BAB 2 LANDASAN TEORI

Pada bab ini akan dijelaskan landasan teori dan teknologi yang digunakan dalam penelitian ini. Pembahasan yang dipaparkan akan dimulai dari teknologi *web service* beserta dengan beberapa standar teknologi yang digunakan untuk *web service*, yaitu XML, WSDL, dan SOAP. Kemudian akan dibahas mengenai proyek Language Grid yang menggunakan teknologi *web service*. Di dalam pengembangan proyek ini, NICT sebagai pemilik proyek ini bekerja sama dengan laboratorium perolehan informasi Fakultas Ilmu Komputer Universitas Indonesia. Terakhir, akan dijelaskan mengenai program *Morphological Analyzer*, yaitu program yang akan dibungkus ke dalam *web service*, dan dimasukkan menjadi salah satu bagian proyek Language Grid.

1.1 *Web Service*

Web service adalah sebuah teknologi yang meliputi sekumpulan standar yang memungkinkan dua aplikasi komputer dapat saling berkomunikasi dan bertukar data di Internet (Deitel, Deitel, & Goldberg, 2004). Standar yang digunakan oleh *web service* meliputi *Extensible Markup Language* (XML), *Simple Object Access Protocol* (SOAP), *Web Service Description Language* (WSDL), dan *Universal Description, Discovery and Integration* (UDDI).

XML digunakan untuk mendeskripsikan data. SOAP digunakan untuk memindahkan data. WSDL digunakan untuk mendeskripsikan layanan yang disediakan oleh sebuah *web service*. UDDI digunakan untuk mendaftarkan *web service* yang tersedia. Penjelasan lebih lanjut mengenai XML, SOAP, dan WSDL dapat dilihat pada Subbab 2.1.1 hingga Subbab 2.1.3.

Dengan adanya *web service*, dua aplikasi yang dikembangkan dengan bahasa pemrograman yang berbeda dan dijalankan pada sistem operasi yang berbeda dapat saling berkomunikasi dan bertukar data melalui Internet dengan menggunakan protokol standar seperti HTTP. Selain itu, teknologi *web service* ini memungkinkan

kolaborasi yang lebih baik dalam pengembangan perangkat lunak, karena para pengembang dapat menggabungkan beberapa perangkat lunak yang dikembangkan dengan bahasa pemrograman yang berbeda dan pada sistem operasi yang berbeda.

1.1.1 *Extensible Markup Language* (XML)

Extensible Markup Language (XML) adalah sebuah *markup language* yang digunakan untuk mendeskripsikan data dalam sebuah struktur yang teratur (Deitel, Deitel, & Goldberg, 2004). *Markup language* adalah sebuah mekanisme untuk mengidentifikasikan struktur dari sebuah dokumen (Walsh, 2008). *Markup* adalah sesuatu yang ditambahkan pada sebuah dokumen yang mendeskripsikan isi dari teks yang ada di dalam dokumen tersebut.

Data yang memiliki struktur yang teratur memiliki isi (dapat berupa kata-kata, gambar, dan lain-lain) dan peran dari setiap bagian isi yang ada di dalam data tersebut. Sebagai contoh, bagian isi yang merupakan judul berbeda dengan bagian isi yang merupakan catatan kaki.

XML dirancang untuk pertukaran dan penyimpanan data. Sebuah dokumen XML hanya berisi data, dan bukan instruksi seperti halnya HTML yang berisi instruksi mengenai bagaimana suatu data ditampilkan pada halaman *web*. Oleh karena itu, aplikasi yang memproses dokumen XML harus menentukan bagaimana data yang ada di dalam dokumen XML tersebut dimanipulasi dan digunakan untuk keperluan aplikasi tersebut. Sebagai contoh, sebuah program komputer dapat menampilkan data yang ada di dalam sebuah dokumen XML dengan cara yang berbeda dengan program komputer lainnya. Untuk menspesifikasikan bagaimana sebuah dokumen XML ditampilkan dalam sebuah *platform* tertentu, digunakan *Extensible Stylesheet Language* (XSL).

Dokumen XML memiliki sifat *portable*. Artinya, untuk melihat atau memodifikasi isi dari dokumen XML (*file* dengan ekstensi `.xml`) tidak diperlukan perangkat lunak yang khusus. Semua *text editor* yang mendukung karakter ASCII/Unicode dapat membuka dokumen XML ini untuk dilihat dan dimodifikasi. Salah satu karakteristik yang penting dari XML adalah dapat dibaca oleh manusia dan

dapat “dibaca” oleh mesin. Mesin dapat memproses dokumen XML dengan menggunakan program yang disebut *XML parser*.

Sebuah dokumen XML dapat memberikan referensi ke sebuah *Document Type Definition* (DTD) atau ke sebuah *XML schema* yang mendefinisikan struktur dari sebuah dokumen XML secara khusus. Berikutnya, *XML parser* akan memeriksa isi dari DTD atau *XML schema* yang direferensikan oleh dokumen XML untuk memastikan apakah dokumen XML tersebut adalah sebuah dokumen XML yang *valid* atau tidak. Jika struktur dokumen XML sesuai dengan struktur yang didefinisikan pada DTD atau *XML schema* yang direferensikannya, maka dokumen XML tersebut dinyatakan *valid*.

Dokumen XML menyimpan data di dalam *tag-tag* yang dapat didefinisikan di dalam DTD atau *XML schema*. *Tag* yang telah didefinisikan dapat digunakan untuk memberikan makna pada bagian dari data yang ada di dalam sebuah dokumen XML. Cara penggunaannya mengikuti beberapa jenis *format* berikut.

```
<[TAG]>[CONTENT]</[TAG]>
<[TAG] />[CONTENT]
<[TAG] [[ATRIBUT]='[CONTENT_ATRIBUT]']>[CONTENT]</[TAG]>
```

Berbeda dengan **[CONTENT]**, **[TAG]** tidak boleh berupa *string* kosong. Selain itu, **[TAG]** juga bersifat *case sensitive*. **[TAG]** dapat didefinisikan sesuai dengan kebutuhan atau keinginan pembuat dokumen XML. Contoh dari sebuah dokumen XML yang sederhana dapat dilihat pada Gambar 0.1.

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

<!-- Article structured with XML -->

<article>

  <title>Simple XML</title>

  <date>July 15, 2003</date>

  <author>
    <firstName>Carpenter</firstName>
```

```
    <lastName>Cal</lastName>
  </author>

  <summary>XML is pretty easy</summary>

  <content>Once you have mastered XHTML, XML is easily
    Learned. You must remember that XML is not for
    Displaying information but for managing information.
  </content>

</article>
```

Gambar 0.1 Contoh dokumen XML (Deitel, Deitel, & Goldberg, 2004)

Pembuat dokumen XML dapat memberikan nama *tag* yang bermacam-macam. Sebagai contoh, pada dokumen XML yang ada pada Gambar 2.1 terdapat beberapa nama *tag* seperti **`article`**, **`title`**, **`date`**, **`author`**, **`firstName`**, dan lain-lain. Hal ini dapat menyebabkan terjadinya konflik penamaan jika terdapat dua *tag* dengan nama yang sama tetapi memiliki makna yang berbeda. Untuk mengatasi konflik penamaan ini, maka diciptakanlah XML *namespaces*. *Namespace* tersebut didefinisikan pada atribut **`xmlns`** pada bagian awal *tag* dari sebuah elemen XML. XML *parser* akan membandingkan isi dari atribut **`xmlns`** tersebut untuk menentukan apakah dua *tag* yang bernama sama terasosiasi pada satu buah *namespace* atau tidak. Isi dari atribut tersebut biasanya berupa sebuah *Uniform Resource Identifier* (URI) yang di dalamnya berisi informasi mengenai *tag* XML tersebut.

Ketika sebuah *namespace* didefinisikan pada sebuah elemen XML, maka anak dari elemen XML tersebut (elemen XML lain yang terdapat di dalam elemen XML tersebut) akan diasosiasikan dengan *namespace* yang sama secara otomatis. Gambar 0.2 menunjukkan contoh penggunaan XML *namespace* dari sebuah dokumen XML untuk membedakan dua buah elemen XML yang memiliki nama *tag* yang sama, yaitu **`<table>`**. Pada contoh tersebut, informasi mengenai *tag* <table> yang pertama bisa didapatkan pada http://www.w3.org/TR/html4/, sedangkan informasi mengenai *tag* **`<table>`** yang kedua bisa didapatkan pada http://www.w3schools.com/furniture. Oleh karena kedua elemen tersebut memiliki *namespace* yang berbeda, maka meskipun nama *tag* dari kedua elemen tersebut sama, kedua *tag* tersebut dianggap sebagai *tag* yang berbeda oleh XML *parser*.

```
<root>
<h:table xmlns:h="http://www.w3.org/TR/html4/">
   <h:tr>
   <h:td>Apples</h:td>
   <h:td>Bananas</h:td>
   </h:tr>
</h:table>
<f:table xmlns:f="http://www.w3schools.com/furniture">
   <f:name>African Coffee Table</f:name>
   <f:width>80</f:width>
   <f:length>120</f:length>
</f:table>
</root>
```

Gambar 0.2 Contoh penggunaan XML *namespace* (Refsnes Data, 2007)

1.1.2 *Web Service Description Language* (WSDL)

Web Service Description Language (WSDL) adalah sebuah bahasa berbasis XML yang digunakan untuk mendeskripsikan sebuah *web service*, termasuk di dalamnya adalah informasi mengenai operasi yang disediakan oleh *web service* tersebut beserta dengan tipe parameter yang diperlukan, dan informasi mengenai lokasi akses dari *web service* tersebut.

Sebuah dokumen WSDL mendeskripsikan sebuah *web service* dengan menggunakan elemen-elemen utama seperti yang dijabarkan pada Tabel 0.1.

Tabel 0.1 Elemen utama dari WSDL

Elemen	Mendefinisikan
<portType>	Operasi yang disediakan oleh *web service*.
<message>	*Message* yang digunakan oleh *web service*.
<types>	Tipe data yang digunakan oleh *web service*.
<binding>	Protokol komunikasi yang digunakan oleh *web service*.

Struktur dari sebuah dokumen WSDL dapat dilihat pada Gambar 0.3.

```
<definitions>
<types>
   definition of types........
</types>

<message>
```

```
   definition of a message....
</message>

<portType>
   definition of a port.......
</portType>

<binding>
   definition of a binding....
</binding>

</definitions>
```

Gambar 0.3 Struktur dokumen WSDL

1.1.3 *Simple Object Access Protocol* (SOAP)

Banyak aplikasi yang menggunakan Internet untuk melakukan pertukaran data. Beberapa aplikasi tersebut berjalan pada sisi *client* dan butuh memanggil *method* yang ada pada mesin lainnya untuk melakukan proses pada data. Banyak dari aplikasi ini yang menggunakan protokol dan spesifikasi data secara khusus untuk aplikasi tersebut, yang membuat komunikasi dengan aplikasi lain menjadi sulit. Untuk mengatasi kesulitan tersebut, IBM, Lotus Development Corporation, Microsoft, DevelopMentor, dan Userland Software mengembangkan *Simple Object Access Protocol* (SOAP). SOAP adalah protokol berbasis XML yang memungkinkan aplikasi-aplikasi dapat berkomunikasi secara mudah di Internet dengan menggunakan dokumen XML yang disebut *SOAP messages* (Deitel, Deitel, & Goldberg, 2004).

Setidaknya terdapat dua buah keunggulan utama dari penggunaan SOAP, yaitu standarisasi dan kemampuan enkapsulasi obyek. SOAP telah menjadi protokol yang standar digunakan oleh program yang ingin berkomunikasi melalui Internet. Selain itu, SOAP juga memungkinkan dua program saling bertukar obyek di dalam sebuah komunikasi melalui Internet, bukan hanya saling bertukar informasi berupa teks seperti yang dimungkinkan oleh pendahulu dari SOAP. Dalam kedua hal inilah, SOAP memiliki keunggulan.

Sebuah *SOAP message* berisi sebuah *envelope*, yaitu sebuah struktur yang mendeskripsikan pemanggilan suatu *method*. Bagian badan dari *SOAP message* berisi salah satu antara sebuah *request* atau sebuah *response*. Badan dari sebuah *request*

message berisi *Remote Procedure Call* (RPC), yaitu sebuah *request* untuk mesin lain agar melakukan suatu tugas tertentu. RPC menspesifikasikan *method* yang harus dipanggil beserta setiap isi parameter yang dibutuhkan oleh *method* tersebut. *SOAP message* dikirim melalui sebuah HTTP POST. *SOAP response message* adalah sebuah dokumen respon HTTP yang mengandung hasil dari pemanggilan suatu *method* (contoh: nilai yang dikembalikan, pesan kesalahan, dan lain-lain). Contoh dari *SOAP request message* dan *SOAP response message* dapat dilihat pada Gambar 0.4 dan Gambar 0.5.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Gambar 0.4 Contoh *SOAP request message* (Refsnes Data, 2008)

Gambar 0.4 menampilkan contoh dari *SOAP request message*. Melalui *message* tersebut, *client* meminta untuk menjalankan operasi **`GetStockPrice`** dengan "IBM" sebagai isi dari parameter **`StockName`** pada operasi tersebut. Selanjutnya, *client* akan mendapatkan *response* berupa nilai dari atribut **`Price`**, yaitu "34.5", yang merupakan keluaran dari operasi **`GetStockPrice`**. Contoh dari *SOAP response message* tersebut dapat dilihat pada Gambar 0.5.

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```

```
  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

Gambar 0.5 Contoh *SOAP response message* (Refsnes Data, 2008)

1.2 Language Grid

Language Grid adalah sebuah infrastuktur bahasa yang dibangun di Internet. Language Grid memungkinkan penggunanya mendapatkan pemahaman yang lebih baik dari informasi yang ada di Internet yang ditulis dalam berbagai bahasa dan oleh orang dari negara yang berbeda-beda dengan memanfaatkan *language service* yang ada di Language Grid. Selain itu, Language Grid juga memungkinkan penggunanya untuk membangun *language service* baru dengan mengkombinasikan *language service* yang telah ada di Language Grid sesuai dengan kebutuhan mereka (NICT, 2008).

Proyek Language Grid telah dimulai pada tahun 2005, sebagian besar oleh peneliti dari universitas dan institut penelitian sekitar Kyoto. Pada bulan April 2006, proyek ini diadopsi oleh NICT Knowledge Creating Communication Research Center. Saat ini, pengembangan proyek Language Grid dikerjakan secara kolaborasi dengan industri, pemerintah, universitas, dan penduduk, termasuk di antaranya dengan Universitas Indonesia.

1.2.1 Latar Belakang Language Grid

Banyak informasi yang tersedia di Internet. Internet juga memungkinkan orang-orang untuk dapat saling berhubungan satu sama lain. Namun, terdapat sebuah kendala yang besar yang ada di Internet, yaitu kendala bahasa. Tidak ada sebuah bahasa yang menjadi standar bahasa yang digunakan di Internet. Bahkan, penggunaan bahasa Inggris di Internet, yang merupakan bahasa internasional, pada tahun 2004 hanya mencapai sekitar 35% dari keseluruhan isi Internet (Murakami, Ishida, & Nakaguchi, 2006). Untuk dapat memahami keseluruhan isi Internet, diperlukan

penguasaan terhadap banyak bahasa di dunia, dan hal tersebut sangatlah sulit. Meskipun saat ini telah ada *language service* seperti mesin penerjemah, tetapi masih ada beberapa masalah seperti kendala di dalam penggunaannya. Kendala tersebut antara lain adalah tingginya biaya di dalam menggunakan dan mengembangkan *language service* tersebut, kontrak yang kompleks, hak kekayaan intelektual, antarmuka yang tidak standar yang menyulitkan penggunanya untuk mengkombinasikan *language service* tersebut. Untuk dapat membuat aktivitas kolaborasi interkultural yang ada di Internet menjadi lebih baik, kendala bahasa ini harus diatasi.

Dua tujuan utama dari proyek Language Grid adalah mengumpulkan *language service* standar yang telah ada saat ini yang dibangun oleh profesional di dalam bidang linguistik dan membantu penggunanya untuk membuat *language service* baru sesuai kebutuhan mereka dengan menggabungkan *language service* yang mereka miliki dengan *language service* yang telah disediakan oleh profesional. Jadi, melalui Language Grid, penggunanya bersama dengan profesional dapat saling membantu dalam mengatasi kendala bahasa yang ada di Internet.

1.2.2 Fungsi Language Grid

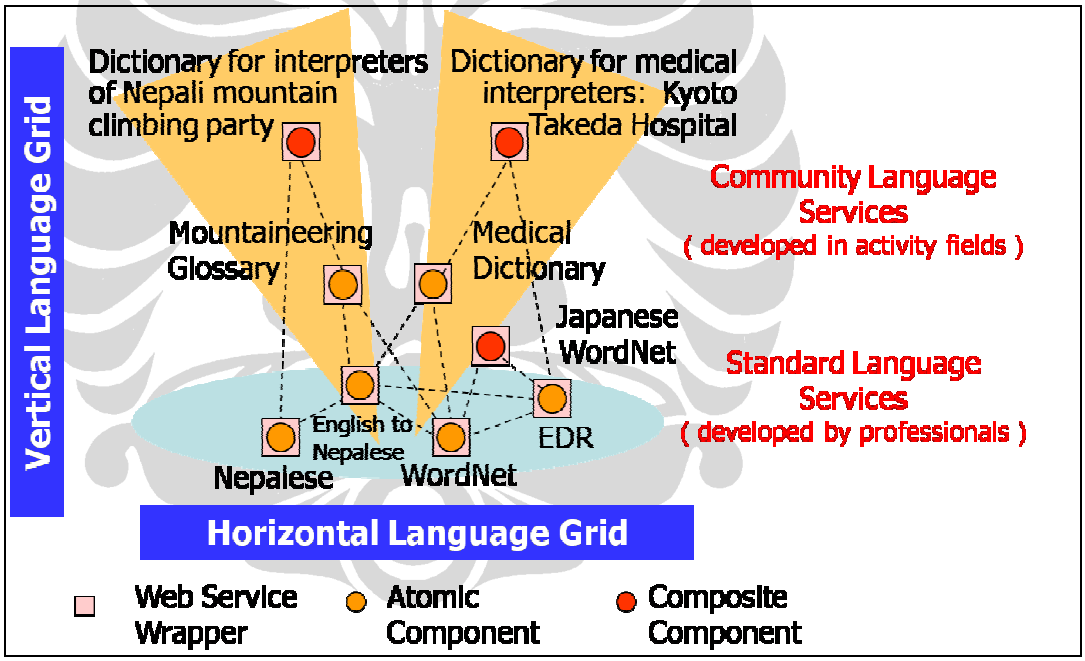
Pada Subbab 2.2.1 telah dijelaskan bahwa Language Grid memungkinkan penggunanya untuk membuat sebuah *language service* baru, yang sesuai dengan kebutuhan mereka, dengan mengkombinasikan *language resource* yang telah ada di Internet dan *language resource* yang mereka miliki. Untuk mencapai hal tersebut, Language Grid mempunyai dua buah struktur utama, yaitu *Horizontal Language Grid*, dan *Vertikal Language Grid* (Ishida, 2006).

Horizontal Language Grid berpusat pada pengembangan *language service* standar yang dikembangkan oleh profesional, seperti kamus dwibahasa, mesin penerjemah, dan lain-lain. Sedangkan, *Vertical Language Grid* berpusat pada pengembangan *language service* untuk memenuhi kebutuhan khusus dari komunitas tertentu, yang dikembangkan oleh komunitas, seperti kamus medis, kamus pendaki

gunung, dan lain-lain. Gambaran kedua struktur Language Grid ini dapat dilihat pada Gambar 0.6.

1.2.3 Teknologi yang Digunakan Language Grid

Untuk memungkinkan kolaborasi yang dibutuhkan di antara *language resources* dan *language processing functions*, teknologi *Semantic Web* akan terus dikembangkan. Termasuk di dalam *language resources* adalah kamus dwibahasa, *thesauruses*, dan lain-lain. Sedangkan, termasuk di dalam *language processing functions* adalah mesin penerjemah, penganalisis morfologi (*Morphological Analyzer*), dan-lain-lain.



Gambar 0.6 Fungsi Language Grid (Ishida, 2006)

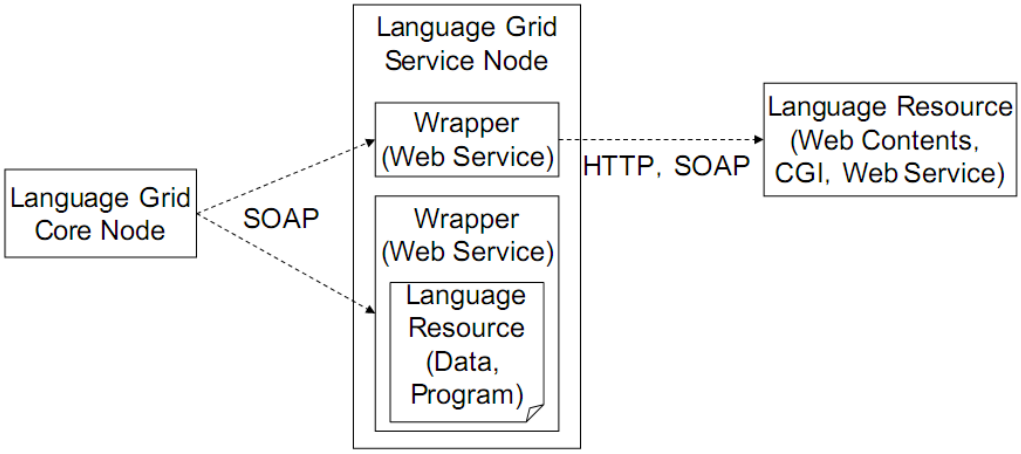
Teknologi yang digunakan Language Grid meliputi *language service ontology* dan *Semantic Web service*. *Language service ontology* adalah sebuah teknologi untuk mendefinisikan *language service* dengan suatu cara yang standar. *Semantic Web service* adalah sebuah teknologi yang memanfaatkan *web service*. Kedua teknologi ini memungkinkan kombinasi dari *language resources* dan *language processing*

functions yang ada. Sebagai contoh, dengan adanya mesin penerjemah bahasa Jepang – bahasa Inggris dan mesin penerjemah bahasa Jepang – bahasa Jerman, dapat dibangun mesin penerjemah bahasa Inggris – bahasa Jerman. Meskipun mutu dari *language service* yang dihasilkan mungkin tidak terlalu bagus bahkan jauh dari sempurna, *language service* tersebut diharapkan telah dapat dimanfaatkan untuk membantu orang yang mengalami kendala bahasa. Jadi, mutu bukanlah fokus utama pada proyek ini.

1.2.4 Definisi *Wrapper*

Berdasarkan (NICT Language Grid Project, 2008), *wrapper* adalah sebuah program yang membuat *language resources* dapat diakses melalui *web service*, dan yang menyesuaikan spesifikasi masukan dan spesifikasi keluaran dari *language resources* tersebut dengan spesifikasi masukan atau keluaran menggunakan "NICT Language Service Interface". *Wrapper* ini akan ditaruh pada "Language Grid Service Node", dan akan menerima *request* dari "Language Grid Core Node".

Saat "Language Grid Service Node" menerima *request* dari "Language Grid Core Node", "Language Grid Service Node" akan mengakses *language resource* yang ada di dalam *wrapper*. Peran *wrapper* dapat dilihat pada Gambar 0.7.



Gambar 0.7 Peran *language service wrapper* (NICT Language Grid Project, 2008)

1.3 *Morphological Analyzer*

Program *Morphological Analyzer* adalah sebuah program yang dapat memberi informasi tentang proses pembentukan suatu kata dengan memberikan kata dasar dari kata tersebut serta analisis morfologi dari kata tersebut berupa *tag-tag* yang berisi informasi sintaksis dan semantik dari kata tersebut (seperti kelas kata *noun*, kelas kata *verb*, *active voice*, dan lain-lain). Program *Morphological Analyzer* yang dibahas di sini adalah program *Morphological Analyzer* untuk bahasa Indonesia yang telah dikembangkan oleh Femphy Pisceldo di dalam tugas akhirnya pada pertengahan tahun 2008 (Pisceldo, 2008).

Terdapat dua buah perintah utama yang dapat diberikan pada program ini, yaitu perintah "**`apply up [kata]`**" dan "**`apply down [kata dasar]+[tag]...`**". Berikut adalah penjelasan dari kedua perintah tersebut.

Perintah "**`apply up [kata]`**" digunakan untuk menganalisis morfologi dari **`[kata]`**, yang merupakan sebuah kata dasar atau kata berimbuhan dalam bahasa Indonesia. Hasil analisis morfologi yang diberikan berupa kata dasar, kelas kata, dan, jika ada, *tag-tag* yang berisi informasi sintaksis dan semantik lain dari **`[kata]`**. Hasil analisis morfologi tersebut akan dipisahkan dengan karakter '+'. Sebagai contoh, perintah "**`apply up memukul`**" akan menghasilkan keluaran "**`pukul+Verb+AV`**", karena kata "memukul" memiliki kata dasar "pukul", termasuk dalam kelas kata kerja (*Verb*), dan termasuk *active voice*. Sebagai tambahan, jika terdapat lebih dari satu keluaran yang memungkinkan, semua keluaran tersebut akan ditampilkan oleh program ini. Jika tidak terdapat keluaran yang memungkinkan, program ini tidak akan mengeluarkan apa-apa.

Perintah "**`apply down [kata dasar]+[tag]...`**" digunakan untuk mencari kata dasar dan kata berimbuhan dalam bahasa Indonesia yang memiliki morfologi tertentu, yaitu memiliki kata dasar **`[kata dasar]`** dan termasuk dalam *tag* sintaksis dan semantik **`[tag]`**. Program ini akan mengeluarkan semua kata yang memenuhi morfologi tersebut. Sebagai contoh, perintah "**`apply down pukul+Verb+AV`**" akan menghasilkan keluaran "**`mengepukulkan`**", "**`menerpukulkan`**", "**`memperpukul`**",

"**memperpukulkan**", "**memperpukuli**", "**memberpukulkan**", "**memukul**", "**memukulkan**", dan "**memukuli**". Semua kata tersebut memiliki kata dasar yang sama, yaitu "pukul", sama-sama termasuk ke dalam kelas kata kerja (*Verb*), dan sama-sama termasuk *active voice*. Jika tidak terdapat kata yang memungkinkan untuk dijadikan keluaran, program ini tidak akan mengeluarkan apa-apa.