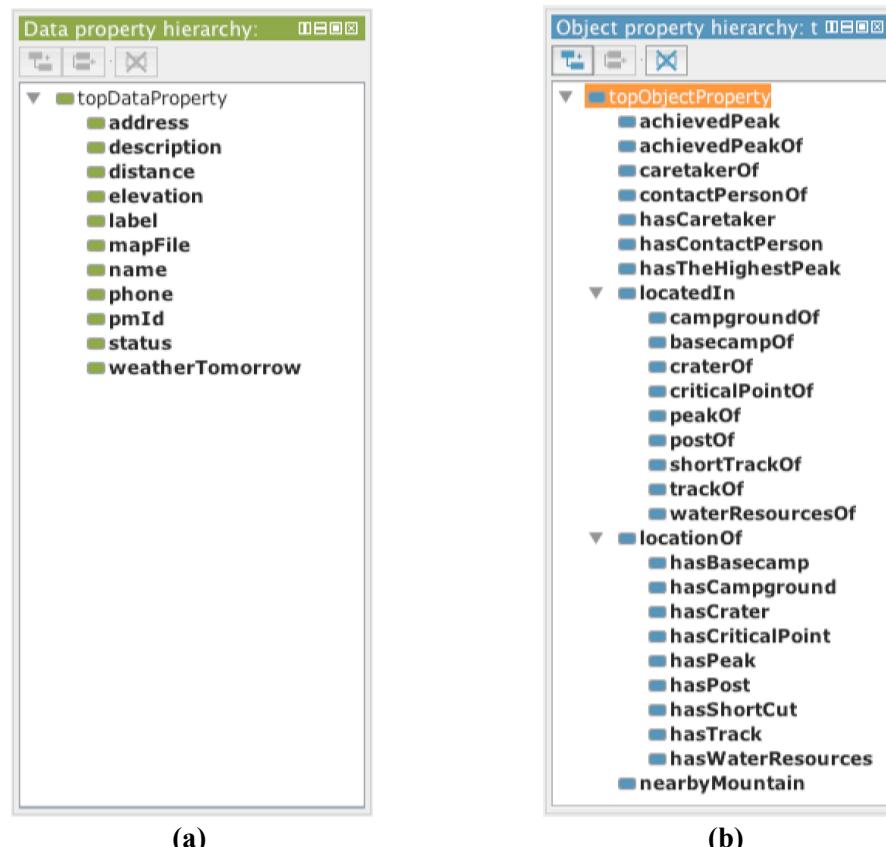
Gambar 5.10 Implementasi *class* pada ontologi *Moutnaineering*

Class pada ontologi *Mountaineering* yang memiliki relasi *disjoint* yaitu: *Class Mountain disjoint* dengan *Class AdministrativeRegions*, *ClimbingTracks*, *ClimbingPoints*, *Crater* dan *Person*. *Class AdministrativeRegions disjoint* dengan *Class ClimbingTracks*, *ClimbingPoints*, *Mountain*, *Crater* dan *Person*. *Class ClimbingPoints disjoint* dengan *Class Mountain*, *AdministrativeRegions*, *ClimbingTracks*, *Crater* dan *Person*. *Class ClimbingTracks disjoint* dengan *Class Mountain*, *AdministrativeRegions*, *ClimbingPoints*, *Crater* dan *Person*. *Class Crater disjoint* dengan *Class Mountain*, *AdministrativeRegions*, *ClimbingPoints*, *ClimbingTracks*, *Crater* dan *Person*. *Class Person disjoint* dengan *Class Mountain*, *AdministrativeRegions*, *ClimbingPoints*, *ClimbingTracks*, *Crater* dan *Person*. Implementasi relasi *disjoint* pada ontologi *Mountaineering* menggunakan *tool Protege* dijabarkan pada Gambar 5.11.



Gambar 5.11 Implementasi relasi *disjoint class* pada ontologi *Mountaineering*

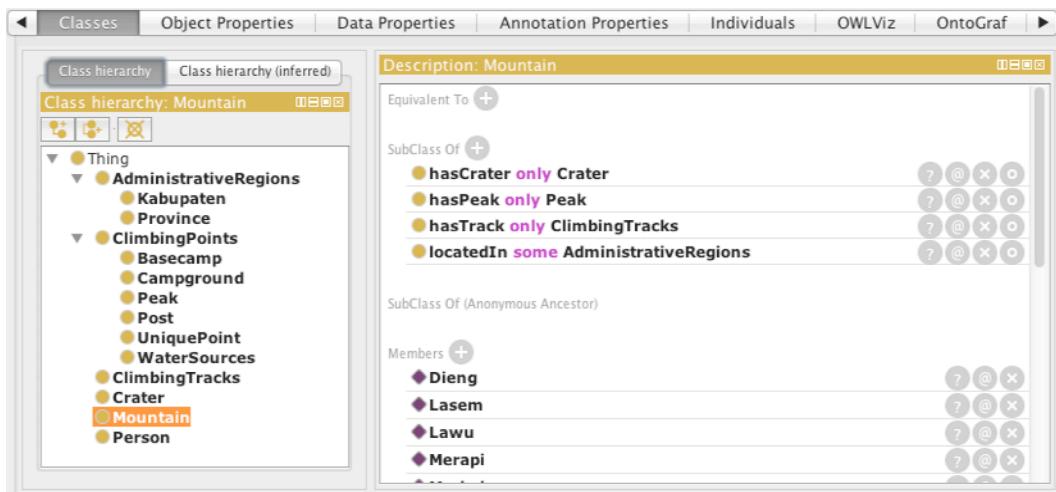
Data Property yang digunakan pada ontologi *Mountaineering* yaitu: *name*, *label*, *description*, *distance*, *elevation*, *label*, *status*, *phone*, *weatherTomorrow*. Implementasi dari *data property* untuk ontologi *Mountaineering* dijabarkan pada Gambar 5.12 (a).



Gambar 5.12 Implementasi *property* pada ontologi *Mountaineering* (a) *data property*. (b) *object property*

Object Property yang digunakan pada ontologi *Mountaineering* yaitu: *locatedIn* [*campGroundOf*, *basecampOf*, *trackOf*, *craterOf*, *postOf*, *peakOf*, *waterSourcesOf*, *criticalPointOf*], *contactPersonOf*, *nearbyMountain*, *caretakerOf* dan *locationOf* [*hasBasecamp*, *hascampground*, *hasTrack*, *hasPost*, *hasPeak*, *hasWaterSources*, *hasCriticalPoint*, *hasCrater*]. Implementasi dari *Object property* dari ontologi *Mountaineering* dijabarkan pada Gambar 5.12 (b).

Setelah mendefinisikan *property* tahap selanjutnya yang dilakukan adalah mendefinisikan kardinalitas *property* dari *class-class* yang terdapat di dalam ontologi *Mountaineering*. Menggunakan *tool Protege* kardinalitas *property* diimplementasikan pada tab *classes*. Implementasi kardinalitas menggunakan *tool Protege* dijabarkan pada Gambar 5.13.



Gambar 5.13 Implementasi kardinalitas pada ontologi *Mountaineering*

Tahap selanjutnya yaitu implementasi *instance* dari *class* yang terdapat di dalam ontologi *Mountaineering*. Contoh *instance* pada *class* *Mountain* yaitu *instance Merapi* seperti yang dideskripsikan pada Gambar 5.14. Implementasi dari *instance Merapi* menggunakan *tool Protege* dijabarkan pada Gambar 5.15. *Instance* lain pada ontologi *Mountaineering* diimplementasikan dengan cara yang sama.

Instance		Class	
Merapi		Mountain	
Data Property	Nilai	Type	
<i>label</i>	gunung	string	
<i>mapFile</i>	KML/JawaTengah/JawaTengah.kml	string	
<i>pmlId</i>	gn-001	string	
<i>name</i>	Merapi	string	
<i>elevation</i>	2930	integer	
<i>description</i>	Gunung Merapi : gunung berapi aktif bertipe stratovolcano Lokasi : Sebagian di Prov. Jawa Tengah (Kab. Boyolali di sisi utara dan timur, Kab. Magelang di sisi barat) dan sebagian di Prov. DI Yogyakarta (Kab. Sleman di sisi selatan) Kawah : Kawah Mati Puncak : Puncak Garuda Jalur Pendakian : New Selo (Boyolali)	string	
Object Property	Nilai		
<i>hasCrater</i>	Merapi_Kawah_Mati		
<i>locationOf</i>	New_Selo_Camp_G_Pos_2, Merapi_Kawah_Mati, New_Selo_Camp_G_Pasar_Bubrah, New_Selo_Pos_2, New_Selo_Monumen_Pendaki, Merapi_New_Selo_Jalur_Lumut, New_Selo_Gazebo, Merapi_New_Selo_Barameru, New_Selo_Camp_G_Pos_1		
<i>hasCaretaker</i>	Maridjan		
<i>nearbyMountain</i>	Sumbing, Merbabu		
<i>hasPeak</i>	Merapi_Puncak_Garuda		

Gambar 5.14 Contoh deskripsi instance Merapi pada class Mountain

The screenshot shows the Protege interface with the following details:

- Class hierarchy:** Shows the Mountain class under Thing, with its subclasses: AdministrativeRegions, ClimbingPoints, ClimbingTracks, Crater, Mountain, and Person.
- Annotations:** Merapi
 - Annotations:**
 - label* [type: string]: Gunung Merapi / Taman Nasional Gunung Merapi
- Description:** Merapi
 - Types:** Mountain
 - Same Individual As:**
 - Different Individuals:**
- Property assertions:** Merapi
 - hasCrater: Merapi_Kawah_Mati
 - locationOf: New_Selo_Camp_G_Pos_2, Merapi_Kawah_Mati, New_Selo_Camp_G_Pasar_Bubrah, New_Selo_Pos_2, New_Selo_Monumen_Pendaki, Merapi_New_Selo_Jalur_Lumut, New_Selo_Gazebo, Merapi_New_Selo_Barameru, New_Selo_Camp_G_Pos_1
 - hasCaretaker: Maridjan
 - locationOf: New_Selo_Pos_2
 - nearbyMountain: Sumbing
 - locationOf: New_Selo_Gazebo
 - hasPeak: Merapi_Puncak_Garuda
 - locationOf: Merapi_New_Selo_Barameru
 - locationOf: New_Selo_Camp_G_Pos_1
- Data property assertions:**
 - label*: "Gunung Merapi"^^string
 - mapFile*: "KML/JawaTengah/JawaTengah.kml"^^string
 - elevation*: 2930
 - description*: "Gunung Merapi : gunung berapi aktif bertipe stratovolcano. Lokasi : Sebagian di Prov. Jawa Tengah (Kab. Boyolali di sisi utara dan timur, Kab. Magelang di sisi barat) dan sebagian di Prov. DI Yogyakarta (Kab. Sleman di sisi selatan). Kawah : Kawah Mati Puncak : Puncak Garuda Jalur Pendakian : New Selo (Boyolali)"
 - pmlId*: "gn-001"^^string
 - name*: "Merapi"^^string

Gambar 5.15 Implementasi instance Merapi dari class Mountain menggunakan tool Protege

Langkah terakhir dalam pengimplementasian ontologi adalah melakukan pemeriksaan inkonsistensi. Pemeriksaan inkonsistensi menggunakan *tool Protege* dilakukan melalui menu *Reasoner* → *Start reasoner*. Berdasarkan hasil pengecekan pada ontologi *Mountaineering* yang dapat dilihat pada Gambar 5.16 menunjukkan bahwa pada ontologi tersebut tidak terdapat inkonsistensi.

```

Using OWL API version 3.4.2
Rebuilding entity indices...
... rebuilt in 1 ms
loading OntologyID(Anonymous-0) from file:/Users/fadhilatangguhadmojo/Dropbox/aplikasi%20data%20gunung/ontologi%20-%20update/mountaineering_new.owl
Setting active ontology to OntologyID(OntologyIRI(<http://www.semanticweb.org/ontologies/mountaineering>))
Rebuilding entity indices...
... rebuilt in 5 ms
... active ontology changed
Setting active ontology to OntologyID(OntologyIRI(<http://www.semanticweb.org/ontologies/mountaineering>))
Rebuilding entity indices...
... rebuilt in 3 ms
... active ontology changed
Initializing the reasoner by performing the following steps:
  class hierarchy
  object property hierarchy
  data property hierarchy
  class assertions
  object property assertions
  same individuals
HermiT 1.3.8 classified in 2551ms

```

Gambar 5.16 Hasil pemeriksaan inkonsistensi pada ontologi *Mountaineering*

5.2 Implementasi KML

KML digunakan untuk menyimpan data-data geografis hasil obeservasi menggunakan alat GPS. Data-data geografis mendeskripsikan *instance* yang ada pada ontologi *Mountaineering*. *Instance* pada ontologi *Mountaineering* yang memiliki koordinat geografis yaitu *instance* dari *class Mountain*, *ClimbingTracks*, *Crater* dan *ClimbingPoints*.

KML diimplementasikan menggunakan *Text Editor*. Dua jenis *placemark* yang diimplementasikan di dalam KML yaitu titik (*point*) dan jalur (*linestring*). Atribut *tag* yang digunakan untuk mendeskripsikan titik (*point*), yaitu: *<name>*, *<extendeddata>*, *<data name>*, *<value>*, *<description>*, *<styleurl>*, *<point>*, *<coordinates>*. Cuplikan implementasi KML untuk mendeskripsikan satu titik (*point*) dapat dilihat pada Gambar 5.17. *Point* lain diimplementasikan dengan cara yang sama.

```

1. <Placemark>
2.   <name>Gardu Pandang New Selo</name>
3.   <ExtendedData>
4.     <Data name="pid">
5.       <value>p0001</value>
6.     </ExtendedData>
7.   <description>
8.     Gardu Pandang New Selo adalah kawasan wisata umum
9.     terdapat tempat parkir mobil yang luas, warung
10.    makan, toilet
11.  </description>
12.  <styleUrl>#point_pendakian</styleUrl>
13.  <Point>
14.    <coordinates>
15.      110.4527944284596,-7.51578663958974,1785
16.    </coordinates>
17.  </Point>
18. </Placemark>
19. <Placemark>
20. ... <!-- other placemark -->
21. </Placemark>

```

Gambar 5.17 Cuplikan data KML berupa *placemark* untuk titik (*point*)

Atribut *tag* yang digunakan untuk mendeskripsikan jalur (*linestring*), yaitu: *<name>*, *<extendeddata>*, *<data name>*, *<value>*, *<description>*, *<styleurl>*, *<lineString>* dan *<coordinates>*. Cuplikan implementasi KML untuk mendeskripsikan satu jalur (*linestring*) dapat dilihat pada Gambar 5.18. Jalur (*linestring*) yang lain diimplementasikan dengan cara yang sama.

```

1. <Placemark>
2.   <name>Basecamp Barameru - Gardu Pandang New Selo</name>
3.   <description>Jalur berupa aspal menanjak</description>
4.   <ExtendedData>
5.     <Data name="plid">
6.       <value>p10002</value>
7.     </ExtendedData>
8.   <styleUrl>#jalur_utama</styleUrl>
9.   <LineString>
10.    <tessellate>1</tessellate>
11.    <gx:altitudeMode>clampToSeaFloor</gx:altitudeMode>
12.    <coordinates>
13.      110.4540968772241,-7.512093830807585,0
14.      110.4540288336796,-7.512071899610136,0
15.      110.4540031738139,-7.512152907569693,0
16.      110.4538827704056,-7.512424347162289,0
17.      110.4536893152096,-7.512670875432103,0
18.      110.4535230739231,-7.512896242244191,0
19.      110.4533850203045,-7.513077514328254,0
20.      110.4532829288595,-7.51324893343194,0
21.      110.4532023492393,-7.513505831718011,0
22.    </coordinates>
23.  </LineString>
24. </Placemark>
25. <Placemark>
26.   <!-- other placemark -->
27. </Placemark>

```

Gambar 5.18 Cuplikan data KML berupa *placemark* untuk jalur (*linestring*)

5.3 Implementasi Aplikasi

Implementasi aplikasi dilakukan dengan menggunakan perangkat lunak sebagai berikut:

- *Eclipse Java EE IDE for Web Developers (Luna)*
- Bahasa pemrograman *Java, Java Server Page (JSP), servlet dan Java Script*
- Bahasa *query SPARQL*.
- *Jena API Library*
- *Google maps API 3.0*
- KML parser *GeoXml3*
- *Jazzy Spell Checker Library*
- *Web server Apache Tomcat 7.0*

5.3.1 Penerimaan *input* dan fitur *spelling checker*

Pengguna dapat memberikan *input* pencarian melalui halaman *utama.jsp*. Pada halaman *utama.jsp*, *input* dari pengguna diterima oleh *form action*. Cuplikan kode dari halaman *utama.jsp* dapat dilihat pada Gambar 5.19.

```

1.   <script src="js/lib/gb.js"></script>
2.   <script src="js/lib/jquery/jquery-1.11.1.min.js"></script>
3.   <script src="js/lib/jqueryui/jquery-ui.min-1.11.1.js"></script>
4.     <!-- other script-->

5.   <form action="token" id="calculate" method="get" class="index">
6.     
7.     <fieldset id="pos_1">
8.       <label id="masukkan"> informasi <strong>gunung / jalur pendakian
          gunung apa yang ingin anda cari : </label>
9.       <a href="bantuan.jsp">
           </a>
10.      <input name="i" id="i" maxlength="200" type="text"
           placeholder="Pencarian..." />
11.      <input type="submit" title="proses" value="" style="width: 0px;
           height: 0px; opacity: 0" />
12.    </fieldset>
13.  </form>
```

Gambar 5.19 Cuplikan kode dari halaman *utama.jsp*

Selama proses *input* berlangsung dan pengguna belum melakukan *submit*, *script gb.js* yang dipanggil oleh halaman *utama.jsp* (Gambar 5.19 baris 1) akan mendeteksi setiap karakter yang di-*input* (Gambar 5.19 baris 10). Cuplikan kode dari *gb.js* dapat dilihat pada Gambar 5.20.

```

1.  $(function() {
2.      $("#i").autocomplete({
3.          source : function(request, response) {
4.              $.ajax({
5.                  url : "wordschecker",
6.                  type : "GET",
7.                  dataType : "json",
8.                  data : {word : request.term},
9.                  success : function(data) {
10.                      response($.map(data, function(item) {
11.                          return {value : item};
12.                      }));
13.                  },
14.                  error : function(error) {
15.                      alert('error: ' + error);
16.                  }
17.              });
18.          },
19.          minLength : 2
20.      });
21.  });

```

Gambar 5.20 Cuplikan kode dari script *gb.js*

Script *gb.js* bertugas mengirimkan setiap karakter *input* kepada *class wordchecker* (Gambar 5.20 baris 5). Di dalam *class wordchecker*, *request* dari *gb.js* diterima menggunakan *method doGet* yang kemudian nilai dari *request* diteruskan kepada *method checkspelling*. Cuplikan kode dari *method doGet* dapat dilihat pada Gambar 5.21.

```

1. protected void doGet(HttpServletRequest request,
2.     HttpServletResponse response) throws ServletException, IOException {
3.     String name = request.getParameter("word");
4.     String buffer = "";
5.     checkSpelling(name);
6.     int indexRight = 0;
7.     JSONObject json = new JSONObject();
8.     ...
9. }

```

Gambar 5.21 *method doGet* di dalam *class wordchecker*

Method checkspelling bertugas melakukan proses pengecekan ejaan kata menggunakan fitur-fitur yang sudah tersedia di dalam *library Jazzy Spellchecker* yang dikembangkan oleh Idzelis (2005). Cuplikan kode dari *method checkspelling* dapat dilihat pada Gambar 5.22.

Fitur-fitur dari *library Jazzy* yang digunakan oleh *method checkSpelling* yaitu *SpellDictionary*, *SpellDictionaryHashMap*, *SpellChecker*, *SpellCheckListener* dan *StringWordTokenizer*.

```

1. public void checkSpelling(String words) {
2.     isError = false;
3.     question.clear();
4.     System.out.println(" " + words);
5.     realQuestion = words.split(" ");
6.     rightWords = new ArrayList<Integer>();
7.     for (int i = 0; i < realQuestion.length; i++) {
8.         rightWords.add(i);
9.     }
10.    try {
11.        SpellDictionary dictionary = new SpellDictionaryHashMap(new File(
12.            "bahasa.0"));
13.        SpellChecker spellChecker = new SpellChecker(dictionary);
14.        spellChecker.addSpellCheckListener(new SuggestionListener());
15.        spellChecker.checkSpelling(new StringWordTokenizer(words));
16.    } catch (FileNotFoundException e) {
17.        e.printStackTrace();
18.    } catch (IOException e) {
19.        e.printStackTrace();
20.    }
21. }
```

Gambar 5.22 method *checkSpelling* di dalam *class wordchecker*

SpellDictionary dan *SpellDictionaryHashMap* (Gambar 5.22 baris 11) digunakan untuk mendefinisikan berkas *dictionary* (kamus kata) yang berisi koleksi kata yang dimiliki sistem. Berkas *dictionary* yang digunakan pada penelitian ini disusun oleh *class BahasaGenerator*. Cuplikan kode dari *class BahasaGenerator* dapat dilihat pada Gambar 5.23.

Bahasa Generator bertugas membentuk *dictionary* (kamus kata) berdasarkan kosakata yang terdapat di dalam ontologi dengan menggunakan *method bahasaReader.getAllKata* dan *method mountReader.getAllKata* (Gambar 5.23 baris 6 dan 8).

Method bahasaReader.getAllKata bertugas mengambil kata dari ontologi Bahasa dengan melakukan *query* pada *data property* yaitu *word* yang dimiliki semua *individual* yang ada di dalam *class Lexicon*. Cuplikan kode *method getAllKata* untuk ontologi Bahasa dapat dilihat pada Gambar 5.24 (a).

Method mountReader.getAllKata mengambil kata dari ontologi *Mountaineering* dengan melakukan *query* pada *data property* yaitu *name* yang dimiliki setiap *individual* dari semua *class* yang ada di dalam ontologi *Mountaineering*. Cuplikan kode *method getAllKata* untuk ontologi *Mountaineering* dapat dilihat pada Gambar 5.24 (b).

```

1. public class BahasaGenerator {
2.     public static final String FILENAME = "bahasa.0";
3.     public static void generateBaseBahasa() {
4.         OntoBahasaReader bahasaReader = new OntoBahasaReader();
5.         MountaineeringReader mountReader = new MountaineeringReader();
6.         List<String> katas = bahasaReader.getAllKata();
7.         List<String> resultList = new ArrayList<String>();
8.         katas.addAll(mountReader.getAllKata());
9.         String[] katakata = null;
10.        boolean alreadyExist = false;
11.
12.        Collections.sort(katas, comparator);
13.        try {
14.            FileWriter writer = new FileWriter(new File(FILENAME));
15.            for (String kata : katas) {
16.                if (CheckIfKataContainSpasi(kata)) {
17.                    katakata = ParseJadiKata(kata);
18.                    for (String item : katakata) {
19.                        if ((item.contains("-") == false)
20.                            && item.trim().length() > 0) {
21.                            alreadyExist = false;
22.                            for (String myWord : resultList) {
23.                                if (myWord.equals(item)) {
24.                                    alreadyExist = true;
25.                                    break;
26.                                }
27.                            }
28.                            if (alreadyExist == false) {
29.                                resultList.add(item);
30.                            }
31.                        }
32.                    } else {
33.                        ... //cek kondisi lain
34.                    }
35.                }
36.                int indexKata = 0;
37.                Collections.sort(resultList);
38.                for (String resultKata : resultList) {
39.                    if (resultKata.length() > 1
40.                        && indexKata < resultList.size() - 1) {
41.                        writer.append(resultKata + "\n");
42.                    } else if (resultKata.length() > 1
43.                        && indexKata == resultList.size() - 1) {
44.                        writer.append(resultKata);
45.                    }
46.                    indexKata++;
47.                }
48.                writer.flush();
49.                writer.close();
50.            } catch (IOException e) {
51.                e.printStackTrace();
52.            }
53.        }

```

Gambar 5.23 Class *BahasaGenerator*

Kosakata yang dihasilkan oleh *method bahasaReader.getAllKata* dan *method mountReader.getAllKata* disusun menjadi koleksi kata dengan menghilangkan spasi, menghilangkan kata yang sama dan diurutkan berdasarkan abjad, kemudian disimpan ke dalam berkas *bahasa.0*. Implementasi pembentukan berkas *bahasa.0* dapat dilihat pada Gambar 5.23 baris 11 - 47.

```

1. public List<String> getAllKata() {
2.     String query = PREFIX_QUERY + " SELECT DISTINCT ?" + NILAI
3.     + " WHERE { ?" + SUBJECT + " bhs:word ?" + NILAI + " ." + " }";
4.     ResultSet rs = runRawQuery(query);
5.     return resultSetToString(rs);
6. }

```

(a)

```

1. public List<String> getAllKata() {
2.     String query = PREFIX_QUERY + "SELECT DISTINCT ?nama "
3.     + "WHERE { ?subject mount:name ?nama . }";
4.     ResultSet rs = runRawQuery(query);
5.     return resultSetToString(rs);
6. }

```

(b)

**Gambar 5.24 Cuplikan kode method *getAllKata* (a) class *OntoBahasaReader*.
(b) class *MountaineeringReader***

Selama proses pengecekan kesalahan ejaan, *spellchecker* menggunakan berkas *bahasa.0* yang dibentuk oleh *BahasaGenerator* sebagai *dictionary* (kamus kata). Pemanggilan berkas *bahasa.0* dapat dilihat pada Gambar 5.22 baris 11-12.

Proses pengecekan dikerjakan oleh *SpellChecker.checkSpelling* dengan membandingkan susunan karakter pada kata *input* dengan susunan karakter pada kata yang terdapat didalam *dictionary* (Gambar 5.22 baris 13 dan 15). Selama proses pengecekan berlangsung *spellCheker* memanggil method *SugestionListener* (Gambar 5.22 baris 14) untuk mendeteksi susunan karakter yang berbeda dari suatu kata yang susunan karakternya sudah memiliki kesamaan. Apabila perbedaan ditemukan, maka *SugestionListener* akan mengirimkan saran kata (*sugestion*) dari *dictionary* sebagai respons atas *request* dari *gb.js*. Script *gb.js* menerima respons dari *wordschecker* kemudian menampilkan *sugestion* kata kepada *user interface* dengan menggunakan fungsi *autocomplete* dari *library jquery* (Gambar 5.19 baris 2 dan 3).

Fitur *spelling checker* merupakan fitur tambahan dan tidak menjadi fokus dari penelitian ini. Oleh karena itu, metode dan algoritma yang digunakan di dalam *library Jazzy Spellchecker* Idzelis (2005) tidak akan dibahas pada penelitian ini.

5.3.2 Input handling dan preprocessing

Input yang telah di-submit oleh pengguna dikirimkan (*request*) oleh *form action* dari halaman *utama.jsp* kepada *class token* yang berfungsi sebagai *servlet* (Gambar 5.19 baris 5). *Class token* menerima *request* dan memberikan respons menggunakan *method doGet*. Cuplikan kode dari *method doGet* pada *class token* dapat dilihat pada Gambar 5.25.

Input yang diterima oleh *method doGet* akan melalui beberapa proses *handling*. Proses *handling input* yang dilakukan didalam *method doGet*, yaitu: membuang karakter *non-alphanumeric* apabila pada *input* mengandung karakter *non-alphanumeric* (Gambar 5.25 baris 5), mengirimkan respons balik berupa *blank form* kepada halaman *utama.jsp* apabila *input* yang di-submit kosong atau hanya berupa karakter spasi saja (Gambar 5.25 baris 7-11), mengganti karakter spasi apabila pada *input* berisi rentetan kata yang diantara kata dipisahkan oleh lebih dari satu spasi. *Input* dengan spasi pemisah antar kata lebih dari satu spasi diproses oleh *method GetStringArrayFromString*. Cuplikan kode dari *method GetStringArrayFromString* dapat dilihat pada Gambar 5.26.

```

1. protected void doGet(HttpServletRequest request, HttpServletResponse
   response) throws ServletException, IOException {
2.     long start = System.currentTimeMillis();
3.     String kalimat = request.getParameter("i");
4.     System.out.println("sebelum replace = " + kalimat);
5.     kalimat = kalimat.replaceAll("[^a-zA-Z0-9, ]", "");
6.     System.out.println("setelah replace = " + kalimat);
7.     if (kalimat.trim().isEmpty()) {
8.         String referer = request.getHeader("Referer");
9.         response.sendRedirect(referer);
10.        return;
11.    }
12.    Preprocessor preprocessor = new Preprocessor();
13.    boolean isValid = preprocessor.CheckIsSentenceValid
14.      (GetStringArrayFromString(kalimat));
15.    if (!isValid) {
16.        request.setAttribute("input", kalimat);
17.        request.setAttribute("output", preprocessor
18.          .generateResponseMessage(Preprocessor.ResponseType.ERROR_INPUT,
19.          kalimat));
20.        request.getRequestDispatcher("/tampiltoken.jsp").forward(request,
21.        response);
22.        return;
23.    }
24.    ... //baris kode lain dari method
25. }
```

Gambar 5.25 *method doGet*

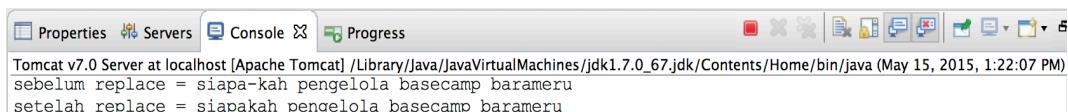
```

1. List<String> GetStringArrayFromString(String sentence) {
2.     List<String> result = new ArrayList<String>();
3.     String[] words = sentence.split(" ");
4.     for (int i = 0; i < words.length; i++) {
5.         if (words[i].trim().length() > 0) {
6.             result.add(words[i].trim());
7.         }
8.     }
9.     return result;
10. }

```

Gambar 5.26 method GetStringArrayFromString

Cuplikan hasil proses *handling* untuk membuang karakter *non-alphanumeric* menggunakan contoh *input* "siapa-kah pengelola basecamp barameru" dapat dilihat pada Gambar 5.27.



The screenshot shows a Java IDE interface with tabs for Properties, Servers, Console, and Progress. The Console tab displays the output of a Tomcat v7.0 Server at localhost. The output shows two lines of text: 'belum replace = siapa-kah pengelola basecamp barameru' and 'setelah replace = siapakah pengelola basecamp barameru'. This demonstrates the replacement of non-alphanumeric characters ('-') with spaces (' ') in the input string.

Gambar 5.27 Handling untuk non-alphanumeric

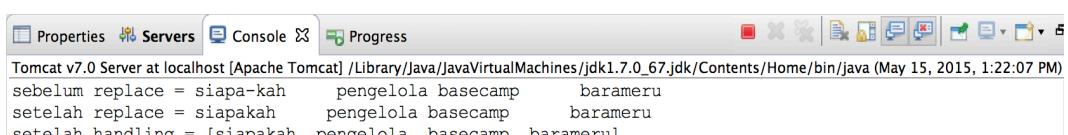
Cuplikan proses *handling* untuk mengirimkan respons balik halaman *utama.jsp* apabila *input* yang di-submit kosong dapat dilihat pada Gambar 5.28.



The screenshot shows a Java IDE interface with tabs for Properties, Servers, Console, and Progress. The Console tab displays the output of a Tomcat v7.0 Server at localhost. It shows the URL 'http://localhost:8080/GunungBeta/utama.jsp' and the response 'belum replace =' followed by 'setelah replace ='. This indicates that the server returned the original input without modification because it was empty.

Gambar 5.28 Handling untuk input kosong

Cuplikan hasil proses *handling* untuk mengganti karakter spasi apabila pada *input* berisi rentetan kata yang diantara kata dipisahkan oleh spasi lebih dari satu. Misalnya pada *input* "siapa-kah(spasi)(spasi)(spasi)pengelola(spasi) basecamp(spasi)(spasi)(spasi) barameru" dapat dilihat pada Gambar 5.29.



The screenshot shows a Java IDE interface with tabs for Properties, Servers, Console, and Progress. The Console tab displays the output of a Tomcat v7.0 Server at localhost. It shows the input 'belum replace = siapa-kah pengelola basecamp barameru' and the result 'setelah replace = siapakah pengelola basecamp barameru'. This demonstrates the replacement of multiple consecutive spaces with a single space.

Gambar 5.29 Handling untuk spasi lebih dari satu

Preprocessing dimulai ketika *method doGet* memanggil *class preprocessor* dan mengirim *input* yang telah di-handling kepada *method CheckIsSentenceValid* (Gambar 5.25 baris 13). Fungi dari *preprocessor* yaitu melakukan validasi *input* dan menghasilkan urutan kata. Proses validasi *input*

dikerjakan oleh *method ValidateInput*. Cuplikan kode *method ValidateInput* dapat dilihat pada Gambar 5.30.

```

1. public boolean ValidateInput(List<String> sentence, int beginCheckIndex) {
2.     int wordsCount = sentence.size();
3.     String wordToCheck = "";
4.     String trueWord = "";
5.     int indexTrueWord = -1;
6.     int wordsKategory = -1;
7.
8.
9.     if (beginCheckIndex > wordsCount - 1) {
10.         System.out.println("kata tidak ketemu : " + listKataTidakKetemu);
11.         System.out.println("kata ketemu : " + listKataKetemu);
12.         if (listKataTidakKetemu.size() == 0) {
13.             if (listFoundGunung.size() == 0 || listFoundGunung.size() > 4)
14.                 return false;
15.             }
16.             return true;
17.         } else if (listKataTidakKetemu.size() > 0) {
18.             return false;
19.         }
20.         return !(listKataTidakKetemu.size() > 0);
21.     }
22.     for (int i = beginCheckIndex; i < sentence.size(); i++) {
23.         if (i == beginCheckIndex) {
24.             wordToCheck += sentence.get(i).trim();
25.         } else {
26.             wordToCheck += " " + sentence.get(i).trim();
27.         }
28.         if (CheckIfWordsInBahasa(wordToCheck)) { //cek kata ke ontologi bahasa
29.             trueWord = wordToCheck;
30.             wordsKategory = 0;
31.             indexTrueWord = i;
32.             continue;
33.         }
34.         if (CheckIfWordsInGunung(wordToCheck)) { //cek kata ke ontologi domain
35.             trueWord = wordToCheck;
36.             wordsKategory = 1;
37.             indexTrueWord = i;
38.         }
39.     }
40.     if (indexTrueWord >= 0) {
41.         if (wordsKategory == 0) {
42.             listFoundBahasa.add(trueWord);
43.             String query = bahasaReader.createSimpleQuery(trueWord);
44.             ResultSet rs = bahasaReader.runRawQuery(query);
45.             List<OntoBahasa> bahasas =
46.                 bahasaReader.simpleResultSetToListBahasa(rs);
47.             this.bahasas.add(0, bahasas);
48.         } else if (wordsKategory == 1) {
49.             listFoundGunung.add(trueWord);
50.             List<OntoBahasa> tempBahasa = bahasaReader.queryTemplate(trueWord);
51.             bahasas.add(0, tempBahasa);
52.         }
53.         listKataKetemu.add(trueWord);
54.         System.out.println("bahasas setelah preproses : " + bahasas);
55.         return ValidateInputModif(sentence, indexTrueWord + 1);
56.     } else {
57.         listKataTidakKetemu.add(sentence.get(beginCheckIndex));
58.     }
59. }
```

Gambar 5.30 method ValidateInput pada class Preprocessor

Proses validasi *input* dimulai dari kata yang paling kiri kemudian bergeser ke kanan hingga selesai (Gambar 5.30 baris 22-59). Proses validasi dilakukan

dengan mencocokkan kata pada *input* dengan kata pada ontologi (Gambar 5.30 baris 28-38).

Proses pengecekan kata ke ontologi Bahasa dikerjakan oleh *method* *CheckIfWordInBahasa* dan untuk pengecekan kata ke ontologi *Mountaineering* *method* *ValidateInput* memanggil *method* *CheckIfWordInGunung*. Cuplikan kode dari *method* *CheckIfWordsInBahasa* dan *method* *CheckIfWordsInGunung* dapat dilihat pada Gambar 5.31.

Method *CheckIfWordsInBahasa* dan *method* *CheckIfWordsInGunung* juga menggunakan *method* *getAllKata* pada Gambar 5.19 untuk menghasilkan *listbahasa* (5.31 a baris 5) dan *listGunung* (5.31 b baris 5).

```

1.  boolean CheckIfWordsInBahasa(String word) {
2.      if (listBahasa == null) {
3.          return false;
4.      }
5.      for (String item : listBahasa) {
6.          if (item.toLowerCase().equals(word.toLowerCase())) {
7.              return true;
8.          }
9.      }
10.     return false;
11. }
```

(a)

```

1.  boolean CheckIfWordsInGunung(String word) {
2.      if (listGunung == null) {
3.          return false;
4.      }
5.      for (String item : listGunung) {
6.          if (item.toLowerCase().equals(word.toLowerCase())) {
7.              return true;
8.          }
9.      }
10.     return false;
11. }
```

(b)

Gambar 5.31 (a) *method CheckIfWordInBahasa*. (b) *method CheckIfWordInGunung*

Satuan kata yang terdapat pada ontologi dapat terdiri dari satu kata atau lebih, oleh karenanya kata terpanjang yang cocok yang akan digunakan (*longest word sequence match*). Misalnya diketahui *input* terdiri dari dua kata, yaitu "new" dan "selo". Proses validasi dari kata tersebut setelah dibandingkan dengan kata pada ontologi akan menghasilkan tiga kata yang cocok, yaitu: "new", "selo" dan "new selo". Dari tiga kata tersebut maka kata "new selo" yang akan digunakan sebagai satuan kata dalam urutan kata. Cuplikan hasil proses validasi

menggunakan *input* "siapa pengelola basecamp new selo" dapat dilihat pada Gambar 5.32.

```

Properties Servers Console Progress
Tomcat v7.0 Server at localhost [Apache Tomcat] /Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (May 15, 2015, 3:09:30 PM)
Waktu konstruksi preproses = 151
wordtocek : siapa
wordtocek : siapa pengelola
wordtocek : siapa pengelola basecamp
wordtocek : siapa pengelola basecamp new
wordtocek : siapa pengelola basecamp new selo
wordtocek : pengelola
wordtocek : pengelola basecamp
wordtocek : pengelola basecamp new
wordtocek : pengelola basecamp new selo
wordtocek : basecamp
wordtocek : basecamp new
wordtocek : basecamp new selo
wordtocek : new
wordtocek : new selo
kata tidak ketemu : []
kata ketemu : [siapa, pengelola, basecamp, new selo]
kata yang masuk ke bahasa : [siapa, pengelola, basecamp]
kata yang masuk ke non-bahasa : [new selo]
waktu untuk validasi = 18

```

Gambar 5.32 Cuplikan hasil proses validasi

Kata yang valid di dalam *Method ValidateInput* disusun ke dalam dua kategori, yaitu kata yang termasuk ke dalam kategori bahasa dan kata yang termasuk ke dalam kategori non-bahasa (Gambar 5.30 baris 40-51). Kata yang termasuk ke dalam kategori bahasa merupakan kata yang terdapat didalam ontologi Bahasa, sedangkan kata yang termasuk ke dalam kategori non-bahasa merupakan kata yang tidak terdapat didalam ontologi Bahasa. Pembagian kategori berfungsi dalam proses pengambilan pengetahuan bahasa (atribut kata) untuk tiap kata, yang nantinya dibutuhkan untuk proses pengecekan sintaksis dan semantik menggunakan aturan-aturan gramatikal.

Untuk mengambil atribut dari kata yang termasuk kategori bahasa, *Method ValidateInput* menggunakan *method createSimpleQuery* untuk mengambil *query*, kemudian menggunakan *method runRawQuery* untuk mengeksekusi *query* dan menyimpan hasil eksekusi ke dalam *list* menggunakan *method simpleResultSetToListBahasa*. Implementasi pengambilan atribut dari kata dapat dilihat pada Gambar 5.30 baris 43-46. Cuplikan kode dari ketiga *method* tersebut dapat dilihat pada Gambar 5.33.

```

1. public String createSimpleQuery(String pertanyaan) {
2.     String query = PREFIX_QUERY + " SELECT DISTINCT ?" + SUBJECT + " ?"
3.             + NILAI + " ?" + KATEGORI + " ?" + NILAI_SEMANTIK + " ?"
4.             + TIPE_SEMANTIK + " WHERE { ?" + SUBJECT + " bhs:"
5.             + NILAI + " ?" + NILAI + " ." + " ?" + SUBJECT
6.             + " bhs:" + KATEGORI + " ?" + KATEGORI + " ." + " ?"
7.             + SUBJECT + " bhs:" + NILAI_SEMANTIK + " ?"
8.             + NILAI_SEMANTIK + " ." + " ?" + SUBJECT + " bhs:"
9.             + TIPE_SEMANTIK + " ?" + TIPE_SEMANTIK + " ."
10.            + " FILTER ( ?" + NILAI + "= \" + pertanyaan
11.            + "\"^^xsd:string) ." + " }";
12.    return query;
13. }

```

(a)

```

1. public ResultSet runRawQuery(String queryString) {
2.     Query query = QueryFactory.create(queryString);
3.     QueryExecution qe = QueryExecutionFactory.create(query, model1);
4.     ResultSet results = qe.execSelect();
5.     return results;
6. }

```

(b)

```

1. public List<OntoBahasa> simpleResultSetToListBahasa(ResultSet rs) {
2.     List<OntoBahasa> bahasas = new ArrayList<OntoBahasa>();
3.     while (rs.hasNext()) {
4.         QuerySolution qs = rs.next();
5.         OntoBahasa ob = new OntoBahasa(this);
6.         ob.setSubject(qs.get(SUBJECT).asNode().getLocalName());
7.         ob.setNilai(qs.get(NILAI).asNode().getLiteralValue().toString());
8.         ob.setNilaiSemantik(qs.get(NILAI_SEMANTIK).asNode().getLocalName());
9.         ob.setTipeSemantik(qs.get(TIPE_SEMANTIK).asNode().getLocalName());
10.        for (OntoBahasa.KategoriSintaks syntaks : OntoBahasa.KategoriSintaks
11.                .values()) {
12.                    if (qs.get(KATEGORI).asNode().getLocalName()
13.                            .equalsIgnoreCase(syntaks.name())) {
14.                        ob.setKategori(syntaks);
15.                        break;
16.                    }
17.                }
18.            bahasas.add(ob);
19.        }
20.    return bahasas;
21. }

```

(c)

Gambar 5.33 (a) *method createSimpleQuery*. (b) *runRawQuery*. (c) *simpleResultSetToListBahasa*

Kata valid yang termasuk ke dalam kategori non-bahasa yang dideskripsikan pada penelitian ini adalah kata yang ditemukan di dalam ontologi *Mountaineering*. Kata valid dari ontologi *Mountaineering* merupakan kosakata yang tidak memiliki atribut kata. Oleh karena itu untuk memberikan atribut kata pada kata-kata tersebut telah disiapkan *instance* dari *subclass Temp* yang terdapat pada ontologi Bahasa. Untuk mengambil atribut kata dari *subclass Temp* dan memberikan atribut tersebut pada kata non-bahasa menggunakan *method*

queryTemplate (Gambar 5.30 baris 49). Cuplikan kode dari *method queryTemplate* dapat dilihat pada Gambar 5.34.

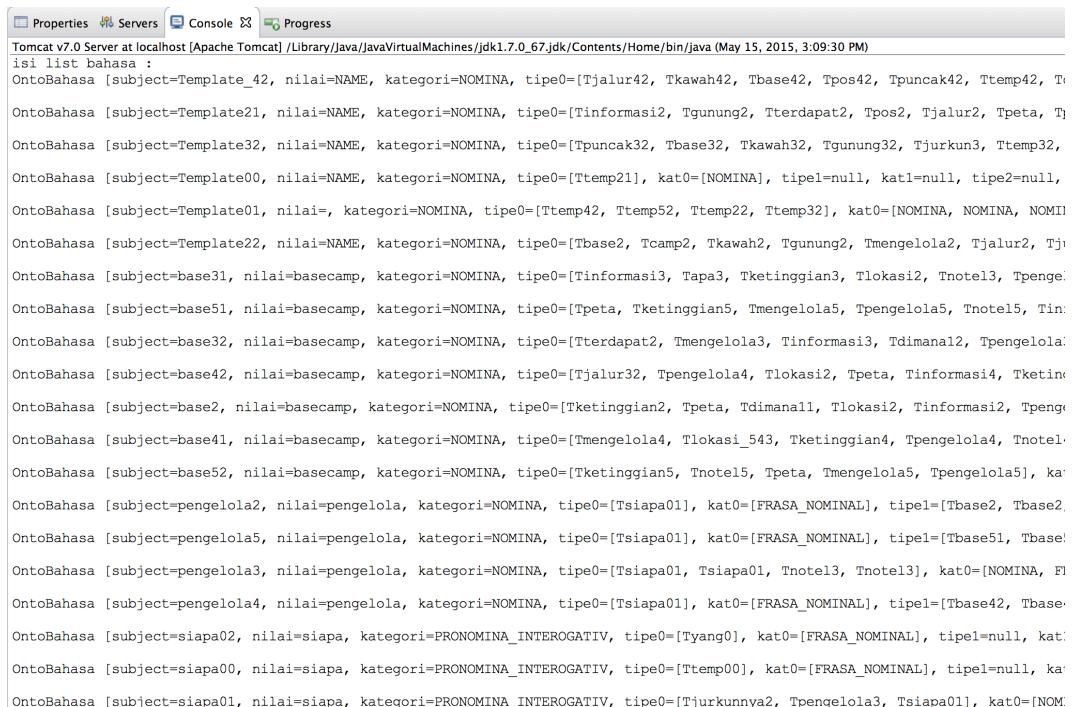
```

1.   public List<OntoBahasa> queryTemplate(String kata) {
2.       String query = createQueryTemplate("Template");
3.       ResultSet rs = runRawQuery(query);
4.       List<OntoBahasa> bahasas = simpleResultSetToListBahasa(rs);
5.       for (OntoBahasa bahasa : bahasas){
6.           bahasa.setShadowNilai(kata);
7.       }
8.       return bahasas;
9.   }

```

Gambar 5.34 Method *queryTemplate*

Setiap kata valid yang telah memiliki atribut kemudian disimpan ke dalam satu buah *list*. Cuplikan *list* hasil *preprocessing* menggunakan *input* "siapa pengelola basecamp new selo" dapat dilihat pada Gambar 5.35.



```

Properties Servers Console Progress
Tomcat v7.0 Server at localhost [Apache Tomcat] /Library/java/javaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (May 15, 2015, 3:09:30 PM)
isi list bahasa :
OntoBahasa [subject=Template_42, nilai=NAME, kategori=NOMINA, tipe0=[Tjalur42, Tkawah42, Tbase42, Tpos42, Tpuncak42, Ttemp42, Tj
OntoBahasa [subject=Template21, nilai=NAME, kategori=NOMINA, tipe0=[Tinformasi2, Tgunung2, Tterdapat2, Tpos2, Tjalur2, Tpeta, Tj
OntoBahasa [subject=Template32, nilai=NAME, kategori=NOMINA, tipe0=[Tpuncak32, Tbase32, Tkawah32, Tgunung32, Tjurkun3, Ttemp32,
OntoBahasa [subject=Template00, nilai=NAME, kategori=NOMINA, tipe0=[Ttemp21], kat0=[NOMINA], tipe1=null, kat1=null, tipe2=null,
OntoBahasa [subject=Template01, nilai=, kategori=NOMINA, tipe0=[Ttemp42, Ttemp52, Ttemp22, Ttemp32], kat0=[NOMINA, NOMINA, NOMI
OntoBahasa [subject=Template22, nilai=NAME, kategori=NOMINA, tipe0=[Tbase2, Tcamp2, Tkawah2, Tgunung2, Tmengelola2, Tjalur2, Tj
OntoBahasa [subject=base31, nilai=basecamp, kategori=NOMINA, tipe0=[Tinformasi3, Tapa3, Tketinggian3, Tlokasi2, Tnote3, Tpenge
OntoBahasa [subject=base51, nilai=basecamp, kategori=NOMINA, tipe0=[Tpeta, Tketinggian5, Tmengelola5, Tpengelola5, Tnote5, Tin
OntoBahasa [subject=base32, nilai=basecamp, kategori=NOMINA, tipe0=[Tterdapat2, Tmengelola3, Tinformasi3, Tdimana12, Tpengelola
OntoBahasa [subject=base42, nilai=basecamp, kategori=NOMINA, tipe0=[Tjalur32, Tpengelola4, Tlokasi2, Tpeta, Tinformasi4, Tketing
OntoBahasa [subject=base2, nilai=basecamp, kategori=NOMINA, tipe0=[Tketinggian2, Tpeta, Tdimana11, Tlokasi2, Tinformasi2, Tpeng
OntoBahasa [subject=base41, nilai=basecamp, kategori=NOMINA, tipe0=[Tmengelola4, Tlokasi_543, Tketinggian4, Tpengelola4, Tnote
OntoBahasa [subject=base52, nilai=basecamp, kategori=NOMINA, tipe0=[Tketinggian5, Tnote5, Tpeta, Tmengelola5, Tpengelola5], ka
OntoBahasa [subject=pengelola2, nilai=pengelola, kategori=NOMINA, tipe0=[Tsiapa01], kat0=[FRASA_NOMINAL], tipe1=[Tbase2, Tbase2
OntoBahasa [subject=pengelola5, nilai=pengelola, kategori=NOMINA, tipe0=[Tsiapa01], kat0=[FRASA_NOMINAL], tipe1=[Tbase51, Tbase
OntoBahasa [subject=pengelola3, nilai=pengelola, kategori=NOMINA, tipe0=[Tsiapa01, Tsiapa01, Tnote3, Tnote3], kat0=[NOMINA, F
OntoBahasa [subject=pengelola4, nilai=pengelola, kategori=NOMINA, tipe0=[Tsiapa01], kat0=[FRASA_NOMINAL], tipe1=[Tbase42, Tbase
OntoBahasa [subject=siapa02, nilai=siapa, kategori=PRONOMINA_INTERROGATIV, tipe0=[Tyang0], kat0=[FRASA_NOMINAL], tipe1=null, kat
OntoBahasa [subject=siapa00, nilai=siapa, kategori=PRONOMINA_INTERROGATIV, tipe0=[Ttemp00], kat0=[FRASA_NOMINAL], tipe1=null, kat
OntoBahasa [subject=siapa01, nilai=siapa, kategori=PRONOMINA_INTERROGATIV, tipe0=[Tjurkunny2, Tpengelola3, Tsiapa01], kat0=[NOM

```

Gambar 5.35 Cuplikan *list* hasil *preprocessing*

Hasil akhir dari *preprocessing* tidak selalu menghasilkan *list*, apabila pada *input* terdapat kata yang tidak dikenali maka *preprocessor* akan mengirimkan nilai balik kepada *method doGet* bahwa *input* tidak valid dan akan men-generate pesan kesalahan bertipe *ERROR_INPUT* kepada halaman *tampiltoken.jsp* (Gambar 5.25 baris 17-18). Misalnya pada *input* "siapa pengelola basecamp semeru" pada *input* tersebut kata "semeru" tidak dikenali, cuplikan hasil *preprocessing* *input* yang

mengandung kata yang tidak dikenali dapat dilihat pada Gambar 5.37. Cuplikan kode untuk *ERROR_INPUT* dapat dilihat pada Gambar 5.36.

```

1. public String generateResponseMessage(ResponseType type, String kalimat) {
2.     String pesan = "";
3.     switch (type) {
4.         case ERROR_INPUT:
5.             if (listKataTidakKetemu.size() > 0) {
6.                 pesan += "Maaf, Sistem tidak dapat memproses input karena pada
7.                 " + kalimat + "' ";
8.                 pesan += "kata ";
9.                 for (int i = 0; i < listKataTidakKetemu.size(); i++) {
10.                     if (i != 0) {
11.                         if (i < listKataTidakKetemu.size() - 1) {
12.                             pesan += ", ";
13.                         } else {
14.                             pesan += " dan ";
15.                         }
16.                         pesan += '<b>' + listKataTidakKetemu.get(i) + "</b>'";
17.                     }
18.                     pesan += " tidak dikenali oleh sistem, silahkan ulangi
19.                     pencarian dan pastikan input-an anda sudah benar";
20.                     break;
21.                 case ERROR_SYNTAX:
22.                     pesan += "Maaf, Sistem tidak dapat memahami input, karena
23.                     berdasarkan aturan sintaksis dan simantik yang dimiliki sistem
24.                     input tidak dapat diproses, silahkan ulangi pencarian" + info;
25.                     break;
26.             }
27.             System.out.println("Pesanan akhir " + pesan);
28.             return pesan;
29.     }
30. }
```

Gambar 5.36 Method generateResponseMessage

```

Properties Servers Console Progress
Tomcat v7.0 Server at localhost [Apache Tomcat] /Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (May 15, 2015, 3:09:30 PM)
Waktu konstruksi preproses = 136
wordtocek : siapa
wordtocek : siapa pengelola
wordtocek : siapa pengelola basecamp
wordtocek : siapa pengelola basecamp semeru
wordtocek : pengelola
wordtocek : pengelola basecamp
wordtocek : pengelola basecamp semeru
wordtocek : basecamp
wordtocek : basecamp semeru
wordtocek : semeru
kata tidak ketemu : [sembru]
kata ketemu : [siapa, pengelola, basecamp]
kata yang masuk ke bahasa : [siapa, pengelola, basecamp]
kata yang masuk ke non-bahasa : []
waktu untuk validasi = 8
Pesanan akhir
Maaf, Sistem tidak dapat memproses input karena pada 'siapa pengelola basecamp semeru' kata '<b>sembru</b>' tidak dikenali oleh sistem

```

Gambar 5.37 Cuplikan hasil preprocessing untuk input yang tidak dikenali

5.3.3 Pembentukan *parse tree* dengan pengecekan aturan gramatikal

List berisi urutan kata beserta atribut yang telah dibentuk oleh *preprocessing* dikirimkan kembali kepada *method doGet* untuk dilanjutkan kepada *method cekSintaksis*. *Method cekSintaksis* bertugas melakukan inisialisasi untuk proses pengecekan sintaksis dan semantik. Cuplikan kode dari *method cekSintaksis* dapat dilihat pada Gambar 5.38.

```

1. public boolean cekSintaksis(String kalimat, List<OntoBahasa> resolusi) {
2.     long start = System.currentTimeMillis();
3.     //other line of method
11.    cekOntoGabung(resolusi);
12.    long end = System.currentTimeMillis();
13.    System.out.println("waktu diperlukan cek sintaksis = " + (end -
14.        start));
14.    if (resolusi.size() > 0) {
15.        return true;
16.    }
17.    return false;
18. }

```

Gambar 5.38 Method cekSintaksis

Proses pengecekan sintaksis dan semantik dilakukan oleh *method cekOntoGabung* (Gambar 5.38 baris 11) dengan membentuk urutan kata menjadi *parse tree* menggunakan aturan-aturan gramatikal selama proses pengecekan. Cuplikan kode dari *method cekOntoGabung* dapat dilihat pada Gambar 5.39.

Proses pengecekan dimulai dari posisi urutan kata yang paling kanan di dalam *list*, kemudian dicocokkan dengan kata yang berada di sebelah kiri kata tersebut. Misalkan pada urutan kata [*siapa, pengelola, basecamp, new selo*] proses pengecekan dimulai dari kata [*new selo*] kemudian dicocokkan dengan kata disebelah kirinya yaitu [*basecamp*]. Proses pengecekan sintaksis dilakukan oleh *method isValidSintax* yang dipanggil dari *method cekOntoGabung* yang pemanggilannya dapat dilihat pada Gambar 5.39 baris 41. Cuplikan kode dari *method isValidSintax* dapat dilihat pada Gambar 5.40.

Proses pengecekan oleh *Method isValidSintax* dimulai dengan mencocokkan kategori dari masing-masing kata. Misalkan pada kata [*new selo*] dan [*basecamp*] memiliki kategori *NOMINA* dan *NOMINA*, maka *method isValidSintax* akan memanggil suatu *method* untuk melakukan pengecekan aturan dari kedua kategori tersebut. Aturan gramatikal yang digunakan untuk pengecekan kategori *NOMINA* dan *NOMINA* dikerjakan oleh *method checkLeftFN*. Cuplikan kode dari *method checkLeftFN* dapat dilihat pada Gambar 5.41.

```

1.  private void cekOntoGabung(List<OntoBahasa> solusi) {
2.      Stack<List<OntoBahasa>> stacks = new Stack<List<OntoBahasa>>();
3.      int j = 0;
4.      while (j < bahasas.size()) {
5.          int i = j;
6.          boolean isAdditionalSkip = false;
7.          if (solusi.isEmpty()) {
8.              if (i < bahasas.size() - 1) {
9.                  isAdditionalSkip = true;
10.                 solusi.addAll(bahasas.get(i));
11.                 i++;
12.             } else {
13.                 break;
14.             }
15.         }
16.         List<OntoBahasa> baselist = new ArrayList<OntoBahasa>(solusi);
17.         List<OntoBahasa> list2 = bahasas.get(i);
18.         List<OntoBahasa> list1 = null;
19.         if (i < bahasas.size() - 1) {
20.             list1 = bahasas.get(i + 1);
21.         }
22.         solusi.clear();
23.         boolean ketemu = false;
24.         for (OntoBahasa b3 : baselist) {
25.             for (OntoBahasa b2 : list2) {
26.                 if (list1 != null) {
27.                     for (OntoBahasa b1 : list1) { //Start cek 3 token-----
28.                         if (SyntaxChecker.isValidSyntax(b1, b2, b3)) {
29.                             ketemu = true;
30.                             OntoBahasa hasil =
31.                                 SyntaxChecker.generateAccepted(b1, b2, b3);
32.                             solusi.add(hasil);
33.                             j++;
34.                             j++;
35.                             if (isAdditionalSkip) {
36.                                 j++;
37.                             }
38.                         }
39.                     }
40.                     if (!ketemu) { // Start cek 2 token-----
41.                         if (SyntaxChecker.isValidSyntax(b2, b3)) {
42.                             ketemu = true;
43.                             OntoBahasa hasil=SyntaxChecker.generateAccepted(b2, b3);
44.                             solusi.add(hasil);
45.                             j++;
46.                             if (isAdditionalSkip) {
47.                                 j++;
48.                             }
49.                         }
50.                     }
51.                 }
52.             }
53.             if (!ketemu) { // masukan ke stack
54.                 stacks.push(baselists);
55.                 if (isAdditionalSkip) {
56.                     j++;
57.                 }
58.             }
59.         }
60.         checkStack(stacks, solusi);
61.     }

```

Gambar 5.39 Pembentukan Parse Tree

```

1.  public static boolean isValidSyntax(OntoBahasa b1, OntoBahasa b2) {
2.      if (isFN(b1)) {
3.          return checkLeftFN(b1, b2);
4.      } else if (isFV(b1)){
5.          return checkLeftFV(b1, b2);
6.      } else if (isFFPrep(b1)) {
7.          return checkLeftFPrep(b1, b2);
8.      } else if (b1.getKategori().equals(OntoBahasa.KategoriSintaks.KLAUSA)) {
9.          return checkLeftKlausa(b1,b2);
10.     } else if (b1.getKategori().equals(OntoBahasa.KategoriSintaks.KALIMAT)) {
11.         return checkLeftKalimat(b1,b2);
12.     } else if
13.         (b1.getKategori().equals(OntoBahasa.KategoriSintaks.ARTIKULA)) {
14.             return checkLeftArt(b1, b2);
15.         }
16.     return false;
}

```

Gambar 5.40 Cuplikan kode dari *Method siValidSintax*

```

1.  private static boolean checkLeftFN(OntoBahasa b1, OntoBahasa b2) {
2.      boolean accepted = false;
3.      if // ... kode untuk aturan gramatikal lain
4.
5.      if (!accepted && isFN(b2)) {
6.          accepted = (
7.              b1.getKat1() != null &&
8.              b1.getKat1().contains(OntoBahasa.KategoriSintaks.FRASA_NOMINAL) &&
9.              b1.getTipe1() != null &&
10.                 b1.getTipe1().contains(b2.getTipeSemantik()) &&
11.                     b2.getKat0() != null &&
12.                     b2.getKat0().contains(OntoBahasa.KategoriSintaks.NOMINA) &&
13.                     b2.getTipe0() != null &&
14.                     b2.getTipe0().contains(b1.getTipeSemantik()))
15.      );
16.
17.      if // ... kode untuk aturan gramatikal lain
18.      return accepted;
19.  }

```

(a)

```

FN0 -> N FN1
        <FN0 arg0> = <N arg0>
        <FN0 arg1> = 0
        <FN0 arg2> = <N arg2>
        <FN0 strip> = <FN1 strip>

        <N argCat1> = <FN1 cat>
        <N argType1> = <FN1 type>
        <N strip> = 0

        <FN1 argCat0> = <N cat>
        <FN1 argType0> = <N type>
        <FN1 arg1> = 0

```

(b)

Gambar 5.41 (a) *Method checkLeftFN* implementasi aturan gramatikal pembentuk *FRASA_NOMINAL*. (b) Aturan gramatikal gramatikal pembentuk *FRASA_NOMINAL*

Method checkLeftFN pada Gambar 5.41 (a) merupakan implementasi dari aturan gramatikal pada Gambar 5.41 (b) yang mana aturan tersebut digunakan untuk pengecekan dua kata yang membentuk *FRASA_NOMINAL*. Aturan gramatikal pada Gambar 5.41 (b) merupakan penjabaran dari tata bahasa Indonesia yang telah dijabarkan oleh Alwi (2003).

Berdasarkan aturan gramatikal pada Gambar 5.41, kata kanan [*new selo*] dan kata kiri [*basecamp*] dapat diterima sebagai pembentuk *FRASA_NOMINAL* jika [*basecamp*] berkategori *NOMINA* dan [*new selo*] berkategori *NOMINA/FRASA_NOMINAL*. Kemudian kata [*basecamp*] dan kata [*new selo*] dapat diterima secara sintaksis dan semantik jika atribut pada kata [*basecamp*], antara lain:

- nilai *argCat1* tidak *null*,
- nilai *argCat1* mengandung *FRASA_NOMINAL*,
- nilai *argType1* tidak *null*,
- nilai *argType1* mengandung nilai *Type* dari kata [*new selo*]

sedangkan atribut pada kata [*new selo*], antara lain:

- nilai *argCat0* tidak *null*
- nilai *argCat0* mengandung *NOMINA*
- nilai *argType0* tidak *null*
- nilai *argType0* mengandung nilai *Type* dari kata [*basecamp*]

Apabila kata [*basecamp*] dan [*new selo*] diterima berdasarkan pengecekan aturan gramatikal maka kedua kata tersebut digabungkan menjadi satuan kata yang baru yaitu [*basecamp new selo*] dengan kategori *FRASA_NOMINAL*.

Proses penggabungan kata menjadi satuan kata yang baru dikerjakan oleh *method GeneratedAccepted*. *Method GeneraterAccepted* melakukan penggabungan *kata* berdasarkan hasil pengecekan pada Gambar 5.41. Selain menggabungkan kata menjadi satuan kata, *method generatedAccepted* juga membentuk atribut baru untuk satuan kata tersebut. Cuplikan kode dari *method GeneratedAccepted* untuk menangani penggabungan kata menjadi satuan *FRASA_NOMINAL* dapat dilihat pada Gambar 5.42.

```

1. public static OntoBahasa generateAccepted(OntoBahasa a, OntoBahasa b) {
2.     OntoBahasa hasil = new OntoBahasa(a.getReader());
3.     if () {
4.         ... // statement penggabungan untuk aturan lain
5.     }
6.     else if ((a.getKategori().equals(OntoBahasa.KategoriSintaks.NOMINA)
7.             || a.getKategori().equals(OntoBahasa.KategoriSintaks.NUMERALIA)) &&
8.             (isFN(b))) {
9.         String pertama = a.getNilai();
10.        String kedua = b.getNilai();
11.        if (a.getShadowNilai() != null) {
12.            pertama = a.getShadowNilai();
13.        }
14.        if (b.getShadowNilai() != null) {
15.            kedua = b.getShadowNilai();
16.        }
17.        hasil.setNilai(pertama + " " + kedua);
18.        hasil.setKategori(OntoBahasa.KategoriSintaks.FRASA_NOMINAL);
19.        hasil.setTipeSemantik(a.getTipeSemantik());
20.        hasil.addVars(a.getVars());
21.        hasil.addVars(b.getVars());
22.        if (a.getNilaiSemantik() != null) {
23.            String queryLogic = a.getReader().getQueryLogic(a);
24.            hasil.addVars(queryLogic);
25.        }
26.        if (b.getNilaiSemantik() != null) {
27.            String queryLogic = a.getReader().getQueryLogic(b);
28.            hasil.addVars(queryLogic);
29.        }
30.        hasil.setKat0(a.getKat0());
31.        hasil.setTipe0(a.getTipe0());
32.        hasil.setKat1(null);
33.        hasil.setTipe1(null);
34.        hasil.setKat2(a.getKat2());
35.        hasil.setTipe2(a.getTipe2());
36.        hasil.setKatS(b.getKatS());
37.        hasil.setTipeS(b.getTipeS());
38.    }
39.    else if () {
40.        ... // statement penggabungan untuk aturan lain
41.    }
42.    return hasil;
43. }

```

(a)

```

FN0 -> N FN1
<FN0 arg0> = <N arg0>
<FN0 arg1> = 0
<FN0 arg2> = <N arg2>
<FN0 strip> = <FN1 strip>

<N argCat1> = <FN1 cat>
<N argType1> = <FN1 type>
<N strip> = 0

<FN1 argCat0> = <N cat>
<FN1 argType0> = <N type>
<FN1 arg1> = 0

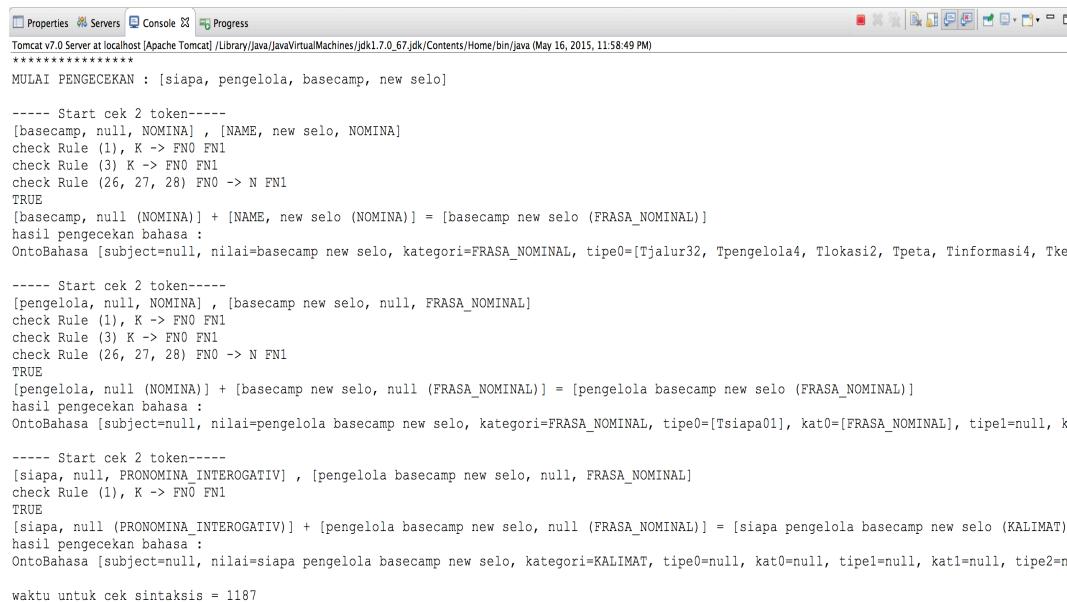
```

(b)

Gambar 5.42 (a) *Method generatedAccepted* untuk menggabungkan dua kata yang membentuk *FRASA_NOMINAL*. (b) aturan gramatikal penggabungan dua kata yang membentuk *FRASA_NOMINAL*

Atribut untuk satuan kata [*basecamp new selo*] dibentuk berdasarkan aturan gramatikal pada Gambar 5.42(b). Berdasarkan aturan gramatikal tersebut, maka nilai atribut untuk *FRASA_NOMINAL* [*basecamp new selo*] yaitu *argCat0*, *argType0*, *argCat2* dan *argType2* menggunakan nilai dari kata [*basecamp*], atribut *argCat1* dan *argType1* bernilai *null*, atribut *stripCat* dan *stripType* menggunakan nilai dari kata [*new selo*].

Setelah proses pengecekan, satuan kata [*basecamp new selo*] akan dikembalikan ke dalam *list* urutan kata, sehingga urutan kata setelah proses pengecekan menjadi [*siapa, pengelola, basecamp new selo*]. *Method cekOntoGabung* akan melanjutkan proses pengecekan antara [*basecamp new selo*] dengan kata [*pengelola*] mengikuti langkah yang sama seperti yang telah dijelaskan sebelumnya. Proses pengecekan akan berhenti ketika mencapai kata paling kiri pada urutan kata, yaitu kata [*siapa*]. Cuplikan hasil proses pengecekan urutan menggunakan aturan gramatikal dapat dilihat pada Gambar 5.43. Ilustrasi proses pengecekan kata dapat dilihat pada Gambar 5.44.



```

Properties Servers Console Progress
Tomcat v7.0 Server at localhost [Apache Tomcat] /Library/java/Jdk1.7.0_67/jdk/Contents/Home/bin/java (May 16, 2015, 11:58:49 PM)
*****
MULAI PENGECEKAN : {siapa, pengelola, basecamp, new selo}

----- Start cek 2 token-----
[basecamp, null, NOMINA] , [NAME, new selo, NOMINA]
check Rule (1), K -> FNO FN1
check Rule (3) K -> FNO FN1
check Rule (26, 27, 28) FNO -> N FN1
TRUE
[basecamp, null (NOMINA)] + [NAME, new selo (NOMINA)] = [basecamp new selo (FRASA_NOMINAL)]
hasil pengecekan bahasa :
OntoBahasa [subject=null, nilai=basecamp new selo, kategori=FRASA_NOMINAL, tipe0=[Tjalur32, Tpengelola4, Tlokasi2, Tpeta, Tinformati4, Tke

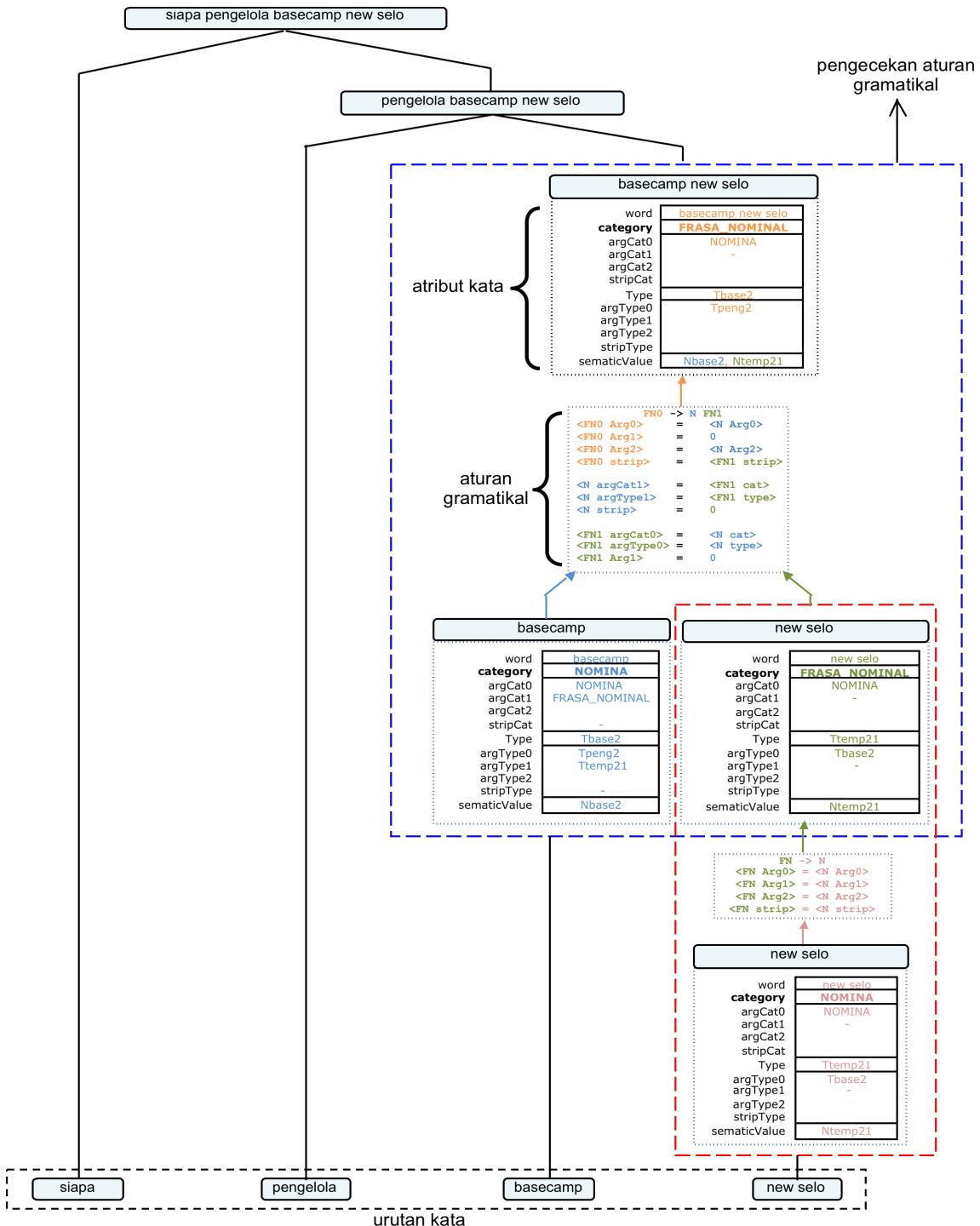
----- Start cek 2 token-----
[pengelola, null, NOMINA] , [basecamp new selo, null, FRASA_NOMINAL]
check Rule (1), K -> FNO FN1
check Rule (3) K -> FNO FN1
check Rule (26, 27, 28) FNO -> N FN1
TRUE
[pengelola, null (NOMINA)] + [basecamp new selo, null (FRASA_NOMINAL)] = [pengelola basecamp new selo (FRASA_NOMINAL)]
hasil pengecekan bahasa :
OntoBahasa [subject=null, nilai=pengelola basecamp new selo, kategori=FRASA_NOMINAL, tipe0=[Tsiapa01], kat0=[FRASA_NOMINAL], tipe1=null, k

----- Start cek 2 token-----
[siapa, null, PRONOMINA_INTEROGATIV] , [pengelola basecamp new selo, null, FRASA_NOMINAL]
check Rule (1), K -> FNO FN1
TRUE
[siapa, null (PRONOMINA_INTEROGATIV)] + [pengelola basecamp new selo, null (FRASA_NOMINAL)] = [siapa pengelola basecamp new selo (KALIMAT)
hasil pengecekan bahasa :
OntoBahasa [subject=null, nilai=siapa pengelola basecamp new selo, kategori=KALIMAT, tipe0=null, kat0=null, tipe1=null, kat1=null, tipe2=n

waktu untuk cek sintaksis = 1187

```

Gambar 5.43 Cuplikan hasil proses pengecekan urutan kata



Gambar 5.44 Ilustrasi proses pengecekan menggunakan aturan gramatikal

Apabila pada proses pengecekan aturan gramatikal, dua kata (kanan dan kiri) tidak dapat diterima dan di dalam urutan kata masih berisi kata, maka kata kanan akan dimasukkan ke dalam *stack* (Gambar 5.39 baris 53-58). Pengecekan

dilanjutkan kembali dari kata yang berada disebelah kiri dari kata yang dimasukkan ke dalam *stack*. Hingga proses pengecekan mencapai kata terakhir dari urutan kata, kata yang ada di dalam *stack* baru dikeluarkan untuk dilakukan pengecekan kembali. Proses pengecekan ke dalam *stack* dilakukan oleh *method CheckStack*. Cuplikan kode untuk *method checkStack* dapat dilihat pada Gambar 5.45.

```

1.  private void checkStack(Stack<List<OntoBahasa>> stacks, List<OntoBahasa>
2.      solusi) {
3.      if (solusi.isEmpty()) {
4.          return;
5.      }
6.      List<OntoBahasa> baselist, list2 = null, list1;
7.      while (!stacks.isEmpty()) {
8.          list1 = list2;
9.          baselist = new ArrayList<OntoBahasa>(solusi);
10.         list2 = stacks.pop();
11.         solusi.clear();
12.         boolean ketemu = false;
13.         for (OntoBahasa b1 : baselist) {
14.             for (OntoBahasa b2 : list2) {
15.                 if (list1 != null) {
16.                     for (OntoBahasa b3 : list1) {
17.                         if (SyntaxChecker.isValidSyntax(b1, b2, b3)) {
18.                             ketemu = true;
19.                             OntoBahasa hasil=SyntaxChecker.generateAccepted(b1,
20.                                     b2, b3);
21.                             solusi.add(hasil);
22.                         }
23.                     }
24.                     if (!ketemu) {
25.                         if (SyntaxChecker.isValidSyntax(b1, b2)) {
26.                             ketemu = true;
27.                             OntoBahasa hasil = SyntaxChecker.generateAccepted(
28.                                 b1, b2);
29.                             solusi.add(hasil);
30.                         }
31.                     }
32.                     if (ketemu) {
33.                         list2 = null;
34.                     } else {
35.                         if (list1 != null) {
36.                             return;
37.                         }
38.                     }
39.                 }
40.             }

```

Gambar 5.45 method checkStack

Misalnya pada urutan kata [*siapa pengelola basecamp new selo*], kata [*new selo*] tidak memenuhi aturan gramatikal ketika dilakukan pengecekan dengan kata [*basecamp*], maka kata [*new selo*] akan dimasukkan ke dalam *stack*. kemudian pengecekan dilanjutkan pada kata [*basecamp*] dengan kata [*pengelola*] menjadi [*pengelola basecamp*] dan pengecekan dilanjutkan lagi pada [*siapa*],

sehingga menjadi [*siapa pengelola basecamp*]. Setelah semua kata dalam urutan kata selesai dicek barulah kata [*new selo*] dikeluarkan dari dalam *stack* untuk dilakukan pengecekan dengan kata [*siapa pengelola basecamp*]. Apabila kata [*basecamp*] masih tidak dapat diterima sebagai pembentuk satuan bahasa menggunakan aturan gramatikal, maka *cekSintaksis* akan mengirimkan nilai balik kepada *method doGet* dan akan men-generate pesan kesalahan bertipe *ERROR_SYNTAX* (Gambar 5.36 baris 20-22) ke halaman *tampiltoken.jsp*. Cuplikan kode dari *method doGet* untuk mengirimkan pesan *ERROR_SYNTAX* dapat dilihat pada Gambar 5.46.

```

1.     boolean sitaksValid = preprocessor.cekSintaksis(resolusi);
2.     if (!sitaksValid) {
3.         request.setAttribute("input", kalimat);
4.         request.setAttribute("output", preprocessor.generateResponseMessage(
5.             Preprocessor.ResponseType.ERROR_SYNTAX, kalimat));
6.         request.getRequestDispatcher("/tampiltoken.jsp").forward(request,
7.             response);
8.     }

```

Gambar 5.46 Cuplikan kode untuk mengirimkan pesan bertipe *ERROR_SYNTAX* kepada halaman *tampiltoken.jsp*

5.3.4 Ekstraksi informasi semantik

Proses ekstraksi informasi semantik dari kata menjadi bentuk *query SPARQL* didasarkan pada nilai dari *property semanticValue* yang dimiliki tiap kata. Nilai dari *property semanticValue* diambil pada saat proses pengecekan aturan gramatikal oleh *method generatedAccepted* (Gambar 5.42 baris 9 - 17). Proses pengambilan nilai dari *property semanticValue* hanya dilakukan terhadap kata yang dapat diterima sebagai penyusun bahasa.

Informasi semantik dalam bentuk *statement query SPARQL* diambil dari *class QueryLogic* pada ontologi Bahasa. Setiap *instance* pada *class QueryLogic* memiliki *property qpart*. *Property qpart* berisi potongan-potongan *query SPARQL* yang mendeskripsikan konsep makna yang dimiliki oleh kata. *Property qpart* dari *instance* pada *class QueryLogic* terhubung melalui *property para* yang mengandung nilai dari *property semanticValue* pada kata. Pengambilan potongan *query SPARQL* dari *class QueryLogic* dengan menggunakan *property para* dilakukan oleh *method getQueryLogic*. Cuplikan kode dari *method getQueryLogic* dapat dilihat pada Gambar 5.47.

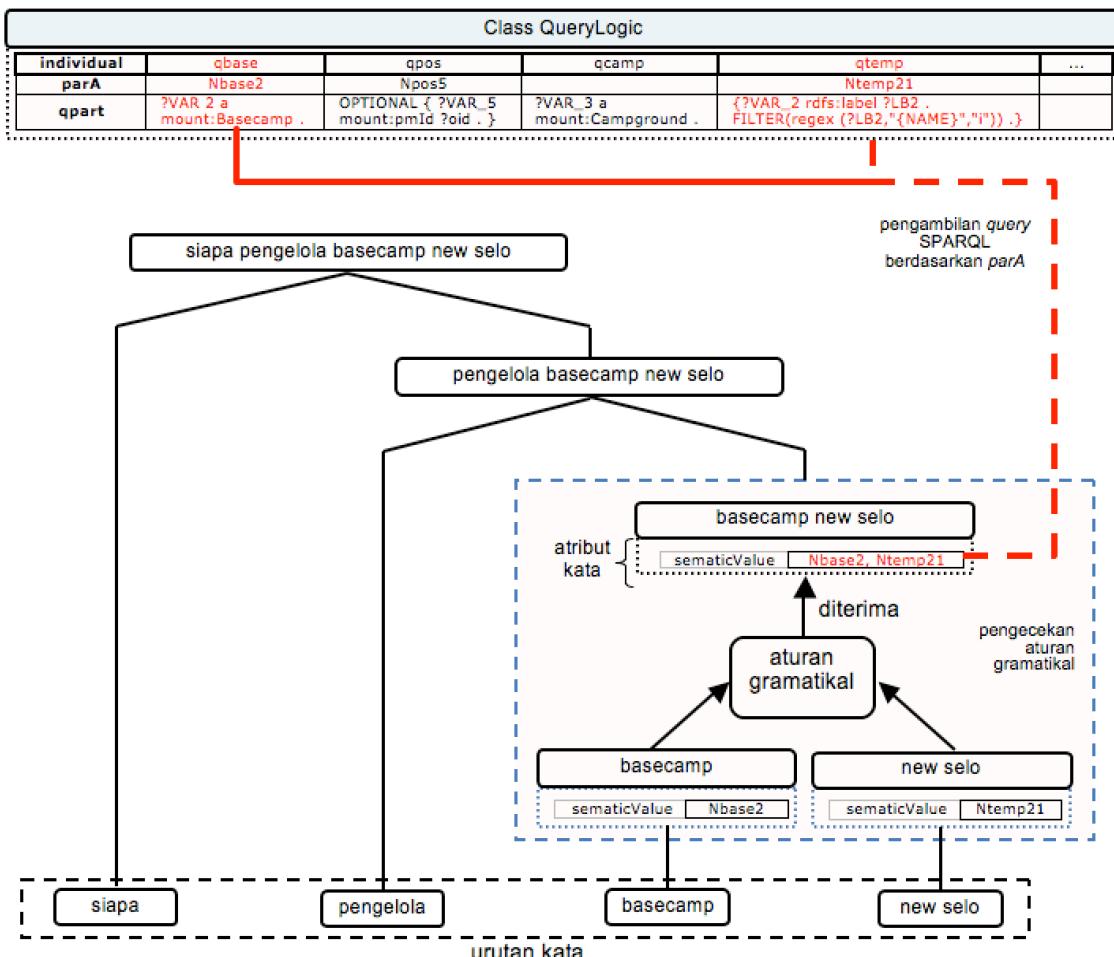
```

1.   public String getQueryLogic(OntoBahasa bahasa) {
2.       String hasil = "";
3.       ResultSet rs = runRawQuery(createQueryLogicQueryOnParA(bahasa
4.           .getNilaiSemantik()));
5.       while (rs.hasNext()) {
6.           QuerySolution qs = rs.next();
7.           String q = qs.get(QPART).asNode().getLiteralValue().toString();
8.           if (q.contains("{NAME}")) {
9.               String shadow = bahasa.getShadowNilai();
10.              shadow = "" +
11.                  shadow.charAt(0).toUpperCase()
12.                  + shadow.substring(1);
13.              q = q.replace("{NAME}", shadow);
14.          }
15.      }
16.      return hasil;
17.  }

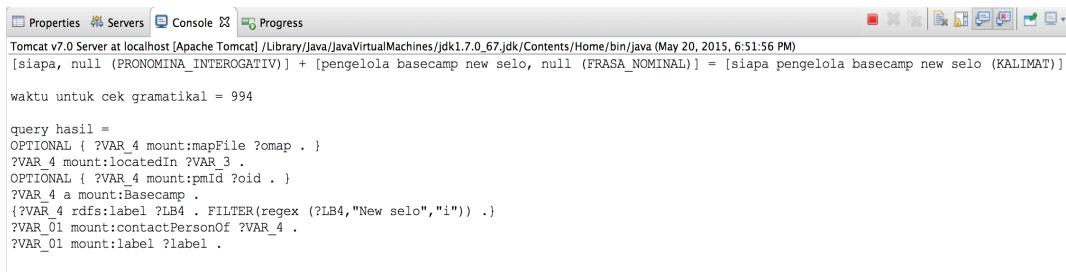
```

Gambar 5.47 method *getQueryLogic*

Ilustrasi pembentukan *query* SPARQL oleh *method getQueryLogic* dijabarkan pada Gambar 5.48. Sedangkan cuplikan hasil proses ekstraksi informasi ke dalam *query* SPARQL dapat dilihat pada Gambar 5.49.



Gambar 5.48 Ilustrasi pembentukan *query* SPARQL oleh *method getQueryLogic*



```

Properties Servers Console 23 Progress
Tomcat v7.0 Server at localhost [Apache Tomcat] /Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (May 20, 2015, 6:51:56 PM)
[siapa, null (PRONOMINA_INTERROGATIV)] + [pengelola basecamp new selo, null (FRASA_NOMINAL)] = [siapa pengelola basecamp new selo (KALIMAT)]
waktu untuk cek gramatikal = 994

query hasil =
OPTIONAL { ?VAR_4 mount:mapFile ?omap . }
?VAR_4 mount:locatedIn ?VAR_3 .
OPTIONAL { ?VAR_4 mount:pmId ?oid . }
?VAR_4 a mount:Basecamp .
{?VAR_4 rdfs:label ?LB4 . FILTER(regex (?LB4,"New selo","i")) . }
?VAR_01 mount:contactPersonOf ?VAR_4 .
?VAR_01 mount:label ?label .

```

Gambar 5.49 Cuplikan hasil ekstraksi dari *method getQueryLogic*

5.3.5 Pencarian informasi

Proses pencarian informasi dilakukan ke dalam ontologi *Mountaineering* dengan mengeksekusi *query* SPARQL yang dihasilkan pada proses ekstraksi informasi. Proses eksekusi *query* dikerjakan oleh *method generateInformasiAkhir*. Cuplikan kode untuk *method generateInformasiAkhir* dapat dilihat pada Gambar 5.50.

```

1.   public List<List<OutputType>> generateInformasiAkhir(List<OntoBahasa>
2.   resolusi) {
3.       long start = System.currentTimeMillis();
4.       try {
5.           List<List<OutputType>> result = new
6.           ArrayList<List<OutputType>>();
7.           if (resolusi.size() == 0) {
8.               for (int i = 0; i < listFoundGunung.size(); i++) {
9.                   String query = mountReader
10.                      .getQueryDetailMount(listFoundGunung.get(i));
11.                      List<OutputType> itemResult = mountReader
12.                        .queryFinalAnswer(query);
13.                        result.add(itemResult);
14.                }
15.            return result;
16.        } else {
17.            for (OntoBahasa bahasa : resolusi) {
18.                String query = bahasa.getFormatedVars();
19.                System.out.println("query hasil = " + query);
20.                List<OutputType> itemResult =
21.                  mountReader.queryFinalAnswer(query);
22.                  result.add(itemResult);
23.              }
24.            }
25.        } finally { //akhirnya.... :)}
26.        long end = System.currentTimeMillis();
27.        System.out.println("waktu diperlukan generate jawaban = "
+ (end - start));
28.    }
29. }

```

Gambar 5.50 Implementasi eksekusi *query* SPARQL

Informasi hasil eksekusi *query* SPARQL yang dikerjakan oleh *method generateInformasiAkhir* dikirimkan kepada *servlet* dan diterima oleh *class Token*.

Cuplikan kode dari *class Token* untuk menerima informasi akhir dapat dilihat pada Gambar 5.51.

```

1.  List<List<OutputType>> outputResults = preprocessor
2.    .generateInformasiAkhir(resolusi);
3.  request.setAttribute("input", kalimat);
4.  if (getStringDesk(outputResults).isEmpty()
5.      && getStringMaps(outputResults).isEmpty()
6.      && getStringPmId(outputResults).isEmpty()
7.      && getCuaca(outputResults).isEmpty()) {
8.    String pesan = "maaf, sistem tidak memiliki pengetahuan tentang
9.                  informasi yang anda minta, silakan ulangi pencarian";
10.   request.setAttribute("output", pesan);
11. } else {
12.   request.setAttribute("output", getStringDesk(outputResults) + "");
13.   request.setAttribute("maps", new JSONArray(
14.     getStringMaps(outputResults)) + "");
15.   request.setAttribute("pmId", new JSONArray(
16.     getStringPmId(outputResults)) + "");
17.   request.setAttribute("cuaca", getCuaca(outputResults));
18.   request.getRequestDispatcher("/tampiltoken.jsp").forward(request,
19.             response);
20. }

```

Gambar 5.51 Cuplikan kode pada *class Token* untuk mengirimkan informasi ke halaman *tampiltoken.jsp*

Class Token membagi informasi yang akan disajikan kepada pengguna ke dalam tiga kategori, yaitu informasi teks (Gambar 5.51 baris 7), informasi data geografis (Gambar 5.51 baris 8 dan 9) dan informasi cuaca (Gambar 5.51 baris 10).

Informasi teks dapat berisi informasi hasil pencarian atau dapat berisi pesan kesalahan apabila *input* pengguna tidak dapat diproses oleh sistem. Pengiriman informasi teks kepada halaman *tampiltoken.jsp* dilakukan oleh *method getStringDesk* yang dapat dilihat pada Gambar 5.52.

Informasi cuaca berisi data-data cuaca yang diambil dari *web service* milik BMKG secara *online*. Pengiriman informasi cuaca kepada halaman *tampiltoken.jsp* dilakukan oleh *method getCuaca* yang dapat dilihat pada Gambar 5.53.

Informasi geografis berisi parameter untuk menampilkan objek pendakian pada peta beserta data untuk pemanggilan peta. Pengiriman informasi objek pendakian ke halaman *tampiltoken.jsp* dilakukan oleh *method getStringPmID*, sedangkan untuk pemanggilan data peta dilakukan oleh *method getStringMaps*. Cuplikan kode dari *method getStringPmID* dapat dilihat Gambar

5.54 (a), sedangkan cuplikan kode dari *method method getStringMaps* dapat dilihat Gambar 5.54 (b).

```

1.  String getStringDesk(List<List<OutputType>> source) {
2.      String result = "";
3.      for (List<OutputType> sublist : source) {
4.          for (OutputType outputType : sublist) {
5.              if (!outputType.labelStr.equals("")) {
6.                  result += outputType.labelStr + "<br>";
7.              }
8.              if (!outputType.deskripsiStr.equals("")) {
9.                  result += outputType.deskripsiStr + "<br>";
10.             }
11.            //...
12.            //condition for other outputType
13.            //...
14.            if (outputType.jarakInt > 0) {
15.                result += outputType.jarakInt
16.                    + " meter *koordinat berdasarkan GPS Garmin 60S, dihitung
menggunakan Haversine Formula" + "<br>";
17.            }
18.        }
19.    return result;
20. }
```

Gambar 5.52 Cuplikan kode *getStringDesk*

```

1.  String getCuaca(List<List<OutputType>> source) {
2.      String result = "";
3.      CuacaChecker cuacaChecker = null;
4.      for (List<OutputType> subList : source) {
5.          for (OutputType outputType : subList) {
6.              if (!outputType.cuacaNama.equals("") &
!outputType.cuacaFile.equals("")) {
7.                  if (cuacaChecker == null) {
8.                      cuacaChecker = new CuacaChecker();
9.                  }
10.                 cuacaChecker.readData(outputType.cuacaFile,
11.                     outputType.cuacaNama);
12.                 result += cuacaChecker.getKota()
13.                     + "<br>" + "&nbsp;&nbsp;&nbsp;&nbsp;Tanggal : "
14.                     + cuacaChecker.getTglMulai() + " sampai "
15.                     + cuacaChecker.getTglSelesai() + "<br> "
16.                     + "&nbsp;&nbsp;&nbsp;&nbsp;Cuaca : "
17.                     + cuacaChecker.getCuaca() + "<br>"
18.                     + "&nbsp;&nbsp;&nbsp;&nbsp;Kelembaban Max : "
19.                     + cuacaChecker.getKelembabanMax() + "<br>"
20.                     + "&nbsp;&nbsp;&nbsp;&nbsp;Kelembaban Min : "
21.                     + cuacaChecker.getKelembabanMin() + "<br>"
22.                     + "&nbsp;&nbsp;&nbsp;&nbsp;Suhu Max : "
23.                     + cuacaChecker.getSuhuMax() + "<br>"
24.                     + "&nbsp;&nbsp;&nbsp;&nbsp;Suhu Min : "
25.                     + cuacaChecker.getSuhuMin() + "<br>"
26.                     + "&nbsp;&nbsp;&nbsp;&nbsp;Kecepatan Angin : "
27.                     + cuacaChecker.getKecepatanAngin() + "<br>"
28.                     + "&nbsp;&nbsp;&nbsp;&nbsp;<sup>*> Berdasarkan balai "
29.                     + cuacaChecker.getBalai() + "</sup>" + "<br><br>";
30.             }
31.         }
32.     return result;
33. }
```

Gambar 5.53 Cuplikan kode *getCuaca*

```

1.     List<String> getStringPmId(List<List<OutputType>> source) {
2.         List<String> result = new ArrayList<String>();
3.         for (List<OutputType> sublist : source) {
4.             for (OutputType outputType : sublist) {
5.                 if (!outputType.pmPidJson.equals("")) {
6.                     result.add(outputType.pmPidJson);
7.                 }
8.             }
9.         }
10.        return result;
11.    }

```

(a)

```

1.     List<String> getStringMaps(List<List<OutputType>> source) {
2.         List<String> result = new ArrayList<String>();
3.         for (List<OutputType> sublist : source) {
4.             for (OutputType outputType : sublist) {
5.                 if (!outputType.mapJson.equals("")) {
6.                     result.add(outputType.mapJson);
7.                 }
8.             }
9.         }
10.        return result;
11.    }

```

(b)

Gambar 5.54 Cuplikan kode (a) *getStringPmId*. (b) *getStringMaps*

5.3.6 Penyajian informasi

Informasi yang dihasilkan sistem dikirimkan oleh *servlet* kepada halaman *tampiltoken.jsp*. Halaman *tampiltoken.jsp* bertugas menerima informasi dari *servlet*, membentuk tampilan dan menyajikan informasi kepada pengguna. Cuplikan kode dari halaman *tampiltoken.jsp* dapat dilihat pada Gambar 5.55.

Informasi diterima oleh halaman *tampiltoken.jsp* dalam bentuk variabel-variabel (Gambar 5.55 baris 2-7) yang kemudian ditampilkan menggunakan *library jquery accordion* (Gambar 5.55 baris 17 dan 29).

Untuk penyajian informasi geografis, pemanggilan peta dikerjakan oleh fungsi *loadKML* (Gambar 5.55 baris 58) yang ditampilkan menggunakan *id="map_canvas"* (Gambar 5.55 baris 37). Pemanggilan peta yang dilakukan oleh fungsi *loadKML* di-inisialisasi menggunakan variabel *maps* dan *pmid* (Gambar 5.55 baris 56-57). Variabel *maps* digunakan untuk memanggil berkas KML sedangkan variabel *pmid* digunakan untuk pemanggilan objek yang ada di dalam berkas KML.

```

1.   <head>
2.     <% String input = (String) request.getAttribute("input");
3.     String output = (String) request.getAttribute("output");
4.     String cuaca = (String) request.getAttribute("cuaca");
5.     output += cuaca;
6.     Object maps = request.getAttribute("maps");
7.     Object pmId = request.getAttribute("pmId"); %>
8.     <link rel="stylesheet" type="text/css" href="css/jquery-ui.min.css">
9.     <link rel="stylesheet" href="js/lib/jqueryui/themes/smoothness/jquery-ui.css">
10.    <script type="text/javascript" src="js/lib/geoxml3.js"></script>
11.    <script type="text/javascript" src="js/parser/parseKML.js"></script>
12.  </head>

14. <body>
15.   <div id="page">
16.     <div class="content-box" style="width: 100%;>

17.       <script type="text/javascript">
18.         $(function() {
19.           $('#accordion').accordion({
20.             heightStyle: "content"
21.           });
22.           var kmlFiles = <%=maps%>;
23.           if(kmlFiles == '' || kmlFiles == null) {
24.             $('#content').html('');
25.             $('#sidebar').html('');
26.           }
27.         });
28.       </script>

29.       <div id="accordion" style="margin-bottom: 20px;">
30.         <h3>Input : <b><%=input%></b></h3>
31.         <div style="max-height: 150px; min-height: 0px">
32.           <p><%=output%></p>
33.         </div>
34.       </div>

35.       <div id="content">
36.         <div class="post">
37.           <div id="map_canvas"></div>
38.         </div>
39.       </div>

40.       <div id="sidebar">
41.         <ul>
42.           <li>
43.             <h2>Legenda</h2>
44.             <div id="side-legend"></div>
45.           </li>
46.           <li style="margin-top: -25px;">
47.             <h2>Detail</h2>
48.             <div id="side-mark" style="margin-bottom: -20px;"></div>
49.           </li>
50.         </ul>
51.       </div></div>
52.     </div>

52.     <script type="text/javascript">
53.       var geoXMLDoc = null;
54.       var pmIdAll = <%=pmId%>;
55.       var kmlFiles = <%=maps%>;
56.       var kml = remArrayKml(kmlFiles);
57.       var pmId = remArraypmId(pmIdAll);
58.       loadKML('map_canvas', 'elevation_chart', kml, useTheData);
57.     </script>
58.   </body>

```

Gambar 5.55 Cuplikan kode pada halaman *tampiltoken.jsp*

5.3.7 Implementasi KML parser

Fungsi *LoadKML* yang dipanggil pada halaman *tampiltoken.jsp* merujuk pada *library parseKML.js* (Gambar 5.55 baris 11). Cuplikan kode dari *library parseKML.js* dapat dilihat pada Gambar 5.56.

```

1.   function parseKML(map_canvas,kml,afterParseFunc) {
2.     map = new google.maps.Map(
3.       document.getElementById(map_canvas),
4.       {
5.         zoom:12,
6.         zoomControl:true,
7.         mapTypeId:google.maps.MapTypeId.TERRAIN,
8.         scrollwheel : false
9.       }
10.    );
11.    infowindow = new google.maps.InfoWindow({});
12.    if(kml != ''){
13.      geoXML = new geoXML3.parser({
14.        map : map,
15.        infoWindow : infowindow,
16.        singleInfoWindow : false,
17.        afterParse : afterParseFunc
18.      });
19.      geoXML.parse(kml);
20.    }
21.  }
22.
23.  function loadKML(map,elevation,KMLFile,afterParse) {
24.    mapCanvas = map;
25.    KML = KMLFile;
26.    afterParseFunc = afterParse;
27.    parseKML(map,KMLFile,afterParse);
28.  }

```

Gambar 5.56 Cuplikan kode dari *parseKML.js*

Fungsi *parseKML* bertugas mendeskripsikan peta dan fitur peta yang akan digunakan dengan menggunakan variabel yang dikirim oleh fungsi *loadKML*.

ParseKML juga bertugas untuk memanggil *library KML parser*. Fungsi dari *library KML parser* yaitu untuk mem-*parsing* data-data pada berkas KML ke dalam bentuk objek, sehingga data pada berkas KML dapat digunakan dalam proses pencarian. *Library KML parser* yang digunakan pada penelitian ini yaitu *geoxml3* yang dikembangkan oleh Udell (2012).

Seperti yang telah dijabarkan pada Sub Bab 4.4, penelitian ini menggunakan fitur *<Extended data>* untuk mendeskripsikan identitas unik (*id*) dari suatu *instance* pada ontologi *Mountaineering*. Oleh karena itu untuk dapat melakukan *parsing* terhadap *<Extended data>* menggunakan *library KML parser geoxml3* Udell (2012), penelitian ini melakukan modifikasi pada *library tersebut*.

Cuplikan kode modifikasi *library geoxml3* agar dapat melakukan *parsing* *<Extended Data>* dijabarkan pada Gambar 5.57.

```

422. //-----
423. // Custom extdata (2014/09/05) for geoxml3 by GunungBeta :
424. //-----
425. var extDataNodes = node.getElementsByTagName('ExtendedData');
426. if (!extDataNodes && extDataNodes.length > 0) {
427.   var dataNodes = extDataNodes[0].getElementsByTagName('Data');
428.   placemark.extdata = [];
429.   for (var d = 0; d < dataNodes.length; d++) {
430.     var dn = dataNodes[d];
431.     var name = dn.getAttribute('name');
432.     if (!name) continue;
433.     var val =
434.       geoXML3.nodeValue(dn.getElementsByTagName('value')[0]);
435.     placemark.extdata[name] = val;
436.   }
437. }
438. //-----

```

Gambar 5.57 Modifikasi *library geoxml3* untuk melakukan *parsing* *<ExtendedData>*

5.4 Implementasi Antarmuka Aplikasi

Sistem yang dikembangkan pada penelitian ini merupakan sistem berbasis *web* sehingga implementasi aplikasi berupa halaman *web*. Aplikasi terdiri dari dua buah halaman yaitu halaman *utama.jsp* dan halaman *tampiltoken.jsp*. Cuplikan tampilan dari halaman *utama.jsp* dapat dilihat pada Gambar 5.58.



Gambar 5.58 Tampilan halaman *utama.jsp*

Halaman *utama.jsp* terdiri dari bagian *form input* dan bagian *footer*. Bagian *form input* digunakan oleh pengguna untuk berinteraksi dengan sistem

sedangkan bagian *footer* digunakan untuk melihat informasi tambahan tentang sistem.

Form input pada halaman *utama.jsp* terdiri dari satu buah *textbox* dan satu buah *button*. *Textbox* digunakan untuk menerima *input* dari pengguna. *Input* yang diberikan pengguna mulai diproses ketika *input* telah di-*submit* dengan menekan tombol *enter/return* pada *keyboard*. Pada sisi kanan *textbox* terdapat satu buah *button* yang berfungsi untuk memberikan informasi bagaimana sistem dapat digunakan.

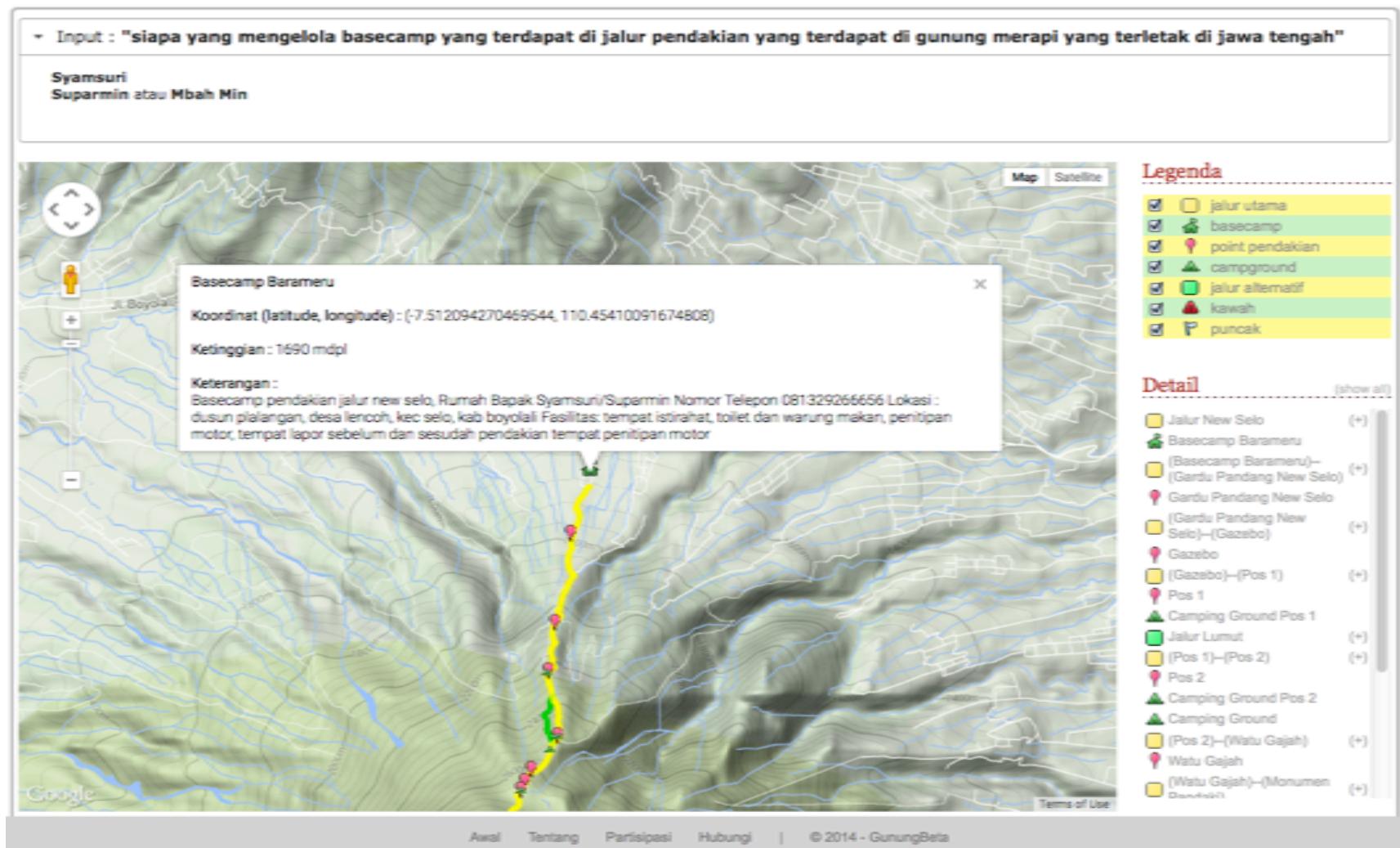
Selain terdapat *form input* pada halaman *utama.jsp* juga *jsp* terdapat *footer*. *Footer* berada pada bagian bawah halaman, *footer* berguna untuk menjelaskan informasi tentang aplikasi.

Halaman *tampiltoken.jsp* berfungsi sebagai halaman untuk menyajikan informasi hasil pemrosesan *input*. Halaman *tampiltoken.jsp* terdiri dari bagian teks, bagian peta dan bagian *footer*. Cuplikan tampilan halaman *tampiltoken.jsp* dapat dilihat pada Gambar 5.55.

Bagian teks pada halaman *tampiltoken.jsp* berada pada posisi paling atas, berfungsi untuk menyajikan informasi berupa teks, informasi teks yang dapat disajikan yaitu: informasi jalur pendakian jika *input* dapat diproses sistem dan informasi pesan kesalahan jika *input* tidak dapat diproses oleh sistem.

Bagian peta terdapat dibawah bagian teks, fungsi dari bagian peta adalah menyajikan informasi geografis berupa peta. Bagian peta terdiri dari tiga komponen, yaitu: komponen peta, komponen legenda dan komponen detail. Pengguna dapat berinteraksi dengan masing-masing komponen dengan menggunakan alat penunjuk (*mouse*).

Komponen peta menyajikan informasi lokasi suatu objek pada permukaan bumi. Komponen legenda berisi *icon* mewakili simbol-simbol yang digunakan oleh objek pada peta. Komponen detail memberikan informasi detail tentang objek. Masing-masing komponen hanya ditampilkan apabila *input* pengguna dapat diproses oleh sistem. Jika pada bagian teks memberikan informasi kesalahan bagian peta tidak dimunculkan.



Gambar 5.59 Tampilan halaman *tampiltoken.jsp*

BAB VI

HASIL DAN PEMBAHASAN

6.1 Hasil Implementasi Sistem

Subbab ini menjabarkan hasil dari implementasi sistem yang telah dijabarkan pada Bab V. Pembahasan hasil implementasi dilakukan untuk mengetahui apakah kemampuan sistem telah sesuai dengan kebutuhan dan latar belakang masalah yang dijabarkan pada Bab I.

Berdasarkan tujuan dari penelitian ini, unjuk kerja kemampuan sistem dibagi ke dalam beberapa bagian, yaitu:

1. Kemampuan sistem dalam mendeteksi kesalahan pengetikan pada *input* dan memberikan saran perbaikan kata menggunakan fitur *spelling checker*.
2. Kemampuan sistem menggunakan *thesaurus* ketika melakukan pencarian informasi.
3. Kemampuan sistem dalam memahami *input* bahasa alami berupa kata, frasa, klausa, kalimat. Dengan batasan-batasan yang telah dijabarkan pada Sub Bab 1.3.
4. Kemampuan sistem dalam mendeteksi *input* yang tidak sesuai dengan kaidah tata bahasa Indonesia.
5. Kemampuan sistem menyajikan informasi dengan basis penyajian informasi berupa peta interaktif.

Input yang digunakan untuk melihat unjuk kerja kemampuan sistem dapat dilihat pada Tabel 6.1.

Tabel 6.1 *Input* yang dipilih untuk melihat kemampuan sistem

No	Input	Keterangan
a1	<i>ciapa jang mengelkola basecamp barameru</i>	contoh <i>input</i> untuk pengecekan kesalahan pengetikan kata
b1	<i>merapi</i>	contoh <i>input</i> untuk penggunaan <i>thesaurus</i>
c1	<i>jalur pendakian merapi</i>	contoh <i>input</i> untuk pemahaman frasa
d1	<i>basecamp yang ada di merapi</i>	contoh <i>input</i> untuk pemahaman klausa
e1	<i>jalur pendakian new selo letaknya di mana</i>	contoh <i>input</i> untuk pemahaman kalimat tunggal
e2	<i>dimana lokasi jalur new selo</i>	
e3	<i>jalur pendakian new selo terdapat di gunung apa</i>	
f1	<i>basecamp barameru dikelola oleh siapa</i>	contoh <i>input</i> untuk pemahaman kalimat tunggal dengan pola yang berbeda namun memiliki kesamaan makna
f2	<i>dikelola oleh siapa basecamp barameru</i>	
g1	<i>berapa ketinggian basecamp pendakian barameru yang terletak di gunung merapi</i>	contoh <i>input</i> untuk pemahaman kalimat majemuk dengan pola berbeda namun memiliki makna yang sama
g2	<i>basecamp barameru yang terletak di gunung merapi ketinggiannya berapa</i>	
h1	<i>apa saja basecamp yang terdapat di jalur pendakian selo yang berlokasi di gunung yang terletak di jawa tengah</i>	contoh <i>input</i> untuk pemahaman kalimat majemuk bertingkat yang akan salah apabila hanya didasarkan pada analisis sintaksis saja
i1	<i>letaknya dimana gunung merapi</i>	contoh <i>input</i> untuk mendeteksi kesalahan sintaksis
i2	<i>siapa merapi</i>	contoh untuk mendeteksi <i>input</i> yang salah secara semantik
i3	<i>apa saja basecamp di gunung</i>	contoh <i>input</i> untuk mendeteksi penyusun kalimat yang tidak lengkap

6.1.1 Fitur *spelling checker*

Penggunaan fitur *spelling checker* bertujuan untuk membantu pengguna mengurangi kesalahan pengetikan selama proses *input*. Fitur *spelling checker* bekerja selama pengetikan *input* sedang berlangsung dan *input* belum di-submit. Misalnya sistem diberikan *input* pada Tabel 6.1 nomor a1 “*ciapa jang mengelkola basecamp barameru*”, saran perbaikan kata atas *input* pada Tabel 6.1 nomor a1 yaitu “*siapa yang mengelola basecamp barameru*”. Tampilan penyajian saran kata-kata perbaikan selama proses *input* dapat dilihat pada Gambar 6.1.

Cuplikan proses pengecekan yang dilakukan sistem dapat dilihat pada Gambar 6.2.

Spelling checker tidak dapat memberikan saran perbaikan kata apabila kata-kata yang di-*input* oleh pengguna tidak dikenali oleh sistem yaitu kata-kata yang tidak terdapat didalam kamus kata yang digunakan oleh *spelling checker* seperti yang telah dijabarkan pada Sub Bab 5.3.1.



Gambar 6.1 Tampilan fitur *spelling checker* ketika memberikan saran perbaikan kata

The screenshot shows a Java console window titled "Tomcat v7.0 Server at localhost [Apache Tomcat] /Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (Mar 11, 2015, 3:07:34 PM)". The console output displays a series of JSON objects, each containing a key "0" and a value representing a misspelled word followed by its suggested correction. The words include "ciapa", "jang", "menge", "mengelkola", "mengelko", "mengelkol", "mengelkola", "mengelola", and "basecamp", with their respective corrections like "siapa", "siapa yang", "siapa yang menge", "siapa yang mengelkola", "siapa yang mengelko", "siapa yang mengelkol", "siapa yang mengelkola", "siapa yang mengelola", and "basecamp".

```

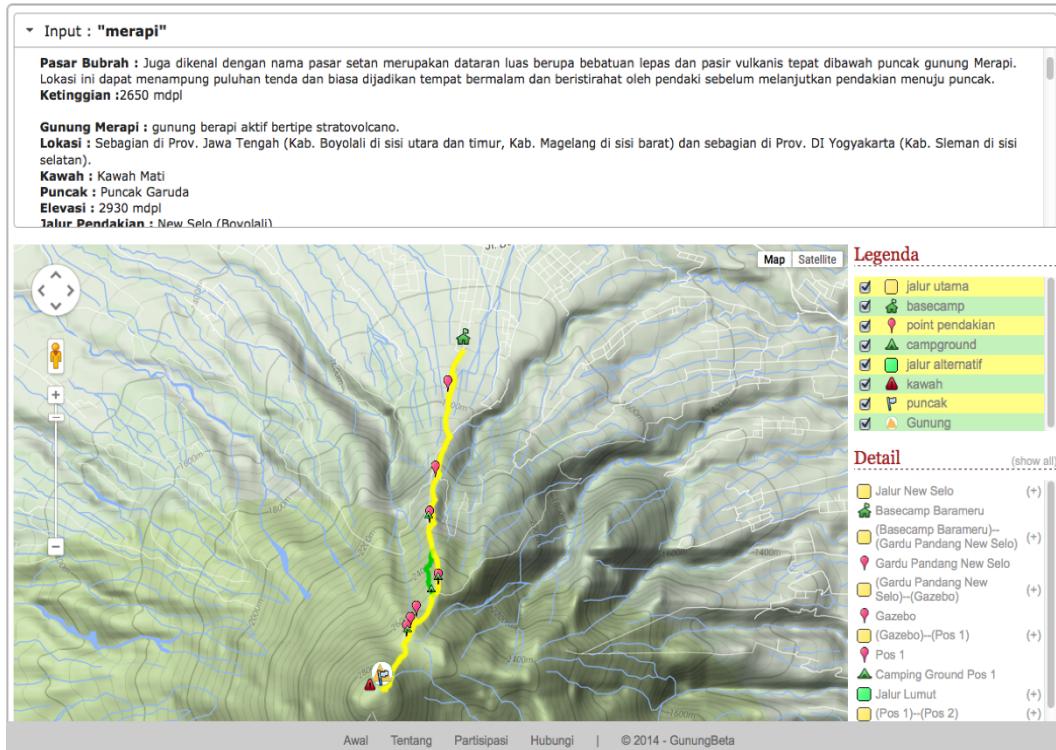
Properties Servers Console Progress
Tomcat v7.0 Server at localhost [Apache Tomcat] /Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (Mar 11, 2015, 3:07:34 PM)
json = {"Key 0":["siapa"]}
ciapa
json = {"Key 0":["siapa"]}
ciapa jang
json = {"Key 0":["siapa yang"]}
ciapa jang men
json = {"Key 0":["siapa yang Min"]}
ciapa jang meng
json = {"Key 0":["siapa yang manakah"]}
ciapa jang menge
json = {"Key 0":["siapa yang menge"]}
ciapa jang mengelk
json = {"Key 0":["siapa yang mengelk"]}
ciapa jang mengeelko
json = {"Key 0":["siapa yang mengelko"]}
ciapa jang mengelkol
json = {"Key 0":["siapa yang mengelkol"]}
ciapa jang mengelkola
json = {"Key 0":["siapa yang mengelola"]}
ciapa jang mengelkola
json = {"Key 0":["siapa yang mengelola"]}
ciapa jang mengelkola base
json = {"Key 0":["siapa yang mengelola pos"]}
ciapa jang mengelkola basec
json = {"Key 0":["siapa yang mengelola basec"]}
ciapa jang mengelkola basecam
json = {"Key 0":["siapa yang mengelola basecam"]}
ciapa jang mengelkola basecamp
json = {"Key 0":["siapa yang mengelola basecamp"]}
ciapa jang mengelkola basecamp ba
json = {"Key 0":["siapa yang mengelola basecamp ba"]}
ciapa jang mengelkola basecamp bara
json = {"Key 0":["siapa yang mengelola basecamp Bara"]}
ciapa jang mengelkola basecamp baram
json = {"Key 0":["siapa yang mengelola basecamp Bara"]}
ciapa jang mengelkola basecamp barameru
json = {"Key 0":["siapa yang mengelola basecamp Barameru"]}

```

Gambar 6.2 Cuplikan proses *spelling checker*

6.1.2 Kemampuan sistem menggunakan *thesaurus* kata

Kemampuan sistem melakukan pencarian informasi menggunakan *thesaurus* kata ditunjukkan dengan memberikan *input* dari Tabel 6.1 nomor b1 “merapi”. Hasil pencarian sistem dapat dilihat pada Gambar 6.3.



Gambar 6.3 Hasil pencarian dengan penggunaan *thesaurus* kata

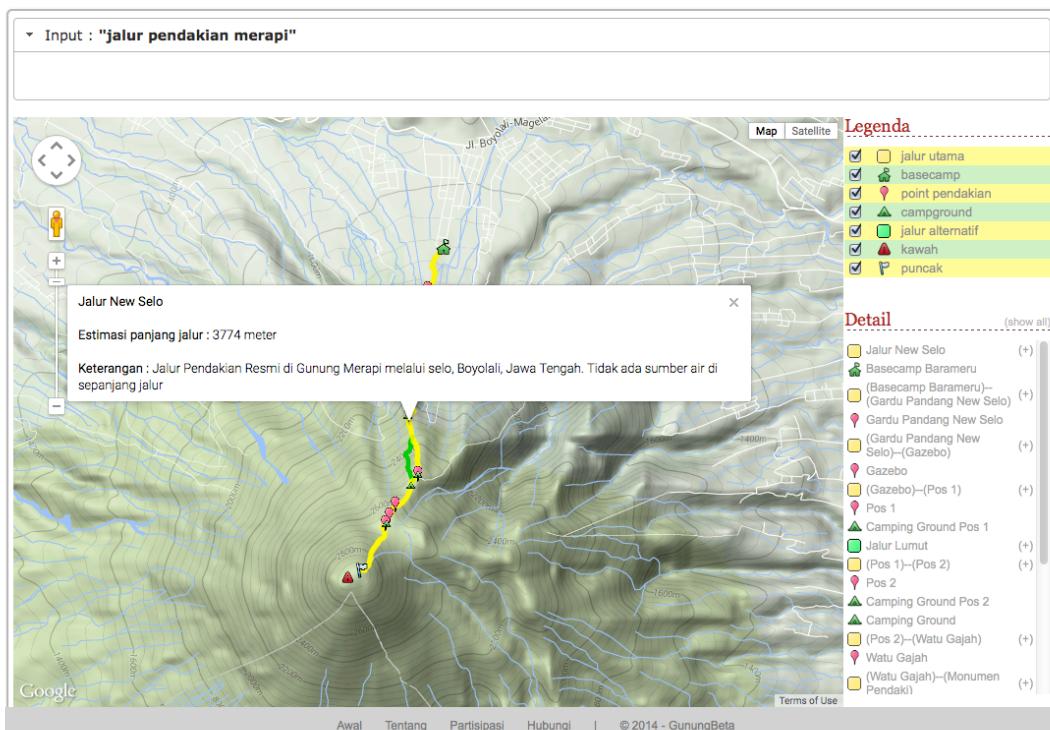
Pada Gambar 6.3 dapat dilihat bahwa sistem mampu menyajikan informasi semua objek pendakian yang memiliki keterkaitan dengan *input* dari Tabel 6.1 nomor b1 yaitu kata “merapi”, informasi juga disertai dengan penyajian objek pada peta. Kemampuan sistem menggunakan *thesaurus* memudahkan pengguna dalam melakukan pencarian, sehingga pengguna tidak perlu melakukan pencarian berulang-ulang untuk mencari informasi yang saling memiliki keterkaitan. Keterkaitan yang dideskripsikan pada penelitian ini adalah keterkaitan berdasarkan letak/lokasi suatu objek dan keterkaitan berdasarkan makna kata.

6.1.3 Kemampuan sistem memahami *input* bahasa alami

Kemampuan sistem dalam memahami *input* bahasa alami mengacu kepada dua hal yaitu analisis struktur luar (*surface structure*) dan analisis struktur dalam

(*deep structure*). Analisis struktur luar berhubungan dengan kemampuan sistem memahami *input* secara sintaksis sedangkan analisis struktur dalam berhubungan dengan kemampuan sistem memahami *input* secara semantis.

Alwi (2003) memaparkan satuan sintaksis terkecil dalam bahasa adalah kata, satuan yang lebih besar secara berurutan yaitu, frasa, klausa dan kalimat. Frasa merupakan satuan sintaksis yang terdiri atas dua kata atau lebih yang tidak mengandung unsur predikat. Kemampuan sistem untuk memahami *input* berupa frasa ditunjukkan dengan memberikan *input* dari Tabel 6.1 nomor c1 “*jalur pendakian merapi*”. Cuplikan hasil pencarian dapat dilihat pada Gambar 6.4.



Gambar 6.4 Hasil pencarian informasi menggunakan *input* berupa frasa

Berdasarkan cuplikan hasil pencarian informasi pada Gambar 6.4 sistem dapat menggunakan pengetahuan yang dimilikinya untuk memahami bahwa maksud dari *input* pada Tabel 6.1 nomor c1 “*Jalur pendakian merapi*” adalah jalur pendakian yang terdapat di gunung Merapi bukan jalur pendakian bernama Merapi dan jalur pendakian yang terdapat di gunung Merapi yaitu jalur New Selo. Cuplikan dari proses pemahaman sistem bahwa *input* “*jalur pendakian merapi*” mengandung frasa dapat dilihat pada Gambar 6.5.

Properties Servers Console Progress

Tomcat v7.0 Server at localhost [Apache Tomcat] /Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (Mar 11, 2015, 3:53:37 PM)

```
----- Start cek 2 token-----
[jalur pendakian, null, NOMINA] , [NAME, merapi, NOMINA]
check Rule (1), K -> FNO FN1
check Rule (3) K -> FNO FN1
check Rule (26, 27, 28) FN -> N FN

----- Start cek 2 token-----
[jalur pendakian, null, NOMINA] , [NAME, merapi, NOMINA]
check Rule (1), K -> FNO FN1
check Rule (3) K -> FNO FN1
check Rule (26, 27, 28) FN -> N FN

----- Start cek 2 token-----
[jalur pendakian, null, NOMINA] , [NAME, merapi, NOMINA]
check Rule (1), K -> FNO FN1
check Rule (3) K -> FNO FN1
check Rule (26, 27, 28) FN -> N FN

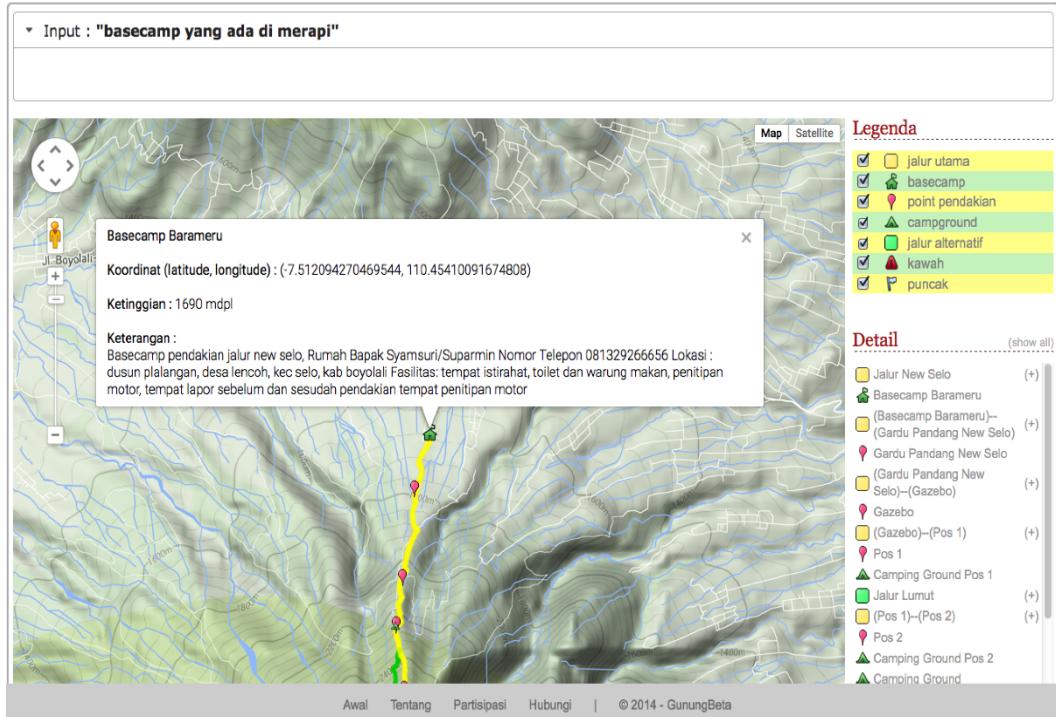
----- Start cek 2 token-----
[jalur pendakian, null, NOMINA] , [NAME, merapi, NOMINA]
check Rule (1), K -> FNO FN1
check Rule (3) K -> FNO FN1
check Rule (26, 27, 28) FN -> N FN
TRUE
OntoBahasa [subject=jalur2, nilai=jalur pendakian, kategori=NOMINA, tipe0=[Ttemp32, Tcamp31, Tterdapatt2, Tdimanall1, Tair31, Tbase31, Ti
OntoBahasa [subject=Template22, nilai=NAME, kategori=NOMINA, tipe0=[Tkawah2, Tgungunung2, Tcamp2, Tjurkun2, Tmengelola2, Tpuncak2, Tjalur
[jalur pendakian, null (NOMINA)] + [NAME, merapi (NOMINA)]= [jalur pendakian merapi (FRASA_NOMINAL)] >
hasil pengecekan bahasa : OntoBahasa [subject=null, nilai=jalur pendakian merapi, kategori=FRASA_NOMINAL, tipe0=[Ttemp32, Tcamp31, Tte

waktu untuk cek sintaksis = 567
query hasil = OPTIONAL { ?VAR_2 mount:pmlid ?jid . } OPTIONAL { ?VAR_2 mount:mapFile ?jmp . } ?VAR_2 a mount:Jalur . {?VAR_2 rdfs:la
final PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX xsd: <http://
waktu diperlukan generate jawaban = 17
waktu diperlukan kirim data = 19
waktu diperlukan semua proses = 885
```

Gambar 6.5 Cuplikan proses untuk *input* yang mengandung satuan sintaksis berupa frasa

Alwi (2003) memaparkan perbedaan mendasar antara frasa dan klausa yaitu pada klausa mengandung unsur predikat sedangkan pada frasa tidak. Untuk menunjukkan kemampuan sistem memahami *input* yang mengandung satuan sintaksis berupa klausa digunakan *input* pada Tabel 6.1 nomor *d1* “*basecamp yang ada di merapi*”. Cuplikan hasil pencarian informasi untuk *input* pada Tabel 6.1 nomor *d1* dapat dilihat pada Gambar 6.6.

Berdasarkan cuplikan pada Gambar 6.6 sistem dapat memahami maksud dari *input* pada Tabel 6.1 nomor *d1* “*basecamp yang ada di merapi*” adalah *basecamp* jalur pendakian yang letaknya di gunung Merapi dan nama *basecamp* yang dimaksud adalah *basecamp* Barameru. Cuplikan proses pemahaman sistem bahwa *input* pada Tabel 6.1 nomor *d1* mengandung satuan sintaksis klausa dapat dilihat pada Gambar 6.7



Gambar 6.6 Hasil pencarian informasi menggunakan *input* yang mengandung klausa

```

Properties Servers Console Progress
Tomcat v7.0 Server at localhost [Apache Tomcat] /Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (Mar 12, 2015, 8:10:47 PM)

----- Start cek 3 token-----
[basecamp, null, NOMINA] , [yang, null, ARTIKULA] , [ada di merapi, null, FRASA_VERBAL]

----- Start cek 3 token-----
[basecamp, null, NOMINA] , [yang, null, ARTIKULA] , [ada di merapi, null, FRASA_VERBAL]

----- Start cek 3 token-----
[basecamp, null, NOMINA] , [yang, null, ARTIKULA] , [ada di merapi, null, FRASA_VERBAL]

----- Start cek 3 token-----
[basecamp, null, NOMINA] , [yang, null, ARTIKULA] , [ada di merapi, null, FRASA_VERBAL]

----- Start cek 3 token-----
[basecamp, null, NOMINA] , [yang, null, ARTIKULA] , [ada di merapi, null, FRASA_VERBAL]

----- Start cek 3 token-----
[basecamp, null, NOMINA] , [yang, null, ARTIKULA] , [ada di merapi, null, FRASA_VERBAL]

----- Start cek 3 token-----
[basecamp, null, NOMINA] , [yang, null, ARTIKULA] , [ada di merapi, null, FRASA_VERBAL]

----- Start cek 2 token-----
[basecamp, null, NOMINA] , [yang ada di merapi, null, KLAUSA]
check Rule (4) FN K' hasil = K'
check Rule (31), FNO -> FNI K'

----- Start cek 2 token-----
[basecamp, null, NOMINA] , [yang ada di merapi, null, KLAUSA]
check Rule (4) FN K' hasil = K'
check Rule (31), FNO -> FNI K'
TRUE
OntoBahasa [subject=base31, nilai=basecamp, kategori=NOMINA, tipe0=[Tkettinggian3, Tinformatasi3, Tmengelola3, Tpengelola3, Tapa3, Tnotel3, Tpeta
OntoBahasa [subject=null, nilai=yang ada di merapi, kategori=KLAUSA, tipe0=[Tpos31, Tjalur31, Tgunung31, Ttitik3, Tkawah31, Tbase31, Tir31, T
[basecamp, null (NOMINA)] + [yang ada di merapi, null (KLAUSA)] = [basecamp yang ada di merapi (FRASA_NOMINAL)]
hasil pengecekan bahasa : OntoBahasa [subject=null, nilai=basecamp yang ada di merapi, kategori=FRASA_NOMINAL, tipe0=[Tkettinggian3, Tinformatasi
waktu untuk cek sintaksis = 1051
query hasil = (?VAR_2 rdfs:label ?LB2 . FILTER(regex (?LB2,"Merapi","i")) .) ?VAR_3 mount:berlokasiDi ?VAR_2 . ?VAR_3 mount:berlokasiDi ?VA
final QUERY = PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX xsd: <http://www.w
waktu diperlukan generate jawaban = 9
waktu diperlukan kirim data = 11
waktu diperlukan semua proses = 1232

```

Gambar 6.7 Cuplikan proses untuk *input* yang mengandung satuan sintaksis berupa klausa

Alwi (2003) memaparkan bahwa orang membentuk kalimat berdasarkan pengetahuan dunia yang ada disekelilingnya. Kalimat dalam banyak hal tidak berbeda dari klausa, baik klausa atau kalimat merupakan konsturksi sintaksis yang mengandung unsur predikasi.

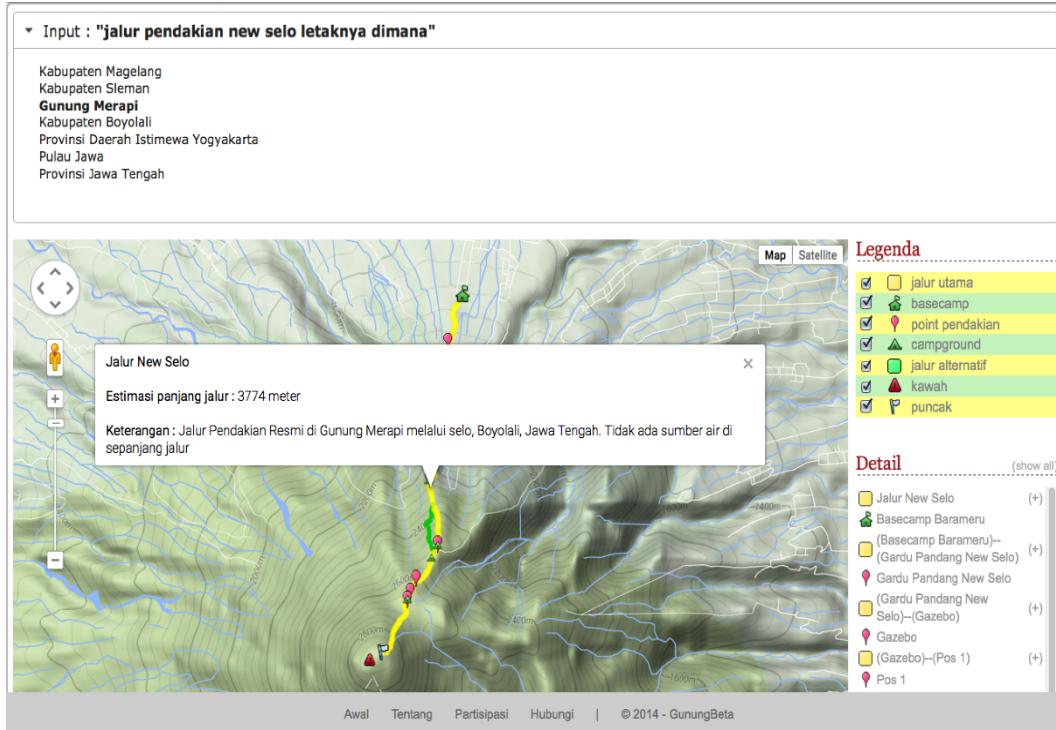
Sistem yang dikembangkan pada penelitian ini membedakan kalimat dan klausa yaitu dari penggunaan kata tanya untuk mendeskripsikan kalimat tanya dan penggunaan kata perintah untuk mendeskripsikan kalimat perintah. Kata tanya yang digunakan yaitu: “siapa”, “berapa”, “dimana” dan “apa”. Kata perintah yang digunakan pada penelitian ini yaitu: “tampilkan”.

Kemampuan sistem memahami *input* yang mengandung satuan sintaksis berupa kalimat ditunjukkan dengan menggunakan *input* pada Tabel 6.1 nomor *e1* “*jalur pendakian new selo letaknya dimana*” yang merupakan kalimat tunggal dengan pola S-P dan *input* nomor *e2* “*dimana lokasi jalur new selo*” yang merupakan kalimat tunggal dengan pola P-S.

Cuplikan hasil pencarian untuk *input* pada Tabel 6.1 nomor *e1* dapat dilihat pada Gambar 6.8, dan cuplikan proses pemahaman sistem untuk *input* pada Tabel 6.1 nomor *e1* dapat dilihat pada Gambar 6.9.

Cuplikan hasil pencarian untuk *input* pada Tabel 6.1 nomor *e2* dapat dilihat pada Gambar 6.10, dan cuplikan proses pemahaman sistem untuk *input* pada Tabel 6.1 nomor *e2* dapat dilihat pada Gambar 6.11.

Berdasarkan cuplikan hasil pencarian yang dijabarkan pada Gambar 6.8 dan Gambar 6.10 menunjukkan bahwa sistem mampu memahami *input* berupa kalimat dengan pola berbeda namun memiliki makna yang sama, selain itu sistem juga mampu menyajikan informasi hasil pencarian ke dalam bentuk peta.



Gambar 6.8 Hasil pencarian infomasi menggunakan *input* berupa kalimat tunggal berpolia S-P

```
Properties Servers Console Progress

Tomcat v7.0 Server at localhost [Apache Tomcat / Library / Java / JavaVirtualMachines / jdk1.7.0_67 jdk / Contents / Home / bin / java (Mar 12, 2015, 8:10:47 PM)
[NAME, new selo, NOMINA] , [letaknya dimana, null, KALIMAT]
check Rule (19), K -> FN K
TRUE
OntoBahasa [subject="Template21, nilai=NAME, kategori=NOMINA, tipe0=[Tdikelola2, Tpengelola2, Tpuncak2, Tketinggian2, Tinformasi2, Tgungung2, Tp
OntoBahasa [subject=null, nilai=letaknya dimana, kategori=KALIMAT, tipe0=null, kat0=null, tipel=null, kat1=null, tipe2=null, kat2=null, tipe5=]

[NAME, new selo (NOMINA) ] + [letaknya dimana, null (KALIMAT) ] = [new selo letaknya dimana (KALIMAT) ]

hasil pengecekan bahasa : OntoBahasa [subject=null, nilai=new selo letaknya dimana, kategori=KALIMAT, tipe0=null, kat0=null, tipel=null, kat1=]

----- Start cek 2 token-----
[jalur pendakian, null, NOMINA] , [new selo letaknya dimana, null, KALIMAT]
check Rule (19), K -> FN K

----- Start cek 2 token-----
[jalur pendakian, null, NOMINA] , [new selo letaknya dimana, null, KALIMAT]
check Rule (19), K -> FN K

----- Start cek 2 token-----
[jalur pendakian, null, NOMINA] , [new selo letaknya dimana, null, KALIMAT]
check Rule (19), K -> FN K

----- Start cek 2 token-----
[jalur pendakian, null, NOMINA] , [new selo letaknya dimana, null, KALIMAT]
check Rule (19), K -> FN K

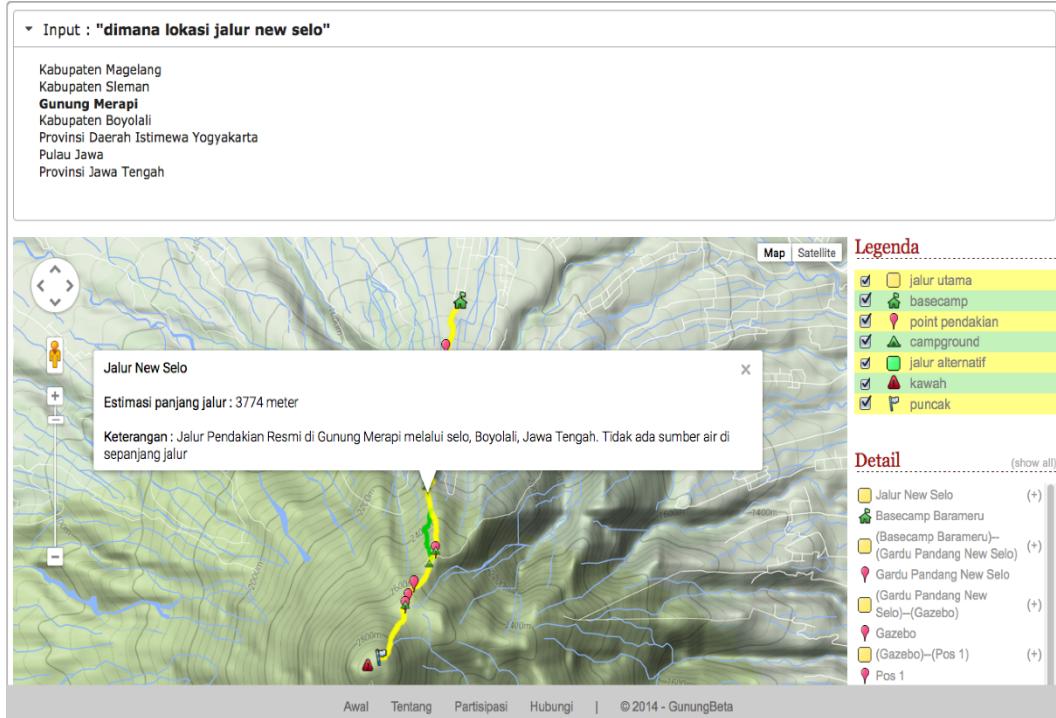
----- Start cek 2 token-----
[jalur pendakian, null, NOMINA] , [new selo letaknya dimana, null, KALIMAT]
check Rule (19), K -> FN K

----- Start cek 2 token-----
[jalur pendakian, null, NOMINA] , [new selo letaknya dimana, null, KALIMAT]
check Rule (19), K -> FN K

hasil pengecekan bahasa : OntoBahasa [subject=null, nilai=jalur pendakian new selo letaknya dimana, kategori=KALIMAT, tipe0=null, kat0=null, t

waktu untuk cek sintaksis = 615
query hasil = ?VAR_1 mount:label ?label . ?VAR_2 mount:berlokasiID ?VAR_1 . (?VAR_2 rdfs:label ?LB2 . FILTER(regex (?LB2,"New selo","i")) . )
final QUERY = PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX xsd: <http://www.w
waktu diperlukan generate jawaban = 43
waktu diperlukan Kirim data = 44
waktu diperlukan semua proses = 808
```

Gambar 6.9 Cuplikan proses pemahaman sistem untuk *input* berupa kalimat tunggal berpola S-P



Gambar 6.10 Hasil pencarian informasi menggunakan *input* berupa kalimat tunggal berpola P-S

```

Properties Servers Console Progress
Tomcat v7.0 Server at localhost [Apache Tomcat] /Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (Mar 12, 2015, 8:10:47 PM)
check Rule (1), K -> FNO FNI
check Rule (3) K -> FNO FNI
check Rule (26, 27, 28) FN -> N FN

----- Start cek 3 token-----
[dimana, null, PRONOMINA_INTERROGATIV] , [lokasi, null, NOMINA] , [jalur new selo, null, FRASA_NOMINAL]

----- Start cek 3 token-----
[dimana, null, PRONOMINA_INTERROGATIV] , [lokasi, null, NOMINA] , [jalur new selo, null, FRASA_NOMINAL]

----- Start cek 3 token-----
[dimana, null, PRONOMINA_INTERROGATIV] , [lokasi, null, NOMINA] , [jalur new selo, null, FRASA_NOMINAL]

----- Start cek 2 token-----
[lokasi, null, NOMINA] , [jalur new selo, null, FRASA_NOMINAL]
check Rule (1), K -> FNO FNI
check Rule (3) K -> FNO FNI
check Rule (26, 27, 28) FN -> N FN
TRUE
OntoBahasa [subject=lokasi2, nilai=lokasi, kategori=NOMINA, tipe0=[Tdimanall], kat0=[NOMINA], tipe1=[Tgunung2, Tgunung2, Tkawah2, Tkawah2, Tter
OntoBahasa [subject=null, nilai=jalur new selo, kategori=FRASA_NOMINAL, tipe0=[Ttemp32, Tcamp31, Tterdapat2, Tdimanall, Tair31, Tbase31, Tpanj
[jaluri, null (NOMINA)] + [jalur new selo, null (FRASA_NOMINAL)] = [lokasi jalur new selo (FRASA_NOMINAL)]

hasil pengecekan bahasa : OntoBahasa [subject=null, nilai=lokasi jalur new selo, kategori=FRASA_NOMINAL, tipe0=[Tdimanall], kat0=[NOMINA], tip

----- Start cek 2 token-----
[dimana, null, PRONOMINA_INTERROGATIV] , [lokasi jalur new selo, null, FRASA_NOMINAL]
check Rule (1), K -> FNO FNI
check Rule (3) K -> FNO FNI
TRUE
OntoBahasa [subject=dimanall2, nilai=dimana, kategori=PRONOMINA_INTERROGATIV, tipe0=[Tcamp52, Tlokasi2, Tjalur2, Tcamp32, Tcamp31, Tair52, Tair2
OntoBahasa [subject=null, nilai=lokasi jalur new selo, kategori=FRASA_NOMINAL, tipe0=[Tdimanall1], kat0=[NOMINA], tipe1=null, kat1=null, tipe2=null
[dimana, null (PRONOMINA_INTERROGATIV)] + [lokasi jalur new selo, null (FRASA_NOMINAL)] <--> [dimana lokasi jalur new selo (KALIMAT)]
hasil pengecekan bahasa : OntoBahasa [subject=null, nilai=dimana lokasi jalur new selo, kategori=KALIMAT, tipe0=null, kat0=null, tipe1=null, k

waktu untuk cek sintaksis = 926
query hasil = OPTIONAL { ?VAR_2 mount:pmId ?jid . } OPTIONAL { ?VAR_2 mount:mapFile ?jmap . } ?VAR_2 a mount:Jalur . { ?VAR_2 rdfs:label ?LB
final QUERY = PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX xsd: <http://www.w
waktu diperlukan generate jawaban = 139
waktu diperlukan kirim data = 142
waktu diperlukan semua proses = 1325

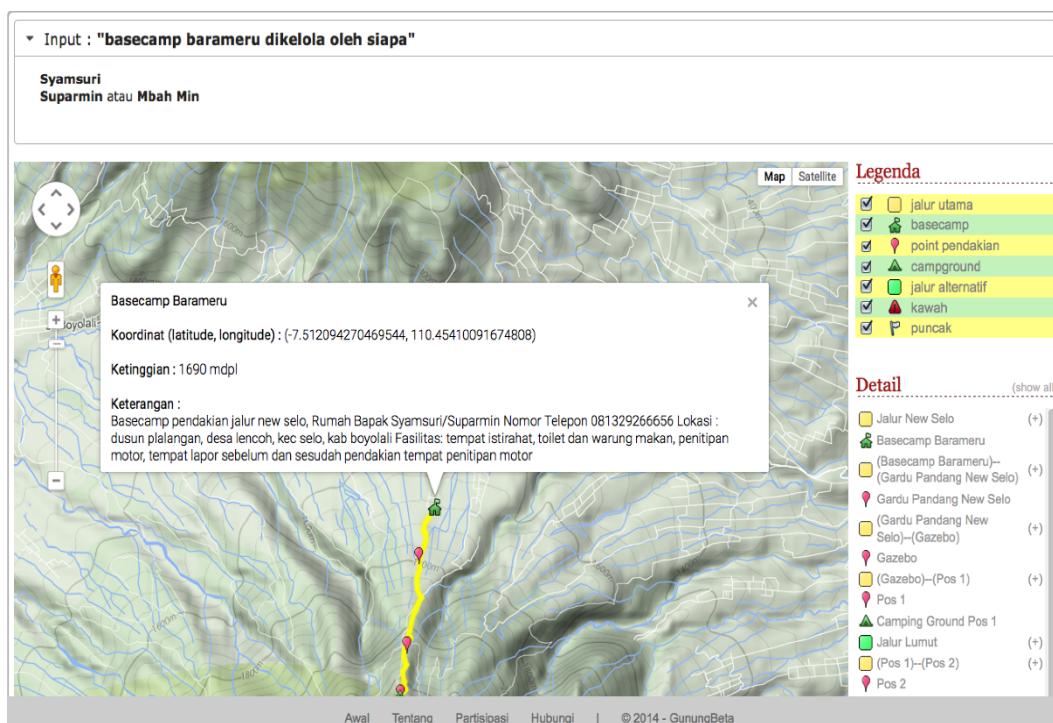
```

Gambar 6.11 Cuplikan proses untuk *input* berupa kalimat tunggal berpola P-S

Alwi (2003) memaparkan kalimat juga dapat berpola S-P-O atau P-O-S jika predikatnya berupa verba, untuk melihat kemampuan sistem memproses kalimat dengan pola S-P-O atau P-O-S digunakan *input* pada Tabel 6.1 nomor *f1* “*basecamp barameru dikelola oleh siapa*” dan nomor *f2* “*dikelola oleh siapa basecamp barameru*”.

Cuplikan hasil pencarian dan proses pemahaman sistem untuk *input* pada Tabel 6.1 nomor *f1* dapat dilihat pada Gambar 6.12 dan Gambar 6.13. Cuplikan hasil pencarian dan proses pemahaman sistem untuk *input* pada Tabel 6.1 nomor *f2* dapat dilihat pada Gambar 6.14 dan Gambar 6.15.

Berdasarkan hasil pencarian yang dijabarkan pada Gambar 6.12 dan Gambar 6.14 menunjukkan bahwa sistem mampu memahami kalimat tanya tunggal dengan pola yang berbeda yaitu S-P-O dan P-O-S, namun memiliki makna yang sama yaitu sama-sama menanyakan informasi tentang orang yang mengelola *basecamp* Barameru.



Gambar 6.12 Hasil pencarian untuk *input* berupa kalimat tanya tunggal berpola S-P-O

```

Properties Servers Console Progress
Tomcat v7.0 Server at localhost [Apache Tomcat] /Library/java/javaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (Mar 13, 2015, 3:58:26 PM)
check Rule (1), K -> FN0 FN1
check Rule (3) K -> FN0 FN1
check Rule (26, 27, 28) FN -> N FN

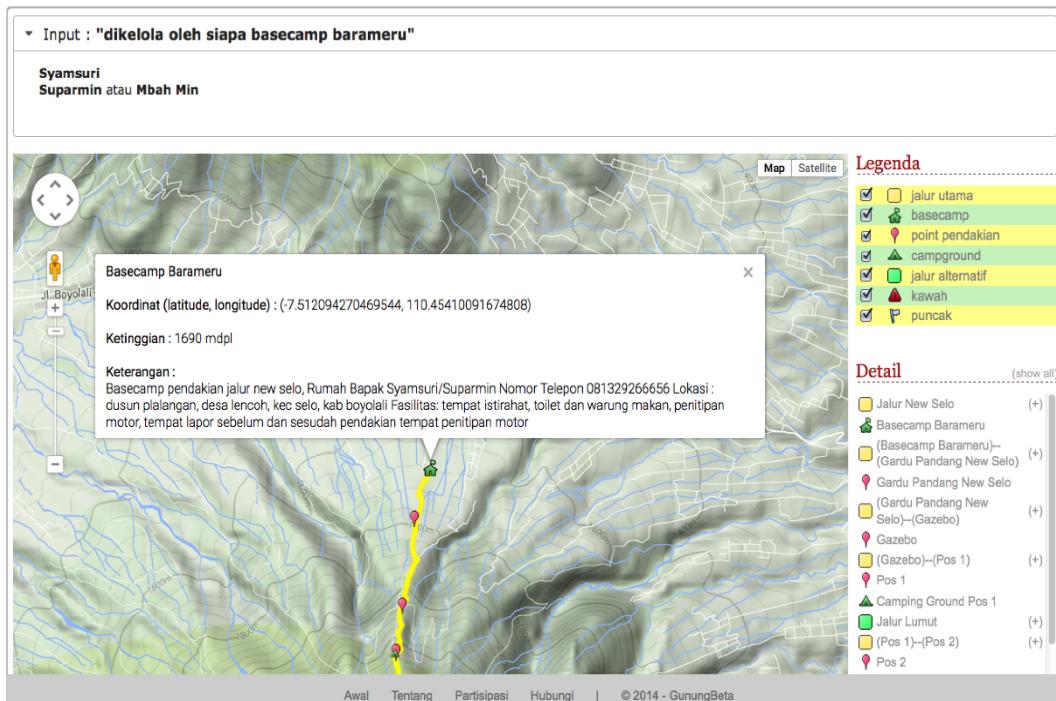
----- Start cek 2 token-----
[basecamp, null, NOMINA] , [NAME, barameru, NOMINA]
check Rule (1), K -> FN0 FN1
check Rule (3) K -> FN0 FN1
check Rule (26, 27, 28) FN -> N FN

----- Start cek 2 token-----
[basecamp, null, NOMINA] , [NAME, barameru, NOMINA]
check Rule (1), K -> FN0 FN1
check Rule (3) K -> FN0 FN1
check Rule (26, 27, 28) FN -> N FN
TRUE
OntoBahasa [subject=base2, nilai=basecamp, kategori=NOMINA, tipe0=[Tdimanall, Tkettinggian2, Tpeta, Tinformatasi2, Tnotel2, Tmengelola2, Tterdapat2, Tpen
OntoBahasa [subject=Template22, nilai=NAME, kategori=NOMINA, tipe0=[Tkawah2, Tgunung2, Tcamp2, Tjurkun2, Tmengelola2, Tpuncak2, Tjalur2, Tbase2], kat0
[basecamp, null (NOMINA)] + [NAME, barameru (NOMINA)] = [basecamp barameru (FRASA_NOMINAL)]

hasil pengecekan bahasa : OntoBahasa [subject=null, nilai=basecamp barameru, kategori=FRASA_NOMINAL, tipe0=[Tdimanall1, Tkettinggian2, Tpeta, Tinformati
check Rule (1), K -> FN0 FN1
check Rule (3) K -> FN0 FN1
check Rule (26, 27, 28) FN -> N FN
check Rule (1), K -> FN0 FN1
TRUE
OntoBahasa [subject=null, nilai=basecamp barameru, kategori=FRASA_NOMINAL, tipe0=[Tdimanall1, Tkettinggian2, Tpeta, Tinformatasi2, Tnotel2, Tmengelola2, T
OntoBahasa [subject=dikelola2, nilai=dikelola oleh siapa, kategori=PRONOMINA_INTERROGATIV, tipe0=[Tbase2], kat0=[FRASA_NOMINAL], tipe1=null, kat1=null,
[basecamp barameru, null (FRASA_NOMINAL)] + [dikelola oleh siapa, null (PRONOMINA_INTERROGATIV)] = [basecamp barameru dikelola oleh siapa (KALIMAT)]
hasil pengecekan bahasa : OntoBahasa [subject=null, nilai=basecamp barameru dikelola oleh siapa, kategori=KALIMAT, tipe0=null, kat0=null, tipe1=null,
waktu untuk cek sintaksis = 1669
query hasil = OPTIONAL { ?VAR_2 mount:pmId ?oid . } OPTIONAL { ?VAR_2 mount:mapFile ?omap . } ?VAR_2 a mount:Basecamp . {?VAR_2 rdfs:label ?LB2 .
final QUERY = PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX xsd: <http://www.w3.org/20
waktu diperlukan generate jawaban = 26
waktu diperlukan kirim data = 28
waktu diperlukan semua proses = 2425

```

Gambar 6.13 Cuplikan proses untuk *input* berupa kalimat tunggal berpola S-P-O



Gambar 6.14 Cuplikan hasil pencarian untuk *input* berupa kalimat tunggal berpola P-O-S

```

Properties Servers Console Progress
Tomcat v7.0 Server at localhost [Apache Tomcat] /Library/java/javaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (Mar 12, 2015, 8:10:47 PM)
---- Start cek 3 token-----
[dikelola oleh siapa, null, PRONOMINA_INTERROGATIV] , [basecamp, null, NOMINA] , [NAME, barameru, NOMINA]

---- Start cek 3 token-----
[dikelola oleh siapa, null, PRONOMINA_INTERROGATIV] , [basecamp, null, NOMINA] , [NAME, barameru, NOMINA]

---- Start cek 3 token-----
[dikelola oleh siapa, null, PRONOMINA_INTERROGATIV] , [basecamp, null, NOMINA] , [NAME, barameru, NOMINA]

---- Start cek 3 token-----
[dikelola oleh siapa, null, PRONOMINA_INTERROGATIV] , [basecamp, null, NOMINA] , [NAME, barameru, NOMINA]

---- Start cek 3 token-----
[dikelola oleh siapa, null, PRONOMINA_INTERROGATIV] , [basecamp, null, NOMINA] , [NAME, barameru, NOMINA]

---- Start cek 3 token-----
[dikelola oleh siapa, null, PRONOMINA_INTERROGATIV] , [basecamp, null, NOMINA] , [NAME, barameru, NOMINA]

---- Start cek 3 token-----
[dikelola oleh siapa, null, PRONOMINA_INTERROGATIV] , [basecamp, null, NOMINA] , [NAME, barameru, NOMINA]

---- Start cek 2 token-----
[dikelola oleh siapa, null, PRONOMINA_INTERROGATIV] , [basecamp barameru, null, FRASA_NOMINAL]
check Rule (1), K -> FNO FN1
check Rule (3) K -> FNO FN1
check Rule (26, 27, 28) FN -> N FN

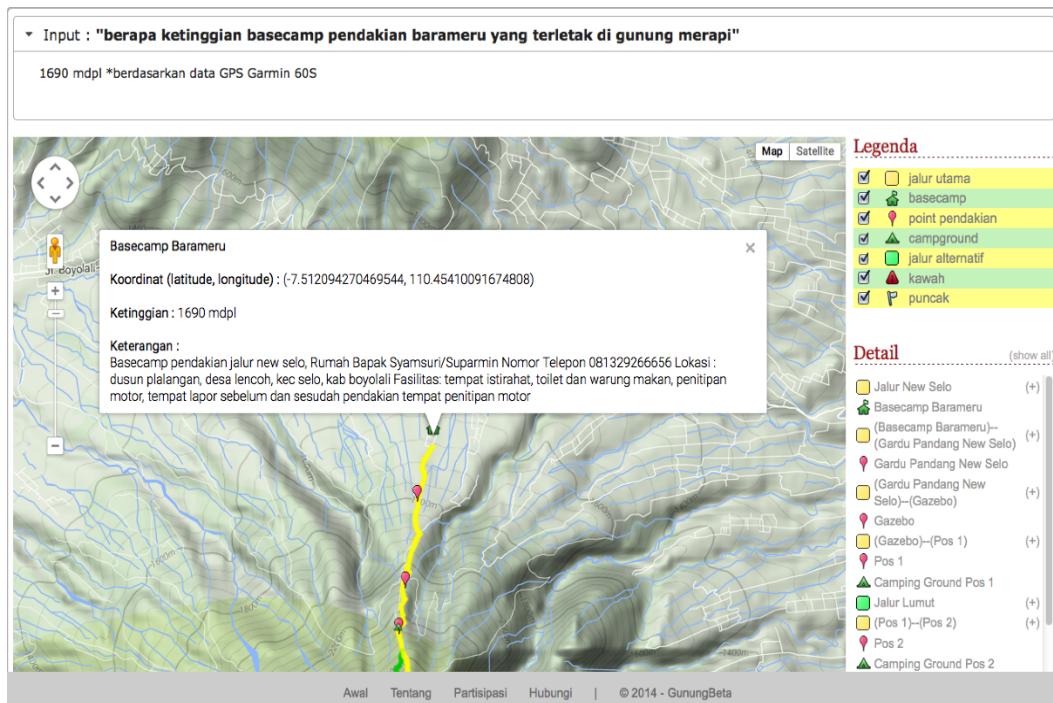
---- Start cek 2 token-----
[dikelola oleh siapa, null, PRONOMINA_INTERROGATIV] , [basecamp barameru, null, FRASA_NOMINAL]
check Rule (1), K -> FNO FN1
check Rule (3) K -> FNO FN1
TRUE
OntoBahasa [subject=dikelola2, nilai=dikelola oleh siapa, kategori=PRONOMINA_INTERROGATIV, tipe0=[Tbase2], kat0=[FRASA_NOMINAL], tipel=null, kat1=null,
OntoBahasa [subject=null, nilai=basecamp barameru, kategori=FRASA_NOMINAL, tipe0=[Tdimanall, Tketinggian2, Tpeta, Tinformasi2, Tnoteil2, Tmengelola2, T
[dikelola oleh siapa, null (PRONOMINA_INTERROGATIV)] + [basecamp barameru, null (FRASA_NOMINAL)] <--> [dikelola oleh siapa basecamp barameru (KALIMAT)]
hasil pengecekan bahasa : OntoBahasa [subject=null, nilai=dikelola oleh siapa basecamp barameru, kategori=KALIMAT, tipe0=null, kat0=null, tipel=null,
waktu untuk cek sintaksis = 638
query hasil = OPTIONAL { ?VAR_2 mount:pmId ?cid . } OPTIONAL { ?VAR_2 mount:mapFile ?omap . } ?VAR_2 a mount:Basecamp . { ?VAR_2 rdfs:label ?LB2 .
final QUERY = PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> PREFIX owl: <http://www.w3.org/2002/07/owl#> PREFIX xsd: <http://www.w3.org/20
waktu diperlukan generate jawaban = 10
waktu diperlukan kirim data = 11
waktu diperlukan semua proses = 806

```

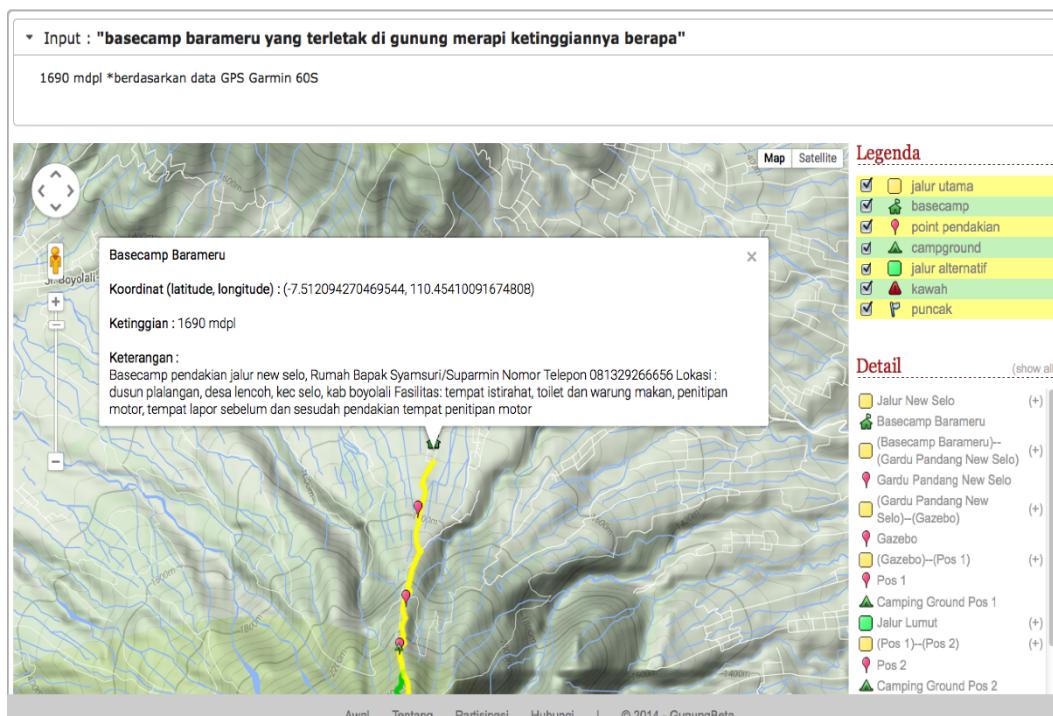
Gambar 6.15 Cuplikan proses untuk *input* berupa kalimat tunggal berpolanya P-O-S

Kalimat juga dapat berpolanya S-P-O-K seperti pada Tabel 6.1 nomor *g1* atau K-S-P-O seperti pada Tabel 6.1 nomor *g2*. Kalimat pada Tabel 6.1 nomor *g1* dan *g2* merupakan kalimat majemuk yang dibentuk dengan menggabungkan beberapa klausa. Kalimat majemuk yang dibentuk dengan menggabungkan beberapa buah unsur klausa dengan hubungan subordinatif disebut juga dengan kalimat majemuk bertingkat Alwi (2003). Cuplikan hasil pencarian untuk melihat kemampuan sistem memahami kalimat majemuk bertingkat dengan pola S-P-O-K atau K-S-P-O dapat dilihat pada Gambar 6.16 dan Gambar 6.17.

Kalimat *input* pada Tabel 6.1 nomor *g1* dan *g2* yaitu “*berapa ketinggian basecamp pendakian barameru yang terletak di gunung merapi*” dan “*basecamp barameru yang terletak di gunung merapi ketinggiannya berapa*” merupakan kalimat tanya majemuk dengan pola berbeda, namun memiliki kesamaan makna yaitu sama-sama menanyakan informasi tingkat ketinggian *basecamp* pendakian Barameru yang terletak di gunung Merapi. Berdasarkan pada Gambar 6.16 dan Gambar 6.17 menunjukkan sistem mampu menyajikan informasi menggunakan *input* kalimat majemuk dengan pola berbeda namun memiliki makna yang sama.

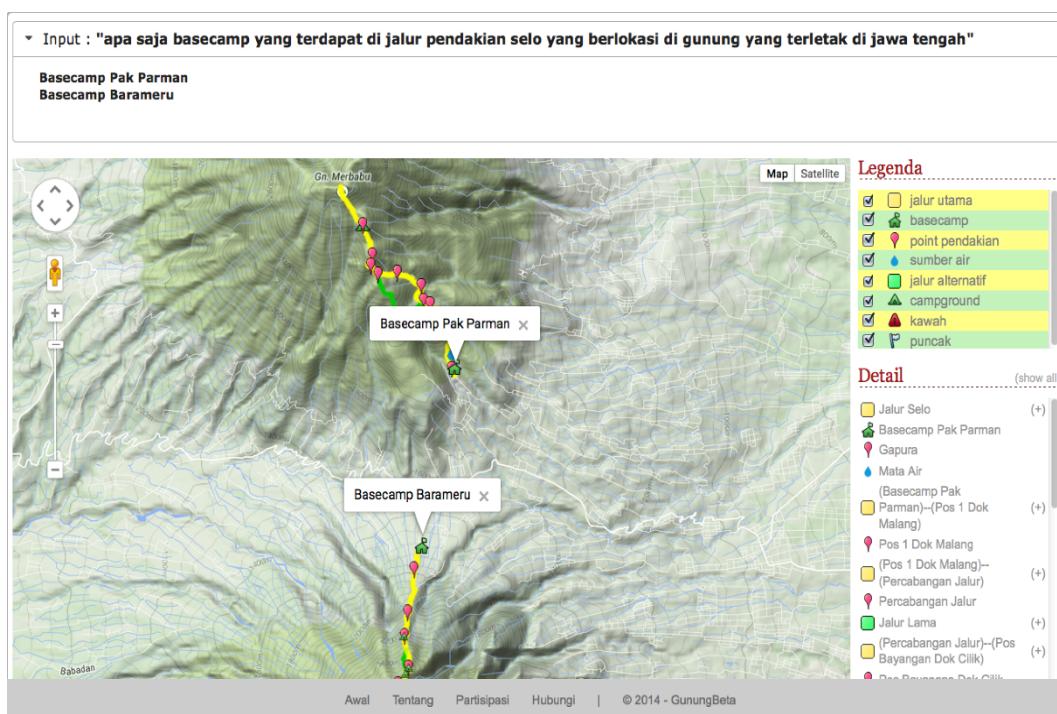


Gambar 6.16 Cuplikan hasil pencarian untuk *input* kalimat majemuk bertingkat berpola S-P-O-K



Gambar 6.17 Cuplikan hasil pencarian untuk *input* kalimat majemuk bertingkat berpola K-S-P-O

Kalimat pada Tabel 6.1 nomor h1 “*apa saja basecamp yang terdapat di jalur pendakian selo yang berlokasi di gunung yang terletak di jawa tengah*”, merupakan contoh kalimat majemuk bertingkat yang jika dipahami dengan menggunakan analisis sintaksis saja akan menghasilkan informasi semantik yang salah. Kemampuan sistem untuk memahami kalimat menggunakan analisis sintaksis dan semantik ditunjukkan menggunakan *input* pada Tabel 6.1 nomor h1 yang dapat dilihat pada Gambar 6.18.



Gambar 6.18 Cuplikan hasil pencarian menggunakan analisis sintaksis dan semantik

6.1.4 Kemampuan sistem mendeteksi *input* yang tidak sesuai dengan kaidah tata bahasa Indonesia

Kemampuan sistem mendeteksi kesalahan *input* dibagi ke dalam tiga kategori, yaitu: kemampuan mendeteksi *input* yang salah secara sintaksis, kemampuan mendeteksi *input* dapat diterima secara sintaksis namun tidak dapat diterima secara semantik, dan kemampuan mendeteksi *input* yang penyusun kalimatnya tidak lengkap. Apabila *input* tidak dapat diterima oleh sistem berdasarkan ketiga kategori tersebut, sistem akan memberikan *output* berupa pesan-pesan kesalahan.

Kemampuan sistem untuk mendeteksi kesalahan sintaksis ditunjukkan menggunakan *input* pada Tabel 6.1 nomor *i1* “*letaknya dimana gunung merapi*”. Cuplikan pesan kesalahan untuk *input* pada Tabel 6.1 nomor *i1* dapat dilihat pada Gambar 6.19. Cuplikan proses pemahaman sistem untuk *input* pada Tabel 6.1 nomor *i1* dapat dilihat pada Gambar 6.20.



Awal Tentang Partisipasi Hubungi | © 2014 - GunungBeta

Gambar 6.19 Cuplikan pesan untuk kesalahan sintaksis

The screenshot shows a Java console window titled 'Tomcat v7.0 Server at localhost [Apache Tomcat] /Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home/bin/java (Mar 15, 2015, 11:29:58 PM)'. The log output is as follows:

```

----- Start cek 3 token-----
[dimana, null, PRONOMINA_INTEROGATIV] , [gunung, null, NOMINA] , [NAME, merapi, NOMINA]

----- Start cek 3 token-----
[dimana, null, PRONOMINA_INTEROGATIV] , [gunung, null, NOMINA] , [NAME, merapi, NOMINA]

----- Start cek 3 token-----
[dimana, null, PRONOMINA_INTEROGATIV] , [gunung, null, NOMINA] , [NAME, merapi, NOMINA]

----- Start cek 3 token-----
[letaknya, null, NOMINA] , [dimana, null, PRONOMINA_INTEROGATIV] , [gunung merapi, null, FRASA_NOMINAL]

----- Start cek 2 token-----
[dimana, null, PRONOMINA_INTEROGATIV] , [gunung merapi, null, FRASA_NOMINAL]
check Rule (1), K -> FNO FN1
check Rule (3) K -> FNO FN1
TRUE
OntoBahasa [subject=dimana2, nilai=dimana, kategori=PRONOMINA_INTEROGATIV, tipe0=[Tcamp52, Tlokasi2, Tjalur2, Tcamp32, Tcamp31, Tair5
OntoBahasa [subject=null, nilai=gunung merapi, kategori=FRASA_NOMINAL, tipe0=[Ttemp32, Tjalur31, Tketinggian2, Tinformatasi2, Tair31, Tk
[dimana, null (PRONOMINA_INTEROGATIV)] + [gunung merapi, null (FRASA_NOMINAL)] = [dimana gunung merapi (KALIMAT)]
hasil pengecekan bahasa : OntoBahasa [subject=null, nilai=dimana gunung merapi, kategori=KALIMAT, tipe0=null, kat0=null, tipe1=null, k

----- Start cek 3 token-----
[letaknya, null, NOMINA] , [dimana, null, PRONOMINA_INTEROGATIV] , [gunung merapi, null, FRASA_NOMINAL]

----- Start cek 3 token-----
[letaknya, null, NOMINA] , [dimana, null, PRONOMINA_INTEROGATIV] , [gunung merapi, null, FRASA_NOMINAL]

----- Start cek 2 token-----
[letaknya, null, NOMINA] , [dimana gunung merapi, null, KALIMAT]
check Rule (19), K -> FV K
FALSE
inser ke stack -> [OntoBahasa [subject=null, nilai=dimana gunung merapi, kategori=KALIMAT, tipe0=null, kat0=null, tipe1=null, kat1=null
waktu untuk cek sintaksis = 503
Pesan akhir Maaf, Sistem tidak dapat memahami input, karena berdasarkan aturan sintaksis dan simantik yang dimiliki sistem input tidak

```

Gambar 6.20 Cuplikan proses untuk kesalahan sintaksis

Pada Gambar 6.20 menunjukkan bahwa sistem dapat mendeteksi *input* yang tidak sesuai dengan tata bahasa Indonesia. *Input* pada Tabel 6.1 nomor *i1* “*letaknya dimana gunung merapi*” merupakan urutan kata yang tidak dapat diterima secara sintaksis karena nomina “*letaknya*” tidak dapat berada di depan kalimat “*dimana gunung merapi*”, nomina “*letak*” dengan kata ganti benda “*nya*” harus berada di belakang nomina yang diatasinya yaitu “*gunung merapi*”.

Kemampuan sistem untuk mendeteksi *input* yang benar secara sintaksis namun salah secara semantik ditunjukkan dengan menggunakan *input* pada Tabel 6.1 nomor *i2* “*siapa merapi*”. Cuplikan pesan kasalahan untuk *input* Tabel 6.1 nomor *i2* dapat dilihat pada Gambar 6.21.



Gambar 6.21 Cuplikan pesan untuk kesalahan semantik

Gambar 6.21 menunjukkan bahwa sistem mampu mendeteksi *input* yang tidak dapat diterima secara semantik meskipun menurut tata bahasa input dapat diterima secara sintaksis.

Kemampuan sistem mendeteksi *input* jika unsur penyusun kalimat tidak lengkap ditunjukkan dengan menggunakan *input* pada Tabel 6.1 nomor *i3* “*apa saja basecamp yang ada di gunung*”. Cuplikan hasil pemrosesan sistem untuk *input* pada Tabel 6.1 nomor *i3* dapat dilihat pada Gambar 6.22.



Gambar 6.22 Cuplikan pesan kesalahan untuk input yang tidak lengkap

Input pada Tabel 6.1 nomor i3 “*apa saja basecamp yang ada di gunung*” dinyatakan tidak lengkap oleh sistem karena berdasarkan pemahaman sistem pada *input* tidak terdapat kata yang berfungsi sebagai predikat, misalnya verba “terletak” selain itu pada *input* juga tidak dijabarkan nama gunung yang dimaksudkan.

6.2 Pengujian Kuantitatif

Pengujian kuantitatif dilakukan untuk mengetahui jumlah *input* yang dapat dipahami oleh sistem. Pengujian dilakukan dengan menggunakan 100 *input* yang dikumpulkan secara acak dari responden. *Input* dari responden dapat dilihat pada Lampiran C.

Berdasarkan pengujian yang dilakukan sistem mampu memproses 69 *input*, 26 *input* tidak dapat diproses disebabkan karena pada *input* mengandung kata-kata yang tidak dikenali dan mengandung aturan gramatiskal yang pada penelitian ini tidak diimplementasikan. Sedangkan 5 *input* tidak dapat diproses karena tidak sesuai dengan aturan tata bahasa Indonesia baku yang sudah diimplementasikan. *Input* yang tidak dapat diproses oleh sistem dan penyebab mengapa *input* tidak dapat diproses dapat dilihat pada Tabel 6.2.

Tabel 6.2 Input dari responden yang tidak dapat diproses oleh sistem

No	Input	Penyebab <i>input</i> tidak dapat diproses sistem
1	dimana spot yang bagus untuk menikmati sunrise di gunung merapi	kata “spot”, “bagus”, “untuk”, “menikmati” dan “sunrise” tidak dikenali oleh sistem
2	informasi posko di gunung merapi	kata “posko” tidak dikenali oleh sistem
3	bagaimana cara ke gunung merbabu	kata “bagaimana”, “cara” dan “ke” tidak dikenali oleh sistem
4	informasi kendaraan yang ke arah gunung slamet	kata “kendaraan”, “ke” dan “arah” tidak dikenali oleh sistem
5	dengan siapa sebaiknya saya mendaki	kata “dengan”, “sebaiknya”, “saya” dan “mendaki” tidak dikenali oleh sistem
6	ada berapa jalur pendakian pada gunung merapi	kata “pada” tidak dikenali oleh sistem
7	berapakah suhu terendah pada gunung merapi	kata “suhu”, “terendah” dan “pada” tidak dikenali oleh sistem
8	kapan saat terbaik melakukan pendakian ke gunung merapi	kata “kapan”, “saat”, “terbaik”, “melakukan”, “pendakian” dan “ke” tidak dikenali oleh sistem
9	jalur yang aman untuk mendaki gunung merapi saat musim hujan	kata “aman”, “untuk”, “mendaki”, “saat”, “musim” dan “hujan” tidak dikenali oleh sistem
10	jalur tercepat ke gunung merapi	kata “tercepat” dan “ke” tidak dikenali oleh sistem
11	jalur yang paling landai untuk mendaki gunung sumbing	kata “paling”, “landai”, “untuk” dan “mendaki” tidak dikenali oleh sistem
12	jalur mana yang bagus untuk turis jika ingin mendaki gunung merbabu	kata “mana”, “bagus”, “untuk”, “turis”, “jika”, “ingin” dan “mendaki” tidak dikenali oleh sistem
13	kapan sebaiknya mendaki gunung merapi	kata “kapan”, “sebaiknya” dan “mendaki” tidak dikenali oleh sistem
14	berapa lama mendaki gunung merapi	kata “lama” dan “mendaki” tidak dikenali oleh sistem
15	kapan waktu terbaik melakukan pendakian di gunung merapi	kata “kapan”, “waktu”, “terbaik”, “melakukan” dan “pendakian” tidak dikenali oleh sistem
16	bagaimana keamanan jalur pendakian di gunung merapi	kata “bagaimana” dan “keamanan” tidak dikenali oleh sistem
17	siapa orang yang mengurus basecamp barameru	kata “orang” tidak dikenali oleh sistem
18	bagaimana transportasi ke basecamp gunung merapi	kata “bagaimana”, “transportasi” dan “ke” tidak dikenali oleh sistem
19	informasi basecamp jalur wekas lengkap	kata “lengkap” tidak dikenali oleh sistem
20	informasi kendaraan menuju basecamp wekas	kata “kendaraan” dan “menuju” tidak dikenali oleh sistem
21	gunung tertinggi di pulau jawa	kata “tertinggi” dan “pulau” tidak dikenali oleh sistem

Tabel 6.2 (lanjutan)

No	Input	Penyebab <i>input</i> tidak dapat diproses sistem
22	tampilkan foto-foto pendakian gunung merapi	kata “ <i>fotofoto</i> ” dan “ <i>pendakian</i> ” tidak dikenali oleh sistem
23	tampilkan semua informasi jalur pendakian yang ada di gunung merbabu	kata “ <i>semua</i> ” tidak dikenali oleh sistem
24	apa nama kawah gunung merapi	kata “ <i>nama</i> ” tidak dikenali oleh sistem
25	di basecamp mana yang paling menyenangkan	kata “ <i>mana</i> ”, “ <i>paling</i> ” dan “ <i>menyenangkan</i> ” tidak dikenali oleh sistem
26	sisi mana gunung merapi viewnya paling menarik	kata “ <i>sisi</i> ”, “ <i>mana</i> ”, “ <i>viewnya</i> ”, “ <i>paling</i> ” dan “ <i>menarik</i> ” tidak dikenali oleh sistem
27	ketinggiannya gunung merbabu berapa	tidak sesuai dengan aturan tata bahasa Indonesia
28	yang mengurus jalur pendakian wekas siapa	tidak sesuai dengan aturan tata bahasa Indonesia
29	dimana lokasinya puncak gunung merapi	tidak sesuai dengan aturan tata bahasa Indonesia
30	juru kuncinya gunung merapi siapakah	tidak sesuai dengan aturan tata bahasa Indonesia
31	dimana terdapat gunung merapi di jawa tengah	tidak sesuai dengan aturan tata bahasa Indonesia

BAB VII

KESIMPULAN DAN SARAN

7.1 Kesimpulan

1. Penggunaan ontologi Bahasa untuk merepresentasikan pengetahuan di bidang linguistik yang meliputi pengetahuan tentang kata, hubungan kata, persamaan kata (*thesaurus*), kategori kata, fungsi sintaksis dan perilaku semantik memungkinkan sistem untuk menggunakan aturan-aturan tata bahasa Indonesia baku dalam memahami *input* bahasa alami baik secara sintaksis maupun semantik. Selain itu dengan penggunaan ontologi Bahasa sistem juga mampu mendeteksi *input* yang tidak sesuai dengan kaidah tata bahasa Indonesia.
2. Hasil pemahaman *input* bahasa alami yaitu berupa *query SPARQL*, proses pencarian informasi jalur pendakian gunung dilakukan dengan mengeksekusi *query SPARQL* tersebut. Proses pencarian yang didasarkan pada pemahaman input bahasa alami menjadikan sistem mampu menyajikan informasi yang relevan sesuai dengan *input* yang diberikan oleh pengguna selama *input* masih dalam ruang lingkup pengetahuan sistem
3. Penggunaan KML berfungsi untuk mendeskripsikan data-data keruangan (spasial) yang dapat diintegrasikan dengan pengetahuan yang ada pada ontologi *mountaineering*, sehingga sistem mampu menyajikan hasil pencarian ke dalam bentuk peta interaktif.
4. Hasil pengujian secara kuantitatif menunjukkan bahwa sistem mampu memahami *input* yang diambil secara acak dari responden sebesar 69%. Meskipun demikian titik berat pemrosesan bahasa dengan penggunaan ontologi Bahasa yang dikembangkan pada penelitian ini bukan pada banyaknya jumlah *input* yang mampu diproses, tapi pada kualitas pemrosesan bahasa.

5. Fitur *Spelling Checker* mampu memberikan saran perbaikan kata asalkan kata tersebut dikenali oleh sistem yaitu kata-kata yang terdapat di dalam kamus kata.

7.2 Saran

1. Ketidakmampuan sistem dalam memproses *input* disebabkan karena kurangnya pengetahuan dan aturan gramatikal yang dimiliki sistem. Penelitian selanjutnya diharapkan dapat mengembangkan pengetahuan-pengetahuan, yaitu :
 - Ontologi Bahasa untuk satuan bahasa yang mengandung unsur adjektiva, adverbia dan konjungtor "dan". Pengembangan ontologi Bahasa sebaiknya melibatkan orang-orang yang ahli di bidang linguistik.
 - Ontologi *Mountaineering* untuk pengetahuan tentang perlengkapan pendakian dan transportasi kendaraan menuju *basecamp*.
2. Dapat ditambahkan fasilitas konversi data dari format KML ke dalam format GPX (*GPS Exchange Format*) sehingga data-data spasial dapat diunduh dan digunakan ke dalam alat GPS.

DAFTAR PUSTAKA

- Alwi, H., Dardjowidjojo, S., Lapolika, H. dan Moeliono, A. M., 2003, *Tata Bahasa Baku Bahasa Indonesia*, Edisi Ketiga, PT Balai Pustaka (Persero), Jakarta.
- Amborowati, A., 2007, Model Ontologi untuk Informasi Jadwal Kereta Api Menggunakan Protégé, *Seminar Nasional Aplikasi Teknologi Informasi 2007 (SNATI 2007)*, Yogyakarta, 16 juni 2007, Hal.9-12.
- Andri, 2011, Rancang Bangun Online Public Access Catalog (OPAC) Berbasis Web Sematik, *Tesis*, Program Studi S2 Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Gadjah Mada, Yogyakarta.
- Anonim, 2013, Keyhole Markup Language, https://developers.google.com/kml/documentation/kml_tut?hl=it-IT#basic_kml, diakses 12 Desember 2013.
- Antoniou, G. dan van Harmelen, F., 2008, *A semantic web primer*, 2nd Edition, MIT press, Cambridge.
- Apriyandi, A. R., 2011, Analisis Kebutuhan yang Mendukung Keberhasilan Pendakian Gunung, *Skripsi*, Program Studi Ilmu Keolahragaan, Jurusan Pendidikan Kesehatan dan Rekreasi, Fakultas Pendidikan Olahraga dan Kesehatan, Universitas Pendidikan Indonesia, Bandung.
- Azhari dan Sholichah, M., 2006, Model Ontologi untuk Informasi Jadwal Penerbangan Menggunakan Protégé, *Jurnal informatika*, No.1, Vol.7, Hal.J67-J76.
- Barr, A. dan Feigenbaum, E. A., 1981, *The Handbook of Artificial Intelligence Volume I*, William Kaufmann Inc., California, USA.
- Bendi, R. J. K., 2010, Sistem *Question Answering* Sederhana Berbasis Ontologi Sebagai Aplikasi Web Semantik, *Tesis*, Program Studi S2 Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Gadjah Mada, Yogyakarta.
- Berners-Lee, T., 1998, Semantic Web Road map, <http://www.w3.org/DesignIssues/Semantic.html>, diakses 12 April 2013.
- Berners-Lee, T., 2006, Artificial Intelligence and the Semantic Web: AAAI2006 Keynote, <http://www.w3.org/2006/Talks/0718-aaai-tbl/Overview.html>, diakses 12 Oktober 2014.
- Berners-Lee, T., Hendler, J. dan Lassila, O., 2001, The Semantic Web, *Scientific American*, No. 5, Vol.284.

- Brickley, D. dan Guha, R. V., 2004, RDF Vocabulary Description Language 1.0: RDF Schema, <http://www.w3.org/TR/rdf-schema/>, diakses 21 Desember 2012.
- Brietman, K. K., Casanova, M. A. dan Truszkowski, W., 2007, *Semantic Web: Concept, Technologies and Applications*, Springer, London.
- Carter, D. dan Agtrisari, I., 2003, *Desain dan Aplikasi SIG*, PT Elex Komputindo, Jakarta.
- Chaer, A., 2006, *Tata Bahasa Praktis Bahasa Indonesia*, Rineka Cipta, Jakarta.
- Clark, A. N., 1990, *Dictionary of Geography*, The Penguin, Puffin, London.
- Daconta, M., Obrst, L. dan Smith, K., 2003, *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*, Wiley Publishing, Indiana.
- Damljanovic, D. dan Bontcheva, K., 2009, Toward Enhanced Useability of Natural Language Interfaces to Knowledge Bases, *Web 2.0 & Semantic Web Annals of Information Systems*, Vol.6, Hal.105-133.
- Dardjowidjojo, S., Porwo, B. P., Kridalaksana, H., Lalamentik, W. H. C. M., Moeliono, A. M., Ramlan, M., Samsuri, Sudaryanto, Silitonga, M., Tampubolon, D. P. dan Tarigan, H.G, 1988, *Tata Bahasa Baku Bahasa Indonesia*, Balai Pustaka, Jakarta.
- Dardjowidjojo, S., 2010, *Psikolinguistik: Pengantar Pemahaman Bahasa Manusia*, Yayasan Obor Indonesia, Jakarta.
- Davies, J., Studer, R. dan Warren, P., 2006, *Semantic Web Tehnologies Trend and Research in Ontology Base System*, John Wiley & Sons, Chichester.
- Djajasudarma, T. F., 1999, *Semantik I Pengantar ke Arah Ilmu Makna*, PT Refika Aditama, Bandung.
- DuCharme, B., 2011, *Learning SPARQL*, O'Reilly Media, Inc, Sebastopol, CA.
- Dykes, L., dan Tittel, E., 2005, *XML for Dummies, 4th edition*, Wiley Publishing, Indianapolis.
- Fazingga, B. dan Lukasiewicz, T., 2010, Semantic search on the Web, *Semantic Web – Interoperability, Usability, Applicability*. No.1, Hal 1-7.
- Giuseppe, P. dan Domenico, T., 2010, UFOme: An Ontology Mapping System With Strategy Prediction Capabilities, *Science Direct Data & Knowledge Engineering*, No.5, Vol.69, Hal.444-471.
- Green, D., 2006, *Because It's There: The Life of George Mallory*, NPI Media Group, UK.
- Gruber, T., 1993, A translation Approach to Portable Ontology Specifications, *Knowledge Acquisition*, No. 2, Vol. 5, hal. 199-220.
- Gruber, T., 2009, Ontology, In: Ling Liu and M. Tamer Ozsu (Eds.), *The Encyclopedia of Database Systems*, Springer-Verlag.

- Handaya, W. B. T. dan Lestari, D. P., 2011, Implementasi Sistem Pemandu Pendakian Gunung, *Seminar Nasional Teknologi Informasi & Komunikasi Terapan 2011 (Semantik 2011)*, Semarang, 16 April 2011, Hal.1-6.
- Hargo, D. U., 2013, Jumlah Pulau di Indonesia, <http://www.dkn.go.id/site/index.php/ruang-opini/126-jumlah-pulau-di-indonesia>, diakses 22 April 2014.
- Hartati, S. dan Zuliarso, E., 2008, Aplikasi Pengolah Bahasa Alami untuk Query Basisdata XML, *Jurnal Teknologi Informasi Dinamik*, No.2, Vol.13, Hal.168-175.
- Ibrahim, F. I., 2007, Materi Pencinta Alam dan Ke "MAHESA" an, <http://www.scribd.com/doc/13452127/Materi-Pencinta-Alam>, diakses 12 Maret 2015.
- Idzelis, M., 2005, Jazzy: The java open soure spell checker, <http://jazzy.sourceforge.net/>, diakses 4 November 2013.
- Irwansyah, E., Adhinugraha, S. dan Datarawijaya, T., 2011, Pengembangan Sistem Informasi Geografis (SIG) Pada Platform Google untuk Penanggulangan Kebakaran di Jakarta Selatan, *Seminar Nasional Aplikasi Teknologi Informasi 2011 (SNATI 2011)*, Yogyakarta, 17-18 juni 2011, Hal.B7-B11.
- Kao, A. dan Poteet, R., 2007, *Natural Language Processing and Text Mining*, Springer, London.
- Kaufmann, E. dan Bernstein, 2007, How useful are natural language interface to the semantic web for casual end-user?, *Proceedings of the 6th International The Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference*, Vol.4825, Hal.281-294.
- Lapolika, H., 1990, *Klausa Pemerlengkapan dalam Bahasa Indonesia*, Kanisius, Yogyakarta.
- Lassila, O. dan Swick, R. R., 1999, Resource Description Framework (RDF) Model and Syntax Specification, Recommendation, World Wide Web Consortium (W3C): <http://www.w3.org/TR/REC-rdf-syntax/> diakses 4 januari 2013.
- Liddy, E. D., 2001, Natural Language Processing, In Encyclopedia of Library and Information Science, 2nd Edition, Marcel Decker Inc, USA.
- Lim, E. H. Y., Liu, J. N. K. dan Lee, R. S. T., 2011, Knowledge Seeker – Ontology Modelling for Information Search and Management, Springer, London.
- McGuinness, D. L., 2004, Question Answering on the Semantic Web, *IEEE Intelligent Systems*, No.1, Vol.19, Hal.82-85.

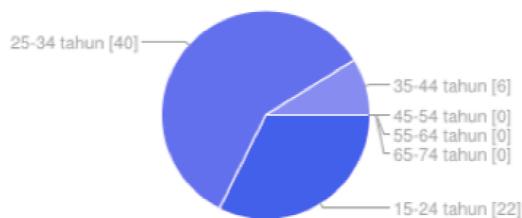
- Muller, R., 2008, Ontologies in Automation, *Tesis*, Vienna University of Technology, Austria.
- Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., dan Swartout, W. R., 1991, Enabling Technology for Knowledge Sharing, *AI Magazine*, No.3, Vol.12, Hal.37-56.
- Noy, N. F. dan McGuinness, D. L., 2001, Ontology Development 101: A Guide to Creating Your First Ontology, http://protégé.stanford.edu/publications/ontology_development/ontology101.pdf, diakses 30 Desember 2012.
- Nurkhamid, M., 2009, Aplikasi Bibliografi Perpustakaan Berbasis Teknologi Web Semantik, *Tesis*, Program Studi S2 Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Gadjah Mada, Yogyakarta.
- Parfet, B., 2009, *Die Trying : One Man's Quest to Conquer the Seven Summits*, Amacom, New York.
- Pirro, G. dan Talia, D., 2010, UfoMe: An ontology mapping system with strategy prediction capabilities, *Data & Knowledge Engineering*, No.5, Vol.69, Hal.444-471.
- Pollock, J. T., 2009, *Semantic Web for Dummies*, Wiley Publishing, Indianapolis.
- Prud'hommeaux, E., Seaborne, A. dan Bristol, 2008, SPARQL Query Language for RDF, <http://www.w3.org/TR/rdf-sparql-query>, diakses 9 Juli 2013.
- Rahutomo, F., 2009, Penerapan Algoritma Weighted Tree Similarity untuk Pencarian Semantik Wikipedia, *Tesis*, Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember, Surabaya.
- Salton, G., 1993, *Introduction to Modern Information Retrieval*, Mcgraw-Hill Book Company, New York.
- Sarno, R., Anistyasari, Y. dan Fitri, R., 2012, *Semantic Search*, Andi Offset, Yogyakarta.
- Sastha, H. B., 2007, *Mountain Climbing for Everybody: Panduan Mendaki Gunung*, Hikmah (PT Mizan Publik), Jakarta.
- Sumitro, 1997, *Buku Pedoman Berolahraga Panjat Tebing*, Gramedia Pustaka Utama, Jakarta.
- Suprihadi, 2005, Penggunaan Bahasa Alami dalam Pengolahan Citra Digital, *Tesis*, Program Studi S2 Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Gadjah Mada, Yogyakarta.
- Suryawan, I. W. D., 2013, Sistem *Question Answering* Menggunakan Pendekatan Berbasis Pengetahuan, *Tesis*, Program Studi S2 Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Gadjah Mada, Yogyakarta.

- Tala, F. Z., 2003, A Study of Stemming Effect on Information Retrieval in Bahasa Indonesia, *Tesis*, Institute for Logic, Language and Computation, Universiteit van Amsterdam, The Netherlands.
- Udell, S., 2012, Geoxml3: KML processor for the Google Maps JavaScript API V3, <http://code.google.com/p/geoxml3/>, diakses 4 November 2012.
- Wellem, T., 2009, Semantic Web Sebagai Solusi Masalah dalam E-Tourism di Indonesia, *Seminar Nasional Aplikasi Teknologi Informasi 2009 (SNATI 2009)*, Yogyakarta, 20 Juni 2009, Hal.D1-D6.
- Wicaksono, I. A. S., Charibaldi, N. dan Jayadianti. H., 2010, Penerapan Teknologi Semantik Web untuk Menentukan Pilihan Jalur Bis Trans Jogja, *Seminar Nasional Informatika 2010 (semnasIF 2010)*, Yogyakarta, 22 Mei 2010, Hal.D102-D110.
- Yu, L., 2011, *A Developer's Guide to the Semantic Web*, Springer, London.
- Yunita, 2011, Pemanfaatan Semantik Web Rule Language (SWRL) Dalam Prototype Sistem Perencanaan Perjalanan Wisata di Sumatera Selatan, *Tesis*, Program Studi S2 Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Gadjah Mada, Yogyakarta.

LAMPIRAN A

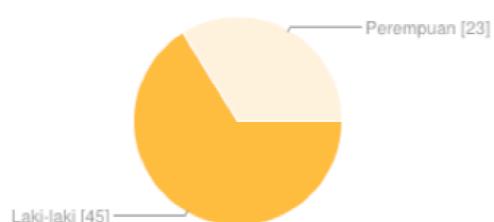
HASIL SURVEI PENDAKI

Usia



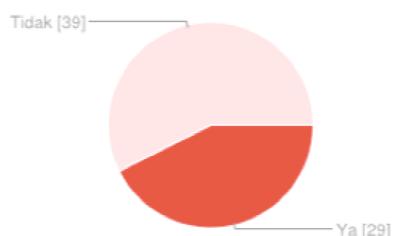
15-24 tahun	22	32.4%
25-34 tahun	40	58.8%
35-44 tahun	6	8.8%
45-54 tahun	0	0%
55-64 tahun	0	0%
65-74 tahun	0	0%

Jenis Kelamin



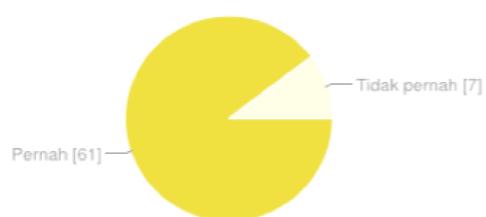
Laki-laki	45	66.2%
Perempuan	23	33.8%

Apakah anda mengikuti organisasi pencinta alam?

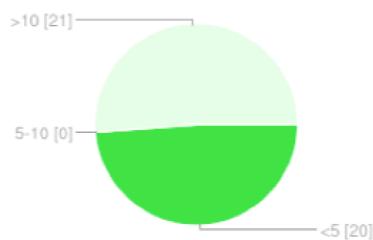


Ya	29	42.6%
Tidak	39	57.4%

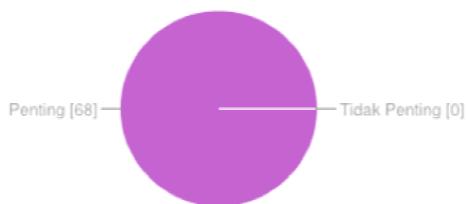
Pernahkah anda mendaki gunung?



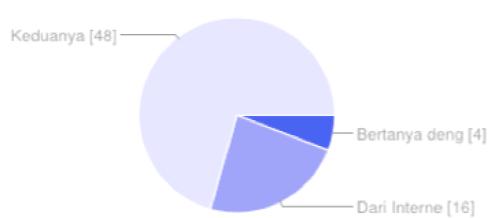
Pernah	61	89.7%
Tidak pernah	7	10.3%

Berapa gunung yang pernah anda daki?

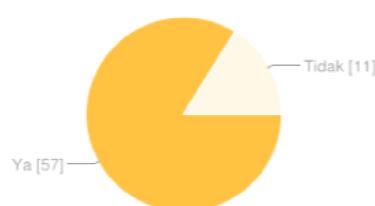
<5	20	29.4%
5-10	0	0%
>10	21	30.9%

Menurut anda, pentingkah mengumpulkan informasi tentang gunung, sebelum melakukan pendakian?

Penting	68	100%
Tidak Penting	0	0%

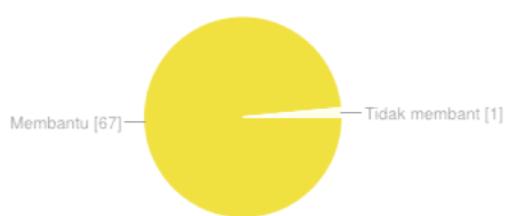
Bagaimana biasanya anda mencari informasi tentang gunung yang akan di daki?

Bertanya dengan pendaki lain	4	5.9%
Dari Internet	16	23.5%
Keduanya	48	70.6%

Apakah anda menemukan masalah ketika melakukan pencarian informasi pendakian melalui internet?

Ya	57	83.8%
Tidak	11	16.2%

Menurut anda, apakah sebuah sistem informasi berbasis peta, yang mana teknologi tersebut dapat menyediakan informasi pendakian yang sesuai dengan keinginan pendaki / minimal mendekati keinginan pendaki, dapat membantu para pendaki dalam mencari informasi pendakian gunung di internet



Membantu	67	98.5%
Tidak membantu	1	1.5%

Apa permasalahan yang biasa anda temukan ketika mencari informasi tentang gunung di internet?

Informasi tidak akurat	21	30.9%
Informasi tidak up to date	32	47.1%
Informasi tidak tersedia	10	14.7%
Membingungkan, terdapat banyak pilihan informasi untuk satu pertanyaan	26	38.2%
Informasi tidak sesuai dengan apa yang saya inginkan	14	20.6%
Lainnya	5	7.4%

LAMPIRAN B

ATURAN GRAMATIKAL

Catatan :

K	=	Kalimat	V	=	Verba
K'	=	Klausa	N	=	Nomina
FN	=	Frasa Nomina	Pi	=	Pronomina
FV	=	Frasa Verba	Nu	=	Numeralia
FPrep	=	Frasa Preposisional	Art	=	Artikula
Arg0	=	argCat0 dan argType0	Arg1	=	argCat1 dan argType1
Arg2	=	argCat2 dan argType2	strip	=	stripCat dan stripType

```

K -> FN0 FN1
<K strip> = <FN0 strip>

<FN0 argCat0> = N
<FN0 Arg1> = 0

<FN1 argCat0> = <FN0 cat>
<FN1 argType0> = <FN0 type>
<FN1 Arg1> = 0
<FN1 strip> = 0

K -> K' FN
<K strip> = <K' strip>

<K' Arg1> = 0
<K' strip> = 0

<FN argCat0> = FN
<FN argType0> = <K' type>
<FN Arg1> = 0
<FN strip> = 0

K -> FN0 FN1
<K strip> = <FN1 strip>

<FN0 argCat0> = <FN1 cat>

```

```

<FN0 argType0> = <FN1 type>
<FN0 Arg1> = 0
<FN0 strip> = 0

<FN1 argCat0> = N
<FN1 Arg1> = 0

K -> FN K'
<K strip> = <K' strip>

<FN argCat0> = FN
<FN argType0> = <K' type>
<FN Arg1> = 0
<FN strip> = 0

<K' Arg1> = 0
<K' strip> = 0

K -> FN FPrep
<K strip> = <FN strip>

<FN argCat0> = N
<FN Arg1> = 0

<FPrep argCat0> = <FN cat>
<FPrep argType0> = <FN type>
<FPrep strip> = 0

K -> K' FPrep
<K strip> = <K' strip>

<K' Arg1> = 0
<K' strip> = 0

<FPrep argCat0> = FN
<FPrep argType0> = <K' type>
<FPrep strip> = 0

K -> FPrep FN
<K strip> = <FN strip>

<FPrep argCat0> = <FN cat>
<FPrep argType0> = <FN type>
<FPrep strip> = 0

```

```

<FN argCat0> = N
<FN Arg1> = 0

K -> FPrep K'
      <K strip> = <K' strip>

      <FPrep argCat0> = FN
      <FPrep argType0> = <K' type>
      <FPrep strip> = 0

      <K' Arg1> = 0
      <K' strip> = 0

K -> FN FV
      <K strip> = <FN strip>

      <FN argCat0> = N
      <FN Arg1> = 0

      <FV argCat0> = <FN cat>
      <FV argType0> = <FN type>
      <FV strip> = 0

K -> K' FV
      <K strip> = <K' strip>

      <K' Arg1> = 0
      <K' strip> = 0

      <FV argCat0> = FN
      <FV argType0> = <K' type>
      <FV strip> = 0

K -> FV FN
      <K strip> = <FN strip>

      <FV argCat0> = <FN cat>
      <FV argType0> = <FN type>
      <FV strip> = 0

      <FN argCat0> = N
      <FN Arg1> = 0

K -> FV K'
```

```

<K strip> = <K' strip>

<FV argCat0> = FN
<FV argType0> = <K' type>
<FV strip> = 0

<K' Arg1> = 0
<K' strip> = 0

K0 -> FN, K1
<K0 strip> = <FN strip>

<FN argCat0> = N
<FN Arg1> = 0
<FN strip> = 0

<K1 strip cat> = <FN cat>
<K1 strip type> = <FN type>

K0 -> K', K1
<K0 strip> = <K' strip>

<K' Arg1> = 0
<K' strip> = 0

<K1 strip cat> = FN
<K1 strip type> = <K' type>

K0 -> FN K1
<K0 strip> = <FN strip>

<FN argCat0> = N
<FN Arg1> = 0
<FN strip> = 0

<K1 strip cat> = <FN cat>
<K1 strip type> = <FN type>

K0 -> K' K1
<K0 strip> = <K' strip>

<K' Arg1> = 0
<K' strip> = 0

<K1 strip cat> = FN

```

```

<K1 strip type> = <K' type>

K0 -> K1, FN
<K0 strip> = <FN strip>

<K1 strip cat> = <FN cat>
<K1 strip type> = <FN type>

<FN argCat0> = N
<FN Arg1> = 0
<FN strip> = 0

K0 -> K1, K'
<K0 strip> = <K' strip>

<K1 strip cat> = FN
<K1 strip type> = <K' type>

<K' Arg1> = 0
<K' strip> = 0

FN -> N
<FN Arg0> = <N Arg0>
<FN Arg1> = <N Arg1>
<FN Arg2> = <N Arg2>
<FN strip> = <N strip>

FN -> Pi
<FN Arg0> = <Pi Arg0>
<FN Arg1> = <Pi Arg1>
<FN Arg2> = <Pi Arg2>
<FN strip> = <Pi strip>

FN -> Nu
<FN Arg0> = <Nu Arg0>
<FN Arg1> = <Nu Arg1>
<FN Arg2> = <Nu Arg2>
<FN strip> = <Nu strip>

FN0 -> N FN1
<FN0 Arg0> = <N Arg0>
<FN0 Arg1> = 0
<FN0 Arg2> = <N Arg2>

```

```

<FN0 strip> = <FN1 strip>

<N argCat1> = <FN1 cat>
<N argType1> = <FN1 type>
<N strip> = 0

<FN1 argCat0> = <N cat>
<FN1 argType0> = <N type>
<FN1 Arg1> = 0

FN0 -> N FN1
<FN0 Arg0> = <N Arg0>
<FN0 Arg1> = FPrep
<FN0 Arg2> = <N Arg2>
<FN0 strip> = <FN1 strip>

<N argCat1> = <FN1 cat>
<N argType1> = <FN1 type>
<N strip> = 0

<FN1 argCat0> = <N cat>
<FN1 argType0> = <N type>
<FN1 Arg1> = 0

FN0 -> N FN1
<FN0 Arg0> = <N Arg0>
<FN0 Arg1> = K'
<FN0 Arg2> = <N Arg2>
<FN0 strip> = <FN1 strip>

<N argCat1> = <FN1 cat>
<N argType1> = <FN1 type>
<N strip> = 0

<FN1 argCat0> = <N cat>
<FN1 argType0> = <N type>
<FN1 Arg1> = 0

FN0 -> FN1 FPrep
<FN0 Arg0> = <FN1 Arg0>
<FN0 Arg1> = 0
<FN0 Arg2> = <FN1 Arg2>
<FN0 strip> = <FN1 strip>

```

```

<FN1 argCat1> = <FPrep cat>

<FPrep argCat0> = <FN1 cat>
<FPrep argType0> = <FN1 type>
<FPrep strip> = 0

FN0 -> FN1 FPrep
<FN0 Arg0> = <FN1 Arg0>
<FN0 Arg1> = K'
<FN0 Arg2> = <FN1 Arg2>
<FN0 strip> = <FN1 strip>

<FN1 argCat1> = <FPrep cat>

<FPrep argCat0> = <FN1 cat>
<FPrep argType0> = <FN1 type>
<FPrep strip> = 0

FN0 -> FN1 K'
<FN0 Arg0> = <FN1 Arg0>
<FN0 Arg1> = 0
<FN0 Arg2> = <FN1 Arg2>
<FN0 strip> = <FN1 strip>

<FN1 argCat1> = <K' cat>

<K' argCat0> = <FN1 cat>
<K' argType0> = <FN1 type>
<K' strip> = 0

FV -> V
<FV Arg0> = <V Arg0>
<FV Arg1> = <V Arg1>
<FV Arg2> = <V Arg2>
<FV strip> = <V strip>

<V Arg1> = 0

FV -> V FN
<FV Arg0> = <V Arg0>
<FV Arg1> = 0
<FV Arg2> = <V Arg2>
<FV strip> = <FN strip>

```

```

<V argCat1> = <FN cat>
<V argType1> = <FN type>
<V Arg2> = 0
<V strip> = 0

<FN argCat0> = N
<FN Arg1> = 0
<FN strip> = 0

FV -> FN V
<FV Arg0> = <V Arg0>
<FV Arg1> = 0
<FV Arg2> = <V Arg2>
<FV strip> = <FN strip>

<FN argCat0> = N
<FN Arg1> = 0
<FN strip> = 0

<V Arg1> = 0
<V strip cat> = <FN cat>
<V strip type> = <FN type>

FV0 -> FV1 FN
<FV0 Arg0> = <FV1 Arg0>
<FV0 Arg1> = <FV1 Arg1>
<FV0 Arg2> = <FV1 Arg2>
<FV0 strip> = <FV1 strip>

<FV1 strip> = 0

<FN argCat0> = <FV1 cat>
<FN argType0> = <FV1 type>
<FN Arg1> = 0
<FN strip> = 0

FV0 -> FV1 FN
<FV0 Arg0> = <FV1 Arg0>
<FV0 Arg1> = <FV1 Arg1>
<FV0 Arg2> = <FV1 Arg2>
<FV0 strip> = <FV1 strip>

<FV1 strip> = 0

```

```

<FN argCat0> = <FV1 cat>
<FN argType0> = <FV1 type>
<FN Arg1> = 0
<FN strip> = 0

FV0 -> FV1 FPrep
<FV0 Arg0> = <FV1 Arg0>
<FV0 Arg1> = <FV1 Arg1>
<FV0 Arg2> = <FV1 Arg2>
<FV0 strip> = <FV1 strip>

<FV1 strip> = 0

<FPrep argCat0> = <FV1 cat>
<FPrep argType0> = <FV1 type>
<FPrep strip> = 0

FV0 -> FV1 FPrep
<FV0 Arg0> = <FV1 Arg0>
<FV0 Arg1> = <FV1 Arg1>
<FV0 Arg2> = <FV1 Arg2>
<FV0 strip> = <FV1 strip>

<FV1 strip> = 0

<FPrep argCat0> = <FV1 cat>
<FPrep argType0> = <FV1 type>
<FPrep strip> = 0

FPrep -> Prep FN
<FPrep Arg0> = <Prep Arg0>
<FPrep Arg1> = 0
<FPrep Arg2> = <Prep Arg2>
<FPrep strip> = <FN strip>

<Prep argCat1> = <FN cat>
<Prep argType1> = <FN type>

<FN argCat0> = N
<FN Arg1> = 0

K' -> Art FN
<K' Arg0> = <Art Arg0>
<K' Arg1> = 0

```

```

<K' Arg2> = <Art Arg2>
<K' strip> = <FN strip>

<FN argCat0> = FN
<FN argType0> = <Art type>
<FN Arg1> = 0
<FN strip> = 0

K' -> Art FV
<K' Arg0> = <Art Arg0>
<K' Arg1> = 0
<K' Arg2> = <Art Arg2>
<K' strip> = <FV strip>

<FV argCat0> = FN
<FV argType0> = <Art type>
<FV strip> = 0

K' -> Art FPrep
<K' Arg0> = <Art Arg0>
<K' Arg1> = 0
<K' Arg2> = <Art Arg2>
<K' strip> = <FPrep strip>

<FPrep argCat0> = FN
<FPrep argType0> = <Art type>
<FPrep strip> = 0

```

LAMPIRAN C

INPUT DARI RESPONDEN

1. campground yang ada di jalur wekas
2. tampilkan peta jalur pendakian gunung merapi
3. informasi jalur pendakian gunung merapi
4. basecamp gunung merbabu
5. dimana letak sumber air di gunung merapi
6. gunung merapi
7. basecamp barameru dikelola oleh siapa
8. berapa ketinggian gunung merapi
9. jalur pendakian yang terletak di gunung merapi
10. basecamp pendakian gunung merapi
11. puncak merapi
12. dimana lokasi puncak gunung merapi
13. apa saja jalur pendakian yang ada di gunung merapi
14. ketinggian gunung merapi
15. berapakah panjang jalur pendakian new selo yang ada di gunung merapi
16. pasar bubrah
17. pasar bubrah terletak dimana
18. informasi basecamp di gunung merapi
19. berapa nomor telpon basecamp jalur pendakian yang ada di gunung merapi
20. apa saja jalur pendakian yang terdapat di gunung merbabu
21. siapa maridjan
22. lokasi basecamp jalur tekelan
23. siapa pengurus basecamp jalur tekelan
24. basecamp tekelan lokasinya dimana
25. apa saja puncak yang ada di gunung merbabu
26. apa saja gunung yang terdapat di jawa tengah
27. sumber air yang ada di gunung merbabu

28. apa saja sumber air yang terdapat di jalur pendakian tekelan
29. dimana lokasi basecamp jalur selo gunung merbabu
30. tampilkan informasi pos yang ada di jalur wekas gunung merbabu
31. dimana letak kawah gunung merapi
32. jembatan setan terletak dimana
33. informasi basecamp jalur cuntel
34. lokasi sumber air yang terdapat di jalur wekas
35. apa saja pos pendakian yang terdapat di jalur kopeng
36. dimana lokasi pasar bubrah
37. tampilkan informasi basecamp yang ada di gunung merapi
38. kawah candradimuka
39. informasi jalur pendakian gunung merapi
40. nomor telpon pengelola basecamp jalur new selo
41. kawah mati terletak dimana
42. informasi jalur pendakian new selo
43. berapa ketinggian puncak garuda
44. puncak garuda terdapat dimana
45. apa saja basecamp yang terletak di jalur pendakian yang terletak di gunung merapi
46. informasi jalur di gunung merapi
47. tampilkan informasi jalur new selo gunung merapi
48. berapa panjang jalur new selo
49. informasi basecamp pendaki gunung merapi
50. siapa penanggung jawab basecamp di gunung merapi
51. apa saja titik pendakian yang terdapat di jalur new selo
52. tampilkan informasi campground yang ada di jalur wekas
53. berapa ketinggian basecamp jalur wekas
54. apa saja basecamp yang jalur wekas
55. dimana lokasi campground yang ada di jalur wekas
56. informasi basecamp yang terdapat di jalur wekas
57. berapa nomor telpon basecamp wekas

58. apa saja puncak gunung merbabu
59. tampilkan informasi gunung yang ada di jawa tengah
60. apa saja gunung yang terdapat di provinsi jawa tengah
61. siapa juru kunci gunung merapi
62. tampilkan informasi jalur pendakian yang ada di gunung merbabu
63. informasi jalur selo gunung merbabu
64. dimana lokasi sumber air di jalur cuntel
65. siapa penanggung jawab basecamp jalur cuntel
66. siapa yang mengelola basecamp mitra indah
67. berapa nomor telpon basecamp jalur pendakian selo gunung merbabu
68. basecamp barameru terletak dimana
69. basecamp new selo gunung merapi