



UNIVERSITAS INDONESIA

PENGEMBANGAN PROGRAM INTEGRASI ONTOLOGI DAN *WEB SERVICES* PADA ZOTONIC

SKRIPSI

ANDRI KURNIAWAN

1306382064

**FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JUNI 2017**



UNIVERSITAS INDONESIA

PENGEMBANGAN PROGRAM INTEGRASI ONTOLOGI DAN *WEB SERVICES* PADA ZOTONIC

SKRIPSI

**Diajukan sebagai salah satu syarat untuk memperoleh gelar
Sarjana Ilmu Komputer**

ANDRI KURNIAWAN

1306382064

**FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JUNI 2017**

HALAMAN PERNYATAAN ORISINALITAS

**Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.**

Nama : Andri Kurniawan
NPM : 1306382064
Tanda Tangan :

Tanggal : 5 Juni 2017

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh :

Nama : Andri Kurniawan

NPM : 1306382064

Program Studi : Ilmu Komputer

Judul Skripsi : Pengembangan Program Integrasi Ontologi dan *Web Services* pada Zotonic

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Ilmu Komputer pada Program Studi Ilmu Komputer, Fakultas Ilmu Komputer, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing : Dr. Ade Azurat ()

Penguji : Penguji 1 ()

Penguji : Penguji 2 ()

Ditetapkan di : Depok

Tanggal : 5 Juli 2017

KATA PENGANTAR

Segala puji dan syukur penulis ucapkan atas kehadiran Allah SWT, Tuhan Yang Maha Esa, karena atas rahmat dan karunia-Nya penulis dapat menyelesaikan skripsi yang berjudul "Pengembangan Program Integrasi Ontologi dan *Web Services* pada Zotonic". Pada kesempatan ini penulis ingin mengucapkan terima kasih kepada seluruh pihak yang telah membantu dan mendukung penulis selama proses pengerjaan skripsi ini, dimana berkat dukungan dan doa mereka skripsi ini dapat diselesaikan.

Penulisan skripsi ini ditujukan untuk memenuhi salah satu syarat untuk menyelesaikan pendidikan pada Program Sarjana Ilmu Komputer, Universitas Indonesia. Penulis sadar bahwa dalam perjalanan perkuliahan hingga penulisan skripsi ini, penulis tidak sendirian. Penulis ingin berterima kasih kepada pihak-pihak berikut :

1. Bapak Ade Azurat selaku dosen pembimbing tugas akhir yang telah meluangkan waktunya sepanjang semester ini untuk dapat memberikan arahan, kritik dan saran kepada penulis agar dapat menyelesaikan proses pengerjaan skripsi ini.
2. Bapak Drs. Lim Yohanes Stefanus M.Math., Ph.D selaku dosen pembimbing akademis penulis, yang selalu membantu penulis selama masa perkuliahan.
3. Drs. H. Zulhaspan, MM dan Hj. Masreni Nasution selaku orangtua dari penulis serta Anita Putri dan Akbar Syarif selaku saudara dari penulis yang selalu mendoakan, mendukung serta menjadi motivasi penulis dalam mengerjakan skripsi ini.
4. Kak Afifun yang telah memberikan banyak masukan terkait hal teknis kepada penulis.
5. Lab RSE
6. Nabila Akiti Hara selaku pacar dari penulis yang selalu memberikan motivasi dan mendukung penulis selama pengerjaan skripsi serta menemani penulis melalui *facetime*.
7. Teman-teman PI BPH IMMM UI (Fadhil, Titto, Fandika, Mawan, Dodo, Okky, Devi, Rara, Ami, Rizky, Ime, Popo, Ilham) yang selalu menghibur

penulis kala jenuh dalam mengerjakan skripsi dan seluruh keluarga IMMMSU UI yang telah menjadi keluarga bagi penulis selama masa perkuliahan.

8. Arief Radityo, Arsi Alhafis, dan M. Gibran yang selalu menjadi teman untuk bermain maupun belajar bagi penulis serta membantu penulis selama perkuliahan.
9. Sahabat-sahabat PPN (Abi, Budi, Cia, Dana, Erwin, Fakhry, Irene, Fadly, Fani, Mawan, Mutia, Sufi, Ulup) yang selalu menjadi penghibur bagi penulis setiap saat.
10. Teman-teman CornedIn (Arsi, Zaki, Dimas, Ilham) yang merupakan teman-teman perjuangan untuk proyek yang mengajarkan banyak hal terkait teknis kepada penulis.
11. Kelompok PPL B1 (Akbar, Dimas, Emon, Fajrin, Fathin), Kelompok PPL B2 (Gilang, Falah, Fatah, Nanda, Hamdan) dan Kelompok PPL B3 (Brigita, Gentur, Kowan, Riscel, Muthy) serta Kak Naya yang telah menemani penulis selama satu semester khususnya hari Rabu dan memberikan penulis pandangan baru mengenai *scrum master*.

Akhir kata, penulis berharap semoga Allah SWT dapat membalas kebaikan yang diberikan oleh orang-orang terdekat penulis dan penulis berharap karya yang penulis buat dapat membantu dan bermanfaat bagi pengembangan ilmu pengetahuan selanjutnya.

Depok, 5 Juni 2017

Andri Kurniawan

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama : Andri Kurniawan
NPM : 1306382064
Program Studi : Ilmu Komputer
Fakultas : Ilmu Komputer
Jenis Karya : Skripsi

demikian demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (Non-exclusive Royalty Free Right)** atas karya ilmiah saya yang berjudul:

Pengembangan Program Integrasi Ontologi dan *Web Services* pada Zotonic

berserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 5 Juni 2017
Yang menyatakan

(Andri Kurniawan)

ABSTRAK

Nama : Andri Kurniawan
Program Studi : Ilmu Komputer
Judul : Pengembangan Program Integrasi Ontologi dan *Web Services* pada Zotonic

Abstrak INA

Kata Kunci:

ABS, Adaptor, SPL, *Web Service*, Zotonic

ABSTRACT

Name : Andri Kurniawan
Program : Computer Science
Title : Development of Ontology and Web Services Integration Program
on Zotonic

Abstract in Eng

Keywords:
one,two,three

DAFTAR ISI

HALAMAN JUDUL	i
LEMBAR PERNYATAAN ORISINALITAS	ii
LEMBAR PENGESAHAN	iii
KATA PENGANTAR	iv
LEMBAR PERSETUJUAN PUBLIKASI ILMIAH	vi
ABSTRAK	vii
Daftar Isi	ix
Daftar Gambar	xi
Daftar Tabel	xii
Daftar Kode	xiii
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Perumusan Masalah	2
1.3 Tujuan Penelitian	2
1.4 Ruang Lingkup Penelitian	3
1.5 Sistematika Penulisan	3
2 TINJAUAN PUSTAKA	5
2.1 Ontologi	5
2.2 Zotonic	5
2.3 <i>Template Engine</i>	5
2.4 <i>Web Services</i>	5
2.5 <i>Software Product Line</i>	5
2.6 X si sesuatu	5
2.6.1 Pengertian X	5
2.6.2 Klasifikasi X	6
2.7 <i>Section in Eng</i>	7
2.7.1 Pengertian <i>Section in Eng</i>	7
2.7.2 Next Subsection <i>Section in Eng</i>	7
2.8 <i>Keatas lagi</i>	7
2.8.1 <i>Masuk lagi</i>	8

	x
3 RANCANGAN	9
3.1 Rancangan Integrasi Ontologi dan <i>Web Service</i>	9
3.2 Desain <i>Adaptor</i>	10
4 IMPLEMENTASI	12
4.1 Implementasi <i>Adaptor</i>	12
4.1.1 Pemanggilan <i>Adaptor</i> Melalui <i>Template Engine</i>	13
4.1.2 Pemanggilan <i>Adaptor</i> Melalui Model	15
4.1.3 Implementasi Fungsi <i>lookup_rules</i>	16
4.1.4 Implementasi Fungsi <i>validate_params</i>	18
4.1.5 Implementasi Fungsi <i>fetch_data</i>	18
4.2 Implementasi <i>Refactoring Business Logic</i>	20
5 HASIL	21
5.1 Implementasi <i>Cluster</i>	21
5.1.1 Instalasi <i>Frontend</i>	21
5.1.2 Konfigurasi	23
5.1.2.1 semakin ke dalam	24
5.2 Pengujian	24
5.2.1 Kasus Uji	24
5.2.2 Kasus Uji	25
6 PENUTUP	26
6.1 Kesimpulan	26
6.2 Saran	26
LAMPIRAN	1
Lampiran 1 : Kode Sumber Model ABS	2
Lampiran 2 : Kode Sumber rules	4
Lampiran 8 : UAT dan Kuesioner	5

DAFTAR GAMBAR

2.1	Contoh masalah yang dikerjakan secara paralel	6
2.2	Arsitektur klasik <i>von Neumann</i>	6
3.1	Rancangan integrasi ontologi dan web service	9
3.2	Rancangan Adaptor	11

DAFTAR TABEL

2.1	Fungsi fundamental MPI	8
4.1	Contoh Tabel	19
4.2	An Example of Rows Spanning Multiple Columns	19
4.3	An Example of Columns Spanning Multiple Rows	19
4.4	An Example of Spanning in Both Directions Simultaneously	19
5.1	Informasi <i>cluster</i> X	21
5.2	Perbandingan Partisi <i>default</i> dan manual	22
1	Tabel UAT dan Kuesioner	6

DAFTAR KODE

3.1	Contoh tabel <i>rules</i>	11
4.1	Struktur tabel <i>rules</i>	12
4.2	Fungsi yang harus diekspor untuk model	12
4.3	Implementasi fungsi <i>m_to_list</i>	13
4.4	Implementasi fungsi <i>m_value</i>	13
4.5	Implementasi fungsi <i>m_find_value</i>	14
4.6	Implementasi fungsi untuk pemanggilan <i>adaptor</i> dari model	16
4.7	Implementasi fungsi <i>lookup_rules</i>	17
4.8	lokasi dari <i>file</i> <i>rules.txt</i>	17
4.9	Implementasi fungsi <i>validate_params</i>	18
4.10	Implementasi fungsi <i>fetch_data</i>	18
5.1	Keluaran output	22
5.2	Keluaran mentah untuk detail <i>job</i>	24
5.3	Potongan skrip submisi <i>job</i> melalui <i>torqace</i>	24
5.4	Potongan <i>Makefile</i> <i>project</i>	25
1	Skrip adaptor <i>m_abs.erl</i>	2
2	Berkas <i>compute.xml</i>	5

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Pada zaman serba teknologi saat ini, kebutuhan akan penggunaan web sebagai sarana berbagi informasi sangat tinggi. Namun, tidak ada sumber daya manusia yang memiliki latar belakang teknis dan merasa sulit untuk membuat sebuah web menjadi sebuah permasalahan yang dihadapi oleh beberapa organisasi di Indonesia terutama organisasi yang non-profit. Selain itu, banyaknya informasi yang terdapat pada web saat ini tidak memiliki hubungan informasi yang terstruktur dan hanya didesain untuk manusia saja sehingga program komputer tidak dapat mengolah informasi tersebut (Berners-Lee, Hendler, dan Lassila, 2001, p. 1). Sehingga pada tahun 2001 dicetuskan ide untuk membuat web semantik oleh Berners-Lee dkk, dimana web semantik akan membuat sebuah struktur untuk konten web sehingga informasi yang terdapat pada web lebih berarti karena dapat diolah oleh program komputer (Berners-Lee, Hendler, dan Lassila, 2001, p. 1). Namun, perkembangan web semantik sendiri tidak seperti yang diharapkan karena mengalami permasalahan. Menurut Rob McCool, Format yang kompleks dan pengguna harus mengorbankan kemudahan ekspresivitas dan membayar biaya yang besar untuk translasi dan perawatan, web semantik tidak akan pernah diadopsi public secara luas (Schoop, de Moor, dan Dietz, 2006).

Untuk mengatasi permasalahan tersebut yang dihadapi oleh web semantik, diciptakan web pragmatis. Tujuan utama dari web pragmatis adalah untuk meningkatkan kolaborasi manusia lebih efektif dengan teknologi yang tepat, seperti sistem untuk negosiasi ontologi, untuk interaksi bisnis berbasis ontologi, dan untuk membangun ontologi pragmatis pada praktik masyarakat (Schoop, de Moor, dan Dietz, 2006). Sehingga web pragmatis dapat melengkapi web semantik untuk berkolaborasi dan meningkatkan kualitas pada level masyarakat. Salah satu perkembangan web semantik pragmatis adalah Zotonic, yaitu sebuah framework sekaligus Content Management System (CMS) yang dibangun di atas bahasa pemrograman erlang dimana zotonic telah mengadopsi konsep web semantik. Kehadiran Zotonic sendiri diharapkan dapat meningkatkan pemanfaatan web semantik dalam proses pembuatan web sehingga informasi yang berada pada web yang dibuat dapat langsung diolah oleh komputer. Tetapi, pengembang perlu mendefinisikan seman-

itik yang akan mereka buat terlebih dahulu sebelum mereka mengembangkannya dalam Zotonic untuk menciptakan sebuah konsistensi. Namun hal ini tentu saja menghambat proses pengembangan karena pengembang membutuhkan waktu yang lebih lama untuk proses translasi dari semantik ke Zotonic.

Untuk membantu para pengembang dalam mentranslasikan semantik ke dalam Zotonic, pada tahun 2016 terdapat sebuah penelitian terkait pembentukan otomatis aplikasi web dengan masukan berupa ontologi, yaitu penelitian terkait pemetaan ontologi yang dihasilkan semantik ke dalam struktur Zotonic (Pangukir el al., 2016). Namun, pada penelitian tersebut pembentukan bussiness logic dari web yang dibentuk masih secara manual. Hal ini karena pada ontologi tidak terdapat bussiness logic. Bussiness logic tersebut dapat diperoleh melalui translasi feature model dari rancangan web tersebut.

Oleh karena itu, untuk menerapkan paradigma Software Product Line (SPL) dimana variasi dan kesamaan akan disusun secara modular dan reusability, diperlukan sebuah program yang akan melakukan pemetaan secara otomatis terhadap bussiness logic dari feature model yang telah dirancang ke dalam Zotonic. Sehingga diharapkan kedepannya, proses pembentukan web berbasis semantik dapat lebih mudah dan cepat agar meningkatkan jumlah web yang berbasis semantik.

1.2 Perumusan Masalah

Berdasarkan latar belakang tersebut, penelitian ini akan mencoba untuk menjawab beberapa pertanyaan penelitian, yaitu

1. Bagaimana proses pemetaan dari ontologi kepada *web service*?
2. Apakah dapat dikembangkan sebuah program *adaptor* yang akan melakukan pemetaan dari zotonic kepada *web service* secara otomatis?

1.3 Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk mengembangkan sebuah program yang dapat menghubungkan antara *web services* dan Zotonic sehingga proses pembuatan *site* pada Zotonic dapat memiliki tingkat adaptasi yang baik terhadap perubahan yang terjadi. Harapannya melalui program yang dibuat, pengembang dapat tetap fokus untuk *maintain* data dan mengembangkan aplikasi, karena data sudah terjadi melalui ontologi.

1.4 Ruang Lingkup Penelitian

Ruang lingkup penelitian ini antara lain:

1. Analisis pemetaan adaptor untuk menghubungkan antara ontologi dan *web service*
2. Perancangan sistem yang dapat menghubungkan antara ontologi dan *web service* melalui zotonic.
3. *Refactoring* pembuatan *business logic* yang sudah ada dengan memanfaatkan sistem yang dibuat

1.5 Sistematika Penulisan

Sistematika penulisan laporan adalah sebagai berikut:

- Bab 1 PENDAHULUAN

Bab 1 berisi tentang informasi terkait penelitian yang dilakukan oleh penulis, dimana bab ini terdiri atas 5 subbab, yaitu latar belakang yang akan membahas kenapa dilakukan penelitian ini, perumusan masalah yang akan membahas masalah yang akan diteliti oleh penulis, tujuan penelitian yang akan menjelaskan kegunaan dari penelitian ini, ruang lingkup penelitian yang akan membahas mengenai batasan dari penelitian yang dilakukan serta sistematika penulisan.

- Bab 2 TINJAUAN PUSTAKA

Bab 2 berisi mengenai teori-teori yang digunakan

- Bab 3 RANCANGAN

Bab 3 berisi penjelasan mengenai rancangan dari sistem yang akan diimplementasikan, dimana bab ini terdiri atas 2 subbab, yaitu rancangan integrasi ontologi dan web service yang akan membahas secara garis besar bagaimana program ini akan berjalan secara keseluruhan serta rancangan *adaptor* yang akan membahas bagaimana *adaptor* yang dibuat nantinya akan bekerja.

- Bab 4 IMPLEMENTASI

Bab 4 berisi penjelasan mengenai implementasi yang dilakukan oleh penulis untuk membuat *adaptor*.

- Bab 5 HASIL

Bab 5 berisi hasil eksperimen yang telah dilakukan oleh penulis.

- Bab 6 PENUTUP

Bab 6 berisi kesimpulan yang didapat dari penelitian serta saran yang diajukan untuk penelitian berikutnya.

BAB 2

TINJAUAN PUSTAKA

Bab ini berisi Pada sub-bab 2.6 akan dijelaskan dasar-dasar ...

2.1 Ontologi

2.2 Zotonic

2.3 *Template Engine*

2.4 *Web Services*

2.5 *Software Product Line*

2.6 X si sesuatu

Dokumen \LaTeX sangat mudah, seperti halnya membuat dokumen teks biasa. Ada beberapa perintah yang diawali dengan tanda `'\'`. Seperti perintah `\\` yang digunakan untuk memberi baris baru. Perintah tersebut juga sama dengan perintah `\newline`. Pada bagian ini akan sedikit dijelaskan cara manipulasi teks dan perintah-perintah \LaTeX yang mungkin akan sering digunakan. Jika ingin belajar hal-hal dasar mengenai \LaTeX , silahkan kunjungi:

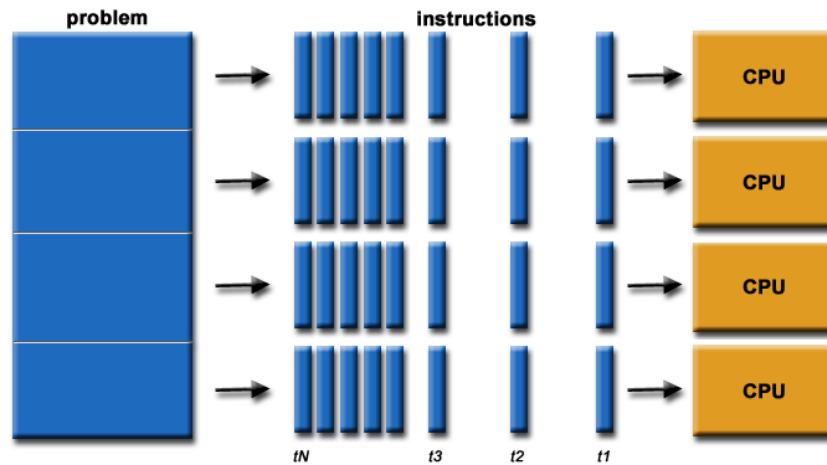
- <http://frodo.elon.edu/tutorial/tutorial/>, atau
- <http://www.maths.tcd.ie/~dwilkins/LaTeXPrimer/>

2.6.1 Pengertian X

Setiap gambar dapat diberikan caption dan diberikan label. Label dapat digunakan untuk menunjuk gambar tertentu. Jika posisi gambar berubah, maka nomor gambar juga akan diubah secara otomatis. Begitu juga dengan seluruh referensi yang menunjuk pada gambar tersebut.

Contoh sederhana adalah Gambar 2.1. Silahkan lihat code \LaTeX dengan nama `bab2-landasan-teori.tex` untuk melihat kode lengkapnya. Harap diingat bahwa caption untuk gambar selalu terletak dibawah gambar.

Dibawah adda figure, jangan lupa dimention dengan 2.1.



Gambar 2.1: Contoh masalah yang dikerjakan secara paralel

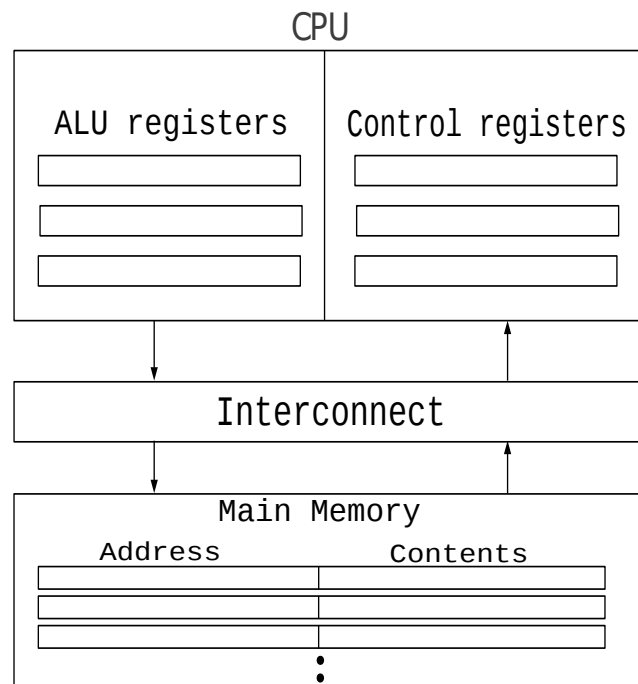
Sumber gambar: (?)

2.6.2 Klasifikasi X

Figure dalam enum dan dua sitasi sekaligus (??) :

1. ***Bold Italic***

Penjelasan..... Untuk gambarannya dapat dilihat di Gambar 2.2.



Gambar 2.2: Arsitektur klasik von Neumann

Sumber gambar terinspirasi dari: (?)

2. *Sesuatunya*

Penjelasan.....

2.7 *Section in Eng*

Hal pertama yang mungkin ditanyakan adalah bagaimana membuat huruf tercetak tebal, miring, atau memiliki garis bawah. Pada Texmaker, Anda bisa melakukan hal ini seperti halnya saat mengubah dokumen dengan LO Writer. Namun jika tetap masih tertarik dengan cara lain, ini dia:

- **Bold**
Gunakan perintah `\textbf{}` atau `\bo{}`.
- *Italic*
Gunakan perintah `\textit{}` atau `\f{}`.
- Underline
Gunakan perintah `\underline{}`.
- Overline
Gunakan perintah `\overline{}`.
- *superscript*
Gunakan perintah `\{}`.
- *subscript*
Gunakan perintah `_{}`.

Perintah `\f` dan `\bo` hanya dapat digunakan jika package `uithesis` digunakan.

2.7.1 *Pengertian Section in Eng*

2.7.2 *Next Subsection Section in Eng*

2.8 *Keatas lagi*

Contoh cite yang ga ada ?. Cite author ?,cite tahun ?, cite mention ?, dan cite di akhir kalimat (?).

2.8.1 Masuk lagi

Footnote example nih : MPICH ¹, LAM/MPI ², dan OpenMPI ³ (?). MPI-3 sedang dalam tahap perencanaan ⁴. Fungsi-fungsi tersebut berada di tabel 2.1. (Contoh tabel).

Tabel 2.1: Fungsi fundamental MPI

No.	Nama Fungsi	Penjelasan
1	MPI_Init	Memulai kode MPI
2	MPI_Finalize	Mengakhiri kode MPI
3	MPI_Comm_size	Menentukan jumlah proses
4	MPI_Comm_rank	Menentukan label proses
5	MPI_Send	Mengirim pesan
6	MPI_Recv	Menerima pesan

Sumber tabel: taro sitasi disini, if i were u

¹<http://www.mpich.org/>

²<http://www.lam-mpi.org/>

³www.open-mpi.org

⁴http://meetings.mpi-forum.org/MPI_3.0_main_page.php

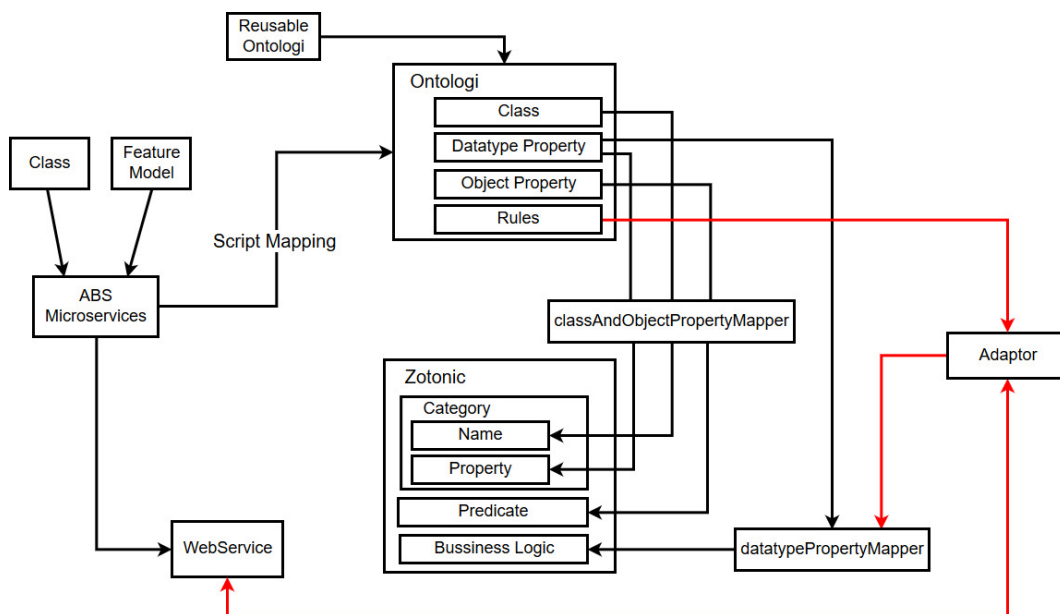
BAB 3

RANCANGAN

Dari permasalahan yang sudah didefinisikan, maka akan dibuat sebuah program yang akan melakukan integrasi ontologi dan *web service* dengan memanfaatkan Zotonic. Sebelum memulai pembuatan program, perlu dirancang bagaimana program ini akan berjalan. Pada bab ini, akan dibahas mengenai rancangan integrasi ontologi dan *web service* yang akan menggambarkan secara keseluruhan bagaimana program akan bekerja serta hal lainnya yang berhubungan dengan program dan juga mengenai rancangan adaptor yang akan menggambarkan secara dalam bagaimana program dapat mengakses *web service* dan terhubung dengan ontologi.

3.1 Rancangan Integrasi Ontologi dan Web Service

Agar dapat menghasilkan program yang bertujuan untuk melakukan integrasi ontologi dan *web services* maka perlu dirancang secara keseluruhan bagaimana program ini akan bekerja. Sesuai dengan kebutuhan dari program, maka integrasi ini akan dilakukan pada sebuah *framework* sekaligus CMS Zotonic, yang telah dimodifikasi agar dapat menerima masukan berupa ontologi. Secara garis besar, berikut merupakan gambaran cara program akan bekerja.



Gambar 3.1: Rancangan integrasi ontologi dan web service

Seperti yang dapat dilihat pada Gambar 3.1, *Class* dan *Feature Model* akan diproses menggunakan program translasi yang akan menghasilkan *ABS Microservices* dimana ini telah dilakukan penelitian sebelumnya sehingga hal ini bukan merupakan bagian dari penelitian penulis. Setelah dihasilkan sebuah *ABS Microservices* dari proses translasi tersebut, maka *ABS Microservices* akan menghasilkan sebuah *web service* yang dapat digunakan oleh program lainnya. Dalam penelitian ini, *web services* yang dihasilkan oleh *ABS Microservices* akan digunakan oleh *Adaptor* yang akan terhubung dengan *Zotonic*. Selain digunakan untuk menghasilkan sebuah *web service*, *ABS Microservices* akan digunakan untuk menghasilkan sebuah ontologi menggunakan sebuah *script* yang akan melakukan pemetaan dari *class* dan *feature model* menjadi *class*, *datatype property*, *object property* dan *rules* pada ontologi. Untuk pemetaan ini sendiri tidak termasuk dalam penelitian ini karena penulis hanya akan menggunakan sebuah ontologi yang telah dirancang sebelumnya. Nantinya melalui proses pemetaan ini, akan dihasilkan sebuah ontologi yang bersifat *reusable*.

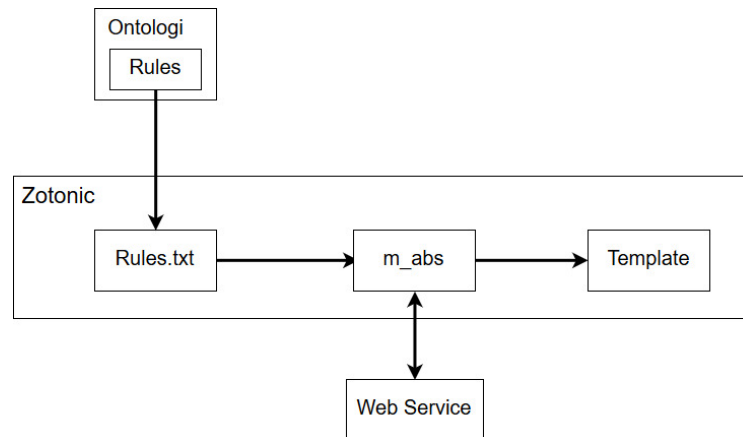
Menggunakan ontologi yang telah dirancang sebelumnya, ontologi tersebut akan dipetakan ke dalam struktur dari *zotonic* yang akan digunakan. Proses pemetaan dari ontologi ke dalam *zotonic* ini sendiri telah dilakukan pada penelitian sebelumnya oleh *Bravyto* dimana setiap *class* dari ontologi akan dipetakan menjadi nama dari kategori pada *zotonic* menggunakan *script classAndObjectPropertyMapper.sh*. Selain melakukan pemetaan pada *class*, *script* tersebut juga akan melakukan pemetaan *datatype property* pada ontologi menjadi *property* dari kategori pada *zotonic* serta pemetaan *object property* pada ontologi menjadi *predicate* pada *zotonic*. Pada penelitian tersebut, dilakukan juga pemetaan dari *datatype property* menjadi *business logic* menggunakan *script datatypePropertyMapper.sh*.

Namun pada penelitian tersebut, pembuatan *business logic* masih bersifat manual yang langsung ditaruh pada *script datatypepropertyMapper.sh*. Pada penelitian ini, penulis akan membuat sebuah adaptor yang akan memanggil *web service* sehingga *business logic* pada *zotonic* akan lebih fleksibel karena memanfaatkan *web service* serta memberikan kemudahan bagi *developer* dalam hal melakukan modifikasi.

3.2 Desain Adaptor

Adaptor merupakan sebuah *script* yang akan memanggil *web service* sehingga dapat digunakan untuk melakukan pemrosesan *business logic* pada *zotonic*. Bagaimana adaptor akan bekerja sehingga menghasilkan *business logic* dapat dilihat pada gam-

bar berikut ini.



Gambar 3.2: Rancangan Adaptor

Seperti yang dapat dilihat pada Gambar 3.2, *rules* yang terdapat pada ontologi akan dilakukan pemetaan menjadi tabel *rules* yang akan disimpan pada file *rules.txt* seperti yang terdapat pada kode 3.1. Namun, proses pemetaan ini berada diluar dari topik penelitian penulis sehingga untuk keperluan penelitian ini maka penulis membuat sebuah tabel *rules* secara manual untuk mengganti proses pemetaan tersebut.

Kode 3.1: Contoh tabel *rules*

```

{
  "createProgram": ["http://54.169.128.6:8080/abs/program/create"
    , 2],
  "createDonation": ["http://54.169.128.6:8080/abs/donation/
    create", 4],
  "updateProgram": ["http://54.169.128.6:8080/abs/program/update"
    , 2],
  "updateDonation": ["http://54.169.128.6:8080/abs/donation/
    update", 4],
  "deleteProgram": ["http://54.169.128.6:8080/abs/program/delete"
    , 1],
  "deleteDonation": ["http://54.169.128.6:8080/abs/donation/
    delete", 1],
  "totalDonation" : ["http://54.169.128.6:8080/abs/program/total-
    donation", 1]
}
  
```

Setelah terbentuk tabel *rules* pada file *rules.txt*, maka ketika model *abs* yang terdapat pada file dijalankan pada *template engine*, model *abs* akan membaca *rules* untuk mengetahui *endpoint* yang akan dijalankan pada proses pemanggilan *web service* serta untuk melakukan pengecekan apakah jumlah parameter yang dimasukkan telah sesuai dengan jumlah parameter yang diterima oleh *web service* atau tidak.

BAB 4

IMPLEMENTASI

Pada bab ini dijelaskan setiap hal yang dilakukan oleh penulis untuk melakukan implementasi terhadap rancangan yang telah dibuat pada bab sebelumnya. Penulis melakukan implementasi *adaptor* yang akan dipanggil melalui *template engine* dan juga melalui model lainnya. Selain itu, penulis juga melakukan *refactoring business logic* dengan memanfaatkan *adaptor* yang sudah diimplementasikan.

4.1 Implementasi *Adaptor*

Sebelum melakukan implementasi *adaptor*, perlu dibuat terlebih dahulu tabel *rules* yang akan dibaca oleh *adaptor*. Namun, karena belum adanya mekanisme pemetaan secara langsung dari *rules* yang dimiliki oleh ontologi ke dalam tabel *rules* maka penulis membuat tabel *rules* secara manual. Tabel *rules* tersebut berbentuk *json* dimana strukturnya sebagai berikut

Kode 4.1: Struktur tabel *rules*

```
{
  nama fungsi : [endpoint, jumlah parameter]
}
```

Setelah tabel *rules* selesai dibuat sesuai dengan struktur pada kode 4.1, tabel *rules* tersebut disimpan ke dalam sebuah file yang bernama *rules.txt* dan ditaruh pada *root* dari *zotonic*. Untuk dapat menjalankan fungsi *adaptor* yang diinginkan, penulis membuat sebuah model baru pada *zotonic* dengan nama *m_abs* pada folder *root/src/models*. Untuk membuat sebuah model pada *zotonic*, *zotonic* mengharuskan setiap model untuk mengeksport beberapa fungsi yang dimiliki oleh *zotonic* yaitu *m_find_value*, *m_to_list*, dan *m_value* agar fungsi tersebut dapat digunakan melalui *template engine*.

Kode 4.2: Fungsi yang harus diekspor untuk model

```
-export ([
  m_find_value/3,
  m_to_list/2,
  m_value/2,
]) .
```

Seperti yang dapat dilihat pada kode 4.2, agar fungsi tersebut dapat dijalankan pada *template engine*, tentu harus diimplementasi sesuai dengan kebutuhan. Sesuai dengan kebutuhannya agar *adaptor* dapat dipanggil melalui *template engine*, maka perlu didefinisikan terlebih dahulu bagaimana nantinya *adaptor* dipanggil. Pada penelitian ini, penulis mendefinisikan untuk pemanggilan *adaptor* pada *template engine* dilakukan dengan membuat perintah `m.abs.namaFungsi[{query param=value}]`. Implementasi dari fungsi `m_find_value` yang digunakan untuk melakukan pencocokan pola akan dijelaskan pada bagian berikutnya. Untuk fungsi `m_to_list` karena pada kasus ini tidak digunakan jadi cukup diimplementasikan seperti kode 4.3 berikut

Kode 4.3: Implementasi fungsi `m_to_list`

```
m_to_list(_, _Context) ->
[].
```

Sama seperti fungsi `m_to_list`, fungsi `m_value` juga tidak dibutuhkan pada implementasi *adaptor* sehingga dapat diimplementasikan seperti kode 4.4 berikut

Kode 4.4: Implementasi fungsi `m_value`

```
m_value(_, _Context) ->
undefined.
```

4.1.1 Pemanggilan *Adaptor* Melalui *Template Engine*

Setelah didefinisikan format pemanggilan *adaptor* pada *template engine*, maka akan diimplementasikan pencocokan pola pada proses pemanggilan *adaptor* dengan menggunakan fungsi `m_find_value`. Berikut adalah implementasi dari fungsi `m_find_value`

Kode 4.5: Implementasi fungsi m_find_value

```
% this method to handle call api from template
-spec m_find_value(Key, Source, Context) -> #m{} | undefined |
  any() when
    Key:: integer() | atom() | string(),
    Source:: #m{},
    Context:: #context{}.

m_find_value(Type, #m{value=undefined} = M, _Context) ->
  M#m{value=[Type]};

m_find_value({query, Query}, #m{value=Q} = _, _Context) when
  is_list(Q) ->
    [Key] = Q,
    [Url, Param] = lookup_rules(Key),
    case validate_params(Param, Query) of
      false ->
        [{error, "Num of Params not same"}];
      true ->
        {DecodeJson} = fetch_data(binary_to_list(Url), jiffy:encode
          ({Query})),
        lager:info("ABS result : ~p", [DecodeJson]),
        proplists:get_value(<<"data">>, DecodeJson)
    end;

% Other values won't be processed
m_find_value(_, _, _Context) ->
  undefined.
```

Pada kode 4.5, tahap awal untuk mengimplementasikan fungsi `m_find_value` adalah membuat sebuah *specifications* untuk fungsi tersebut. *Specifications* yang merupakan ketentuan yang harus dipenuhi mengenai *input* yang akan diterima oleh fungsi tersebut dan *output* yang akan dihasilkan oleh fungsi tersebut sehingga fungsi akan dijalankan jika dan hanya jika memenuhi dari *specifications* yang telah didefinisikan ?. Seperti yang sudah didefinisikan bahwa pemanggilan *adaptor* melalui *template engine* dengan cara `m.abs.namaFungsi[{query param=value}]`, maka ketika dijalankan `m_abs` akan menjalankan `m_find_value({query, Query}, #m{value=Q})` dimana `namaFungsi` yang didapatkan dari hasil pemanggilan pada *template engine* akan menjalankan fungsi `m_find_value(Type, #m{value=undefined})` untuk yang akan mengembalikan sebuah *maps* yang akan diterima kembali oleh fungsi `m_find_value({query, Query}, #m{value=Q})` yang menyimpan *maps* hasil kembalian tersebut ke dalam variabel `Q`. Lalu

parameter yang terdapat pada *template engine* akan disimpan oleh fungsi `m_find_value({query, Query}, #m{value=Q})` ke dalam variabel `Query`.

Setelah mendapatkan informasi tentang `Query` dan `Q`, fungsi `m_find_value` selanjutnya akan mengambil *key* yang akan dijalankan pada tabel *rules*. Langkah pertama, akan diekstrak *key* yang ingin dijalankan dari `Q` dengan cara *pattern matching*. Setelah mendapatkan *key* yang diinginkan, *key* tersebut akan dijadikan sebagai input untuk pemanggilan fungsi `lookup_rules` yang akan mengembalikan *endpoint* yang akan dipanggil oleh `m_abs` serta jumlah parameter dari fungsi yang akan dijalankan. Setelah mendapatkan *endpoint* serta jumlah parameter dari hasil pemanggilan fungsi `lookup_rules` yang berbentuk *list*, *list* tersebut akan diekstrak menjadi variabel `Url` yang menyimpan informasi *endpoint* dan variabel `Param` yang menyimpan informasi jumlah parameter dari fungsi tersebut. Selanjutnya fungsi `m_abs` akan memanggil fungsi `validate_params` dengan parameter `Param` yang menyimpan informasi jumlah parameter dan variabel `Query` yang menyimpan informasi mengenai *list* dari parameter yang diberikan pada *template engine*.

Setelah mendapatkan hasil dari pemanggilan fungsi `validate_params`, pada fungsi `m_find_value` akan mengembalikan *error* jika hasil dari pemanggilan fungsi `validate_params` bernilai `false`. Sebaliknya, jika hasil dari pemanggilan fungsi `validate_params` bernilai `true` maka fungsi `m_find_value` akan memanggil fungsi `fetch_data` dengan parameter berupa variabel `Url` dan juga Parameter `Query` yang sudah dijadikan dalam bentuk *json* menggunakan `jiffy:encode/1`. Hasil dari pemanggilan fungsi `fetch_data` akan mengembalikan sebuah *json* yang berisikan data yang ingin ditampilkan. Sesuai dengan rancangan *web service* yang sudah dijelaskan pada bab sebelumnya, data yang ingin ditampilkan terdapat pada parameter "data" pada *json* sehingga kita perlu mengambil nilai dari parameter "data" tersebut dengan cara `proplists:get_value(«"data"», DecodeJson)` dimana `DecodeJson` merupakan *json* hasil dari `fetch_data`.

4.1.2 Pemanggilan *Adaptor* Melalui Model

Selain dipanggil melalui *template engine*, ada kebutuhan untuk memanggil *adaptor* dari model lain seperti pemanggilannya pada model `m_rsc` agar dapat mengirim data dari *zotonic* kepada *web service* melalui *adaptor*. Untuk itu, perlu diimplementasi sebuah fungsi baru dengan nama `call_api_controller` pada *adaptor* yang dapat diakses secara langsung oleh model lain seperti berikut

Kode 4.6: Implementasi fungsi untuk pemanggilan *adaptor* dari model

```

call_api_controller(Key, Data) ->
  [Url, Param] = lookup_rules(Key),
  case validate_params(Param, Data) of
    false ->
      [{error, "Num of Params not same"}];
    true ->
      lager:info("key ~p", [Data]),
      lager:info("key ~s", [jiffy:encode({Data})]),
      {DecodeJson} = fetch_data(binary_to_list(Url), jiffy:encode
        ({Data})),
      lager:info("[ABS] result ~p", [DecodeJson]),
      case proplists:get_value(<<"status">>, DecodeJson) of
        200 ->
          DataResult = proplists:get_value(<<"data">>, DecodeJson
            ),
          lager:info("[ABS] status 200 ~p", [DataResult]);
        201 ->
          Message = proplists:get_value(<<"message">>, DecodeJson
            ),
          lager:info("[ABS] status 201 ~p", [binary_to_list(
            Message)]);
        400 ->
          Message = proplists:get_value(<<"message">>, DecodeJson
            ),
          lager:error("[ABS] status 400 ~p", [Message]);
        _Other ->
          lager:error("[ABS] status undefined ~p", [_Other])
      end
    end.

```

Pada kode 4.6, fungsi `call_api_controller` menerima dua parameter yaitu *Key* dan *Data*. *Key* merupakan nama fungsi yang ingin dipanggil oleh *web service*, dan *Data* merupakan data yang akan dikirim ke *web service*.

4.1.3 Implementasi Fungsi `lookup_rules`

Fungsi `lookup_rules` merupakan fungsi yang membaca berkas `rules.txt` dan mengembalikan *list* yang berisikan *endpoint* dan jumlah parameter dari *key* yang diberikan sebagai *input*. Adapun implementasi dari fungsi `lookup_rules` sebagai berikut

Kode 4.7: Implementasi fungsi lookup_rules

```

lookup_rules(Key) ->
  File = ?RULES,
  case read_file(File) of
    {error, Error} ->
      [{error, Error}];
    [] ->
      [{error, "File empty"}];
    Json ->
      {DecodeJson} = jiffy:decode(Json),
      proplists:get_value(atom_to_binary(Key, latin1), DecodeJson
      )
  end.

read_file(File) ->
  case file:read_file(File) of
    {ok, Data} ->
      Data;
    eof ->
      [];
    Error ->
      {error, Error}
  end.

```

Pada kode 4.7, dapat dilihat bahwa fungsi lookup_rules akan menjalankan fungsi read_file yang membaca seluruh isi dari file yang telah didefinisikan sebagai tabel *rules*. Perlu didefinisikan juga dimana lokasi dari tabel *rules* yang akan dibaca sebagai variabel final dengan nama variabel *RULES* pada m_abs seperti kode 4.8

Kode 4.8: lokasi dari file rules.txt

```
-define(RULES, "../rules.txt").
```

4.1.4 Implementasi Fungsi `validate_params`

Implementasi dari fungsi `validate_params` dapat dilihat sebagai berikut

Kode 4.9: Implementasi fungsi `validate_params`

```
validate_params(Param, Query) ->
  case length(Query) == Param of
    false ->
      false;
    true ->
      true
  end.
```

Pada kode 4.9, fungsi `validate_params` akan mengembalikan boolean hasil pengecekan jumlah parameter yang didapatkan dari tabel *rules* dan jumlah parameter yang terdapat pada *list* dari *Query*. Kembalikan *true* jika jumlah keduanya sama, dan kembalikan *false* jika jumlahnya tidak sama.

4.1.5 Implementasi Fungsi `fetch_data`

Adapun implementasi dari fungsi `fetch_data` sebagai berikut

Kode 4.10: Implementasi fungsi `fetch_data`

```
-spec fetch_data(Url, Query) -> list() when
  Url:: list(),
  Query:: list().
fetch_data(_, []) ->
  [{error, "Params missing"}];
fetch_data("", _) ->
  [{error, "Url missing"}];
fetch_data(Url, Query) ->
  case post_page_body(Url, Query) of
    {error, Error} ->
      [{error, Error}];
    Json ->
      jiffy:decode(Json)
  end.
```

Seperti pada gambar, tabel juga dapat diberi label dan caption. Caption pada tabel terletak pada bagian atas tabel. Contoh tabel sederhana dapat dilihat pada Tabel 4.1.

Tabel 4.1: Contoh Tabel

	kol 1	kol 2
baris 1	1	2
baris 2	3	4
baris 3	5	6
jumlah	9	12

Ada jenis tabel lain yang dapat dibuat dengan \LaTeX berikut beberapa diantaranya. Contoh-contoh ini bersumber dari <http://en.wikibooks.org/wiki/LaTeX/Tables>

Tabel 4.2: An Example of Rows Spanning Multiple Columns

No	Name	Week 1			Week 2		
		A	B	C	A	B	C
1	Lala	1	2	3	4	5	6
2	Lili	1	2	3	4	5	6
3	Lulu	1	2	3	4	5	6

Tabel 4.3: An Example of Columns Spanning Multiple Rows

Percobaan	Iterasi	Waktu
Pertama	1	0.1 sec
Kedua	1	0.1 sec
	3	0.15 sec
Ketiga	1	0.09 sec
	2	0.16 sec
	3	0.21 sec

Tabel 4.4: An Example of Spanning in Both Directions Simultaneously

		Title			
		A	B	C	D
Type	X	1	2	3	4
	Y	0.5	1.0	1.5	2.0
Resource	I	10	20	30	40
	J	5	10	15	20

4.2 Implementasi *Refactoring Business Logic*

Berkas ini berisi seluruh berkas Latex yang dibaca, jadi bisa dikatakan sebagai berkas utama. Dari berkas ini kita dapat mengatur bab apa saja yang ingin kita tampilkan dalam dokumen.

BAB 5

HASIL

5.1 Implementasi *Cluster*

5.1.1 Instalasi *Frontend*

Tabel model lain, ditunjukkan pada tabel 5.1.

Tabel 5.1: Informasi *cluster X*

Host Name	X
Cluster Name	X
Certificate Organization	UI
Certificate Locality	Depok
Certificate State	West Java
Certificate Country	ID
Contact	X
URL	http://grid.ui.ac.id

Ada pagebreak disini.

Another type of table

Tabel 5.2: Perbandingan Partisi *default* dan manual

	Partisi default	Partisi manual yang dilakukan
/	16 GB	30 GB
/var	4 GB	18 GB
swap	1 GB	2 GB
/export	55 GB	26 GB

Program menghasilkan keluaran seperti pada kode 5.1.

Kode 5.1: Keluaran output

```
[root@nas-0-0 ~]# cat /proc/mdstat
Personalities : [raid1]
md0 : active raid1 sda4[0] sdb2[1]
      1917672312 blocks super 1.2 [2/2] [UU]

unused devices: <none>
[root@nas-0-0 ~]# mdadm --detail /dev/md0
/dev/md0:
    Version : 1.2
  Creation Time : Fri May  3 15:38:52 2013
    Raid Level : raid1
    Array Size : 1917672312 (1828.83 GiB 1963.70 GB)
  Used Dev Size : 1917672312 (1828.83 GiB 1963.70 GB)
    Raid Devices : 2
  Total Devices : 2
 Persistence : Superblock is persistent

Update Time : Tue May 28 11:27:49 2013
    State : clean
 Active Devices : 2
Working Devices : 2
 Failed Devices : 0
  Spare Devices : 0

    Name : nas-0-0.local:0 (local to host nas-0-0.local)
   UUID : 0754726d:3dfbd4b9:42b0f587:68631556
 Events : 28

   Number   Major   Minor   RaidDevice State
    0         8         4         0     active sync   /dev/sda4
    1         8        18         1     active sync   /dev/sdb2
```

5.1.2 Konfigurasi

Contoh verbatim dalam itemize :

- **Bold ini**

dijalankan perintah berikut :

```
# javac Ganteng.java
# java Ganteng
```

Perilaku sistem

```
# hai
# enable
# cd /export/rocks/install/
# create distro
# sh sesuatu.sh
# reboot
```

- **Menambahkan *package* pada *compute node***

Langkah yang dilakukan adalah sebagai berikut :

1. Masuk ke dalam direktori `/procfs/`
2. Membuat/Mengubah berkas `xx.xml`. Jika tidak terdapat berkas tersebut, dapat disalin dari `skeleton.xml`.
3. Menambahkan *package* yang ingin dipasang pada *compute node* diantara *tag* `<package>` seperti berikut : `<package>[package yang akan dipasang]</package>`.
4. Menjalankan perintah berikut termasuk perintah untuk melakukan instalasi ulang seluruh *compute node*:

```
# cd /export/somedir
# create
# run host
```

5.1.2.1 semakin ke dalam

Kode 5.2: Keluaran mentah untuk detail *job*

```
[ardhi@xx ~]$ qstat -f 138
Job Id: 138.xx
  Job_Name = cur-1000-1np
  Job_Owner = ardhi@xx
  resources_used.cput = 27:21:35
  resources_used.mem = 86060kb
  resources_used.vmem = 170440kb
  resources_used.walltime = 27:24:50
  job_state = R
  queue = default
  server = hastinapura.grid.ui.ac.id
  Checkpoint = u
  ctime = Fri May 31 10:27:37 2013
  Error_Path = xx:/home/ardhi/xx/curcumin-1000/cur-1000-1np.e138
  exec_host = compute-0-5/0
  exec_port = 15003
  Hold_Types = n
  Join_Path = n
  Keep_Files = n
  Mail_Points = e
  Mail_Users = ardhi.putra@ui.ac.id
  mtime = Fri May 31 10:27:47 2013
  Output_Path = xx:/home/ardhi/xx/curcumin-1000/cur-1000-1np.o138
  Priority = 0
  qtime = Fri May 31 10:27:37 2013
  Rerunnable = True
  Resource_List.nodes = 1:ppn=1
  session_id = 5768
  etime = Fri May 31 10:27:37 2013
  submit_args = cur-1000-1np.pbs
  start_time = Fri May 31 10:27:47 2013
  submit_host = xx
  init_work_dir = /home/ardhi/xx/curcumin-1000
```

5.2 Pengujian

5.2.1 Kasus Uji

Berwarna!

Kode 5.3: Potongan skrip submisi *job* melalui torque

```
# Go To working directory
cd $PBS_O_WORKDIR

#openMPI prerequisite
. /opt/torque/etc/openmpi-setup.sh
```

```
mpirun -np 5 -machinefile $PBS_NODEFILE mdrun -v -s \
  curcum400ps.tpr -o md_prod_curcum400_5np.trr -c lox_pr.gro
...
```

5.2.2 Kasus Uji

Contoh skrip yang dimasukkan pada *form* yang disediakan dapat dilihat pada kode 5.4.

Kode 5.4: Potongan Makefile *project*

```
# Make file for MPI
SHELL=/bin/sh

# Compiler to use
# You may need to change CC to something like CC=mpiCC
# openmpi : mpiCC
# mpich2   : /opt/mpich2/gnu/bin/mpicxx
CC=mpiCC
...
...
```

BAB 6

PENUTUP

Pada bab terakhir ini,

6.1 Kesimpulan

6.2 Saran

LAMPIRAN

LAMPIRAN 1 : KODE SUMBER MODEL ABS

m_abs.erl

Skrip ini diletakkan pada direktori `/usr/sesuatu` dan hanya dapat dieksekusi oleh *root*. Skrip ini berguna untuk menambahkan pengguna baru sesuai dengan konfigurasi baru yang telah ditetapkan.

Kode 1: Skrip adaptor m_abs.erl

```
%% @author Andri Kurniawan <andrikurniawan.id@gmail.com>
%% @copyright 2017 Andri Kurniawan
%% Date: 2017-05-11
%%
%% @doc Template access for abs model

%% Copyright 2017 Andri Kurniawan
%%
%% Licensed under the Apache License, Version 2.0 (the "License");
%% you may not use this file except in compliance with the License.
%% You may obtain a copy of the License at
%%
%%     http://www.apache.org/licenses/LICENSE-2.0
%%
%% Unless required by applicable law or agreed to in writing, software
%% distributed under the License is distributed on an "AS IS" BASIS,
%% WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
%% See the License for the specific language governing permissions and
%% limitations under the License.
-module(m_abs).
-behaviour(gen_model).

-export([
    m_find_value/3,
    m_to_list/2,
    m_value/2,
    call_api_controller/2
]).

-include_lib("zotonic.hrl").

-define(RULES, "/home/andri/skripsi/zotonic/rules.txt").

% this method to handle call api from template
-spec m_find_value(Key, Source, Context) -> #m{} | undefined | any() when
    Key:: integer() | atom() | string(),
    Source:: #m{},
    Context:: #context{}.

m_find_value(Type, #m{value=undefined} = M, _Context) ->
    M#m{value=[Type]};
```

```

m_find_value({query, Query}, #m{value=Q} = _, _Context) when is_list(Q) ->
    [Key] = Q,
    [Url, Param] = lookup_rules(Key),
    case validate_params(Param, Query) of
        false ->
            [{error, "Num of Params not same"}];
        true ->
            {DecodeJson} = fetch_data(binary_to_list(Url), jiffy:encode({Query})),
            lager:info("ABS result : ~p", [DecodeJson]),
            proplists:get_value(<<"data">>, DecodeJson)
    end;

% Other values won't be processed
m_find_value(_, _, _Context) ->
    undefined.

m_to_list(_, _Context) ->
    [].

m_value(_, _Context) ->
    undefined.

% this method to handle call api from another module
call_api_controller(Key, Data) ->
    [Url, Param] = lookup_rules(Key),
    case validate_params(Param, Data) of
        false ->
            [{error, "Num of Params not same"}];
        true ->
            lager:info("key ~p", [Data]),
            lager:info("key ~s", [jiffy:encode({Data})]),
            {DecodeJson} = fetch_data(binary_to_list(Url), jiffy:encode({Data})),
            lager:info("[ABS] result ~p", [DecodeJson]),
            case proplists:get_value(<<"status">>, DecodeJson) of
                200 ->
                    DataResult = proplists:get_value(<<"data">>, DecodeJson),
                    lager:info("[ABS] status 200 ~p", [DataResult]);
                201 ->
                    Message = proplists:get_value(<<"message">>, DecodeJson),
                    lager:info("[ABS] status 201 ~p", [binary_to_list(Message)]);
                400 ->
                    Message = proplists:get_value(<<"message">>, DecodeJson),
                    lager:error("[ABS] status 400 ~p", [Message]);
                _Other ->
                    lager:error("[ABS] status undefined ~p", [_Other])
            end
        end
    end.

-spec fetch_data(Url, Query) -> list() when
    Url::list(),
    Query::list().
fetch_data(_, []) ->
    [{error, "Params missing"}];
fetch_data("", _) ->
    [{error, "Url missing"}];
fetch_data(Url, Query) ->

```

```

    case post_page_body(Url, Query) of
      {error, Error} ->
        [{error, Error}];
      Json ->
        jiffy:decode(Json)
    end.

post_page_body(Url, Body) ->
  case http:request(post, {Url, [], "application/json", Body}, [], []) of
    {ok, {_, _, Response}} ->
      Response;
    Error ->
      {error, Error}
  end.

lookup_rules(Key) ->
  File = ?RULES,
  case read_file(File) of
    {error, Error} ->
      [{error, Error}];
    [] ->
      [{error, "File empty"}];
    Json ->
      {DecodeJson} = jiffy:decode(Json),
      proplists:get_value(atom_to_binary(Key, latin1), DecodeJson)
  end.

read_file(File) ->
  case file:read_file(File) of
    {ok, Data} ->
      Data;
    eof ->
      [];
    Error ->
      {error, Error}
  end.

validate_params(Param, Query) ->
  case length(Query) == Param of
    false ->
      false;
    true ->
      true
  end.

```

LAMPIRAN 2 : KODE SUMBER RULES

rules.txt

Kode 2: Berkas compute.xml

```
{
  "createProgram": ["http://54.169.128.6:8080/abs/program/create"
    , 2],
  "createDonation": ["http://54.169.128.6:8080/abs/donation/
    create", 4],
  "updateProgram": ["http://54.169.128.6:8080/abs/program/update"
    , 2],
  "updateDonation": ["http://54.169.128.6:8080/abs/donation/
    update", 4],
  "deleteProgram": ["http://54.169.128.6:8080/abs/program/delete"
    , 1],
  "deleteDonation": ["http://54.169.128.6:8080/abs/donation/
    delete", 1],
  "totalDonation" : ["http://54.169.128.6:8080/abs/program/total-
    donation", 1]
}
```

LAMPIRAN 8 : UAT DAN KUESIONER

Tabel 1: Tabel UAT dan Kuesioner

No.	Langkah Penggunaan	Fitur Berjalan	Tingkat Kemudahan (1-5)	Tingkat Kepuasan (1-5)	Saran / Komentar
		Berhasil /Tidak	1:Sangat sulit ; 5:sangat mudah	1 : Sangat kecewa ; 5 : sangat puas	
Use Case : Login					
1.1	Pengguna berada pada halaman depan torqace				
1.2	Pengguna memasukkan username dan password pada field yang telah disediakan.Kemudian menekan tombol 'login'				
1.3	Apabila Sukses, maka pengguna masuk ke dalam sistem dan dihadapkan pada menu utama				
Use Case : Register					
2.1	Pengguna berada pada halaman registrasi pengguna torqace				

2.2	Pengguna memasukkan user-name,password, dan email pada field yang telah disediakan. Kemudian menekan tombol 'submit'				
2.3	Sistem akan mengonfirmasi masukan, dan akan mengirimkan email untuk memberitahu pengguna apabila proses pendaftaran telah selesai				
Use Case : Logout					
3.1	Pengguna memilih menu untuk melakukan logout				
3.2	Sistem akan mengeluarkan pengguna, dan pengguna tidak dapat menggunakan fitur-fitur utama aplikasi				
Use Case : Upload Job Sederhana					
4.1	Pengguna memilih menu upload file/project pada menu utama				
4.2	Pengguna memilih pilihan 'single file' pada tipe project				

4.3	Pengguna memilih berkas yang akan diunggah, mengisi label, dan menentukan apakah akan menimpa project sebelumnya dengan nama yang sama atau tidak				
4.4	Pengguna menekan tombol 'submit' dan mengonfirmasi				
4.5	Sistem akan menampilkan informasi terkait berkas yang diupload				
Use Case : Upload Job Compressed					
5.1	Pengguna memilih menu upload file/project pada menu utama				
5.2	Pengguna memilih pilihan 'compressed files' pada tipe project				
5.3	Pengguna memilih arsip yang akan diunggah, mengisi label, menentukan akan melakukan make atau tidak dan menentukan apakah akan menimpa project sebelumnya dengan nama yang sama atau tidak				
5.4	Pengguna menekan tombol 'submit' dan mengonfirmasi				

5.5	Sistem akan menampilkan informasi terkait berkas yang diupload dan diekstrak. Keluaran make juga akan ditampilkan bila dipilih				
Use Case : Upload Array Job					
6.1	Pengguna memilih menu upload file/project pada menu utama				
6.2	Pengguna memilih pilihan 'array' pada tipe project				
6.3	Pengguna memilih arsip-arsip yang akan diunggah, mengisi label, menentukan akan melakukan make atau tidak dan menentukan apakah akan menimpa project sebelumnya dengan nama yang sama atau tidak				
6.4	Pengguna menekan tombol 'submit' dan mengonfirmasi				
6.5	Sistem akan menampilkan informasi terkait berkas yang diupload dan diekstrak. Keluaran make juga akan ditampilkan bila dipilih				

Use Case : Melihat antrian pada queue					
7.1	Pengguna memilih menu queue status pada menu utama				
7.2	Pengguna berada pada halaman yang berisi informasi queue				
Use Case : Melihat detil antrian					
8.1	Dari halaman status queue, pengguna memilih job tertentu				
8.2	Informasi mengenai detil job tersebut ditampilkan dalam bentuk tabel				
8.2.1	Apabila job tersebut bukan milik pengguna, maka sistem akan melarang pengguna melihat informasi detil suatu job				
Use Case : Membuat script job					
9.1	Pengguna memilih untuk melakukan 'generate script' baik dari laporan upload berkas, atau dari penjelajahan direktori				
9.2	Pengguna mengisi nama job, parameter job, dan script yang akan dijalankan.				
9.3	Pengguna mengonfirmasi konfirmasi submit job				

9.4	Pengguna dapat melihat informasi script secara keseluruhan dan pesan apakah terjadi kegagalan atau tidak, serta id job yang diberikan				
Use Case : Load spesifikasi job lain					
10.1	Pengguna berada pada halaman untuk membuat script				
10.2	Pengguna memilih 'Load a Previous Job'				
10.3	Pengguna memilih job mana yang akan dimuat dan menekan tombol 'Load'				
10.4	Pengguna kembali ke halaman pembuatan script dengan spesifikasi job sebelumnya				
Use Case : Menjelajah Direktori					
11.1	Pengguna memilih menu 'View File/Project' pada menu utama				
11.2	Pengguna dapat melakukan navigasi untuk masuk ke dalam direktori tertentu, atau kembali ke direktori di atasnya, dan dapat melihat terdapat berkas apa saja dalam direktori				

Use Case : Menghapus Berkas/Direktori					
12.1	Pengguna berada pada halaman penjelajahan direktori				
12.2	Pengguna memilih pilihan untuk menghapus berkas/direktori di samping item yang akan dihapus				
12.3	Pengguna mengonfirmasi konfirmasi penghapusan				
Use Case : Mengunduh Berkas/Direktori					
13.1	Pengguna berada pada halaman penjelajahan direktori				
13.2	Pengguna memilih pilihan untuk mengunduh berkas/direktori di samping item yang akan dihapus				
Use Case : Melihat Berkas					
14.1	Pengguna berada pada halaman penjelajahan direktori				
14.2	Pengguna memilih berkas yang berupa berkas teks				
14.3	Sistem akan menampilkan konten dari berkas tersebut				