BAB 2

LANDASAN TEORI

Pada bab ini, penulis memaparkan beberapa teori yang relevan dengan perancangan aplikasi *help desk* ini. Teori-teori tersebut di antaranya mengenai *help desk* itu sendiri, konsep intranet, konsep *Model-View-Controller*, tahapan dan proses-proses dalam *Natural Language Processing*, kaidah-kaidah dalam tata bahasa Indonesia, diagram-diagram dalam *Unified Modeling Language*, metodologi penelitian dengan *System Development Life Cycle*, dan teori tentang Interaksi Manusia Komputer.

2.1 *Help desk*

Help desk merupakan sumber daya informasi dan asistensi yang menangani permasalahan berkaitan dengan komputer atau produk lainnya yang sejenis. Perusahaan sering memberikan dukungan *help desk* kepada pelanggan mereka melalui sebuah nomor bebas pulsa, *website*, atau *email*. Sementara menurut Wooten (2001, p5), *help desk* merupakan organisasi formal yang memberikan fungsi dukungan kepada *user* dari produk, layanan, atau teknologi perusahaan.

Help desk memiliki beberapa fungsi. Biasanya, *help desk* mengatur permintaan melalui perangkat lunak, seperti *incident tracking system*, yang memungkinkan untuk mengecek permintaan *user* melalui sebuah nomor tiket yang unik. Tiket merupakan sebuah *file* yang diisi dengan sebuah *issue tracking system* yang berisi informasi tentang dukungan yang diberikan oleh staf teknis atau pihak ketiga dari *end user*. *End user* merupakan pihak yang melaporkan adanya kejadian

yang mencegah mereka untuk bekerja dengan komputer mereka, sebagaimana yang mereka harapkan seharusnya bisa dilakukan. Tiket biasanya dibuat di dalam lingkungan *help desk* atau *call center*. Tiket akan memiliki nomor referensi yang unik, yang disebut juga dengan *case*, *issue*, atau *call log number* yang digunakan untuk mengizinkan *user* atau staf teknis untuk menempatkan, menambah, atau memberitahukan status dari permintaan *user* dengan cepat.

Perangkat lunak *help desk* ini seringkali menjadi peralatan yang bermanfaat ketika digunakan untuk menemukan, menganalisis, dan mengeliminasi permasalahan yang kerap terjadi dalam lingkungan komputerisasi perusahaan.

User memberitahu *help desk* tentang permasalahannya, kemudian *help desk* mengeluarkan tiket yang memiliki detail permasalahan. Jika teknisi level pertama dapat menyelesaikan permasalahan, maka status tiket ditutup dan diperbarui dengan dokumentasi solusi sebagai referensi bagi teknisi lainnya di masa yang akan datang. Jika permasalahan perlu diperluas, dilakukan pembaruan, dan dicatat apa saja yang sudah diusahakan oleh teknisi, lalu dikirim ke teknisi level ke dua.

Dari penelitian pada pertengahan tahun 1990 oleh Middleton di The Robert Gordon University, ditemukan bahwa banyak organisasi mulai menyadari bahwa nilai sebenarnya dari *help desk* tidak semata-mata berasal dari respon reaktif terhadap permasalahan *user*, tetapi dari posisi unik *help desk* yang berkomunikasi sehari-hari dengan sejumlah besar pelanggan atau karyawan. Hal ini memberikan kemampuan bagi *help desk* untuk memonitor lingkungan *user* dari permasalahan teknis hingga ke kepuasan *user*. (http://en.wikipedia.org/wiki/Help_desk).

2.2 Internet dan Intranet

2.2.1 Internet

Internet adalah sebuah jaringan dari kumpulan jaringan, yang bertukar informasi tanpa terlihat, dengan menggunakan standardisasi dan protokol yang terbuka dan tanpa pemilik. Internet merupakan kumpulan dari banyak jaringan komputer tunggal yang dimiliki oleh pemerintah, universitas, grup nonprofit, dan perusahaan. Internet juga merupakan sebuah jaringan *packet-switched* yang menggunakan *Transmission Control Protocol/Internet Protocol* (TCP/IP). *User* dapat terkoneksi ke Internet melalui sebuah *server* LAN, SLIP/PPP, atau melalui sebuah layanan *online* (*Internet Service Provider*).

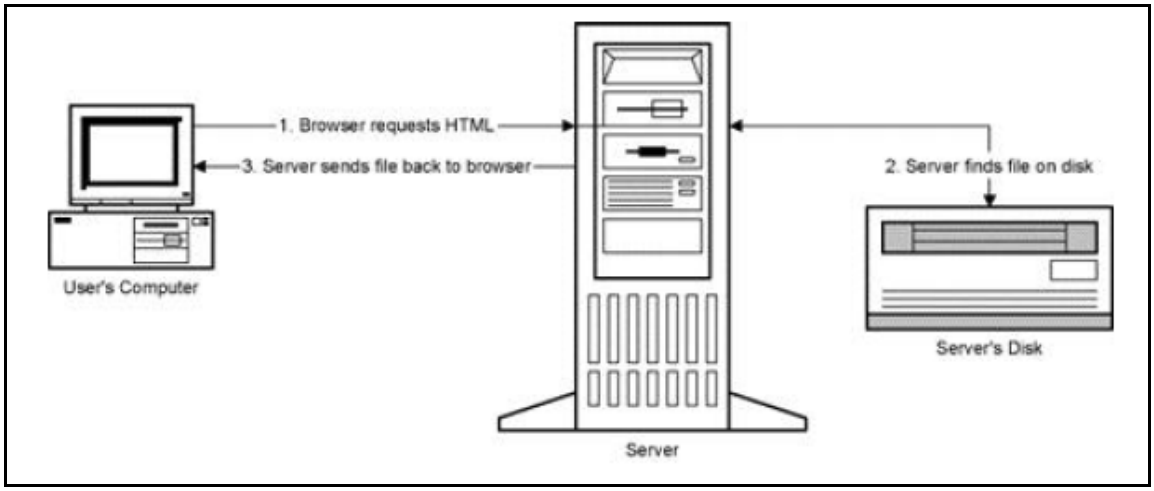
Internet menyediakan tiga tipe layanan utama: komunikasi, *information retrieval*, dan layanan *Web*. Layanan komunikasi meliputi *electronic mail* (*email*), USENET *newsgroup*, LISTSERV, *chatting*, Telnet, *Internet telephony*, dan *Internet fax*. Layanan *Information Retrieval* meliputi gophers, Archie, WAIS, *File Transfer Protocol* (FTP), dan Veronica. Sedangkan layanan *Web* adalah aplikasi perangkat lunak yang dikirim sebagai layanan melalui Internet. (Turban, Rainer, dan Potter, 2003, p230).

2.2.2 *World Wide Web*

Web adalah sebuah sistem yang secara universal digunakan untuk menyimpan, menerima kembali, memformat, dan menampilkan informasi melalui sebuah arsitektur klien/*server*. *Web* menangani segala jenis

informasi digital, termasuk teks, *hypermedia*, grafik, dan suara, dan sangat mudah digunakan karena menggunakan *Graphical User Interface* (GUI). Istilah *Web* tidak sama dengan Internet. Internet berfungsi sebagai mekanisme transportasi, sedangkan *Web* adalah aplikasi yang menggunakan fungsi transportasi tersebut. Aplikasi yang lain juga berjalan di Internet, misalnya *email* yang sangat populer digunakan. (Turban, Rainer, dan Potter, 2003, pp230-231).

Transaksi *Web* melibatkan dua hal: *browser* dan *server*. *Browser* merupakan program sederhana dan ringan yang memungkinkan *user* untuk bernavigasi melalui data. Data dapat berupa *plain text*, HTML, gambar, dan lain-lain. *Browser* akan me-*render* semua data dalam suatu cara di mana manusia dapat mengerti dan berinteraksi dengannya. Sementara *server* merupakan program yang lebih kompleks. *Server* bertanggung jawab untuk mencari data yang diminta *browser*, memaketkan data untuk transmisi, dan mengirimkannya kembali ke *browser*.



Gambar 2.1 Hubungan Antara *Browser* dengan *Server*

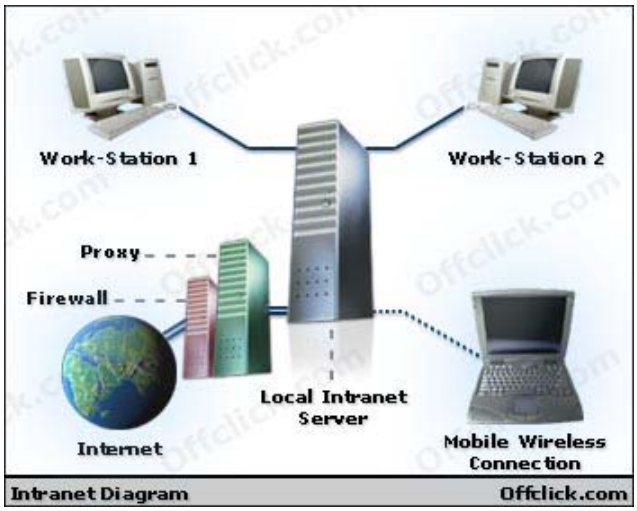
2.2.3 Intranet

Sebuah intranet adalah sebuah jaringan privat (*private network*) yang menggunakan protokol-protokol Internet (TCP/IP), untuk membagi informasi rahasia perusahaan atau operasi dalam perusahaan tersebut kepada karyawannya. Kadang-kadang, istilah intranet hanya merujuk kepada layanan yang terlihat, yakni situs *Web* internal perusahaan. Untuk membangun sebuah intranet, maka sebuah jaringan haruslah memiliki beberapa komponen yang membangun Internet, yakni protokol Internet (protokol TCP/IP, alamat IP, dan protokol lainnya), klien dan juga *server*. Protokol HTTP dan beberapa protokol Internet lainnya (FTP, POP3, atau SMTP) umumnya merupakan komponen protokol yang sering digunakan. Umumnya, sebuah intranet dapat dipahami sebagai sebuah "versi pribadi dari jaringan Internet", atau sebagai sebuah versi dari Internet yang dimiliki oleh sebuah organisasi. (http://id.wikipedia.org/wiki/Intranet).

Menurut Turban, Rainer, dan Potter (2003, pp222-224), intranet adalah jaringan privat yang menggunakan perangkat lunak Internet dan protokol TCP/IP. Intinya, intranet adalah Internet privat, atau grup dari segmen privat dari jaringan Internet publik, digunakan oleh orang-orang yang diberikan otorisasi untuk menggunakan jaringan tersebut.

Keamanan intranet merupakan suatu hal yang penting. Perusahaan dapat mencegah gangguan yang tidak diinginkan dengan berbagai cara. *Public key security* digunakan sebagai perantara otorisasi untuk masuk ke dalam intranet. Ada dua bagian di dalamnya, yaitu enkripsi dan *digital certificate*. Enkripsi mengacaukan data yang keluar, sementara *digital*

certificate seperti kartu identitas elektronik yang memastikan bahwa orang yang mengakses intranet adalah *user* yang valid. Cara lain yang digunakan perusahaan untuk melindungi intranet adalah dengan menggunakan *firewall*. *Firewall* merupakan sebuah alat yang ditempatkan di antara jaringan internal perusahaan (intranet) dan jaringan eksternal (Internet).



Gambar 2.2 Diagram Intranet

2.3 *Model-View-Controller*

Bagian dalam kode dari suatu aplikasi yang sering mengalami perubahan adalah *user interface*. Bagian inilah yang paling terlihat oleh *user* dan menjadi titik fokus pengubahan berdasarkan kemudahan penggunaan. *Business logic* yang rumit pada *user interface* akan mengakibatkan perubahan pada *user interface* menjadi lebih kompleks dan memperbesar resiko terjadinya kesalahan.

Pola *Model-View-Controller* (MVC), yang adalah suatu pola arsitektur yang digunakan dalam *Software Engineering*, menyediakan sebuah solusi terhadap permasalahan tersebut dengan memisahkan antara *business logic, presentation*

logic, dan *control logic* yang kemudian disebut sebagai bagian *Model, View,* dan *Controller*.

Model

Model merepresentasikan data atau informasi dari suatu aplikasi. Dengan pola MVC, data dikumpulkan pada suatu area tersendiri dan tidak tersebar di dalam keseluruhan lingkup aplikasi. Hal ini memberikan keuntungan dalam proses *maintenance* aplikasi. Selain itu, bagian ini juga dapat digunakan kembali oleh aplikasi lain yang memiliki kegunaan hampir sama.

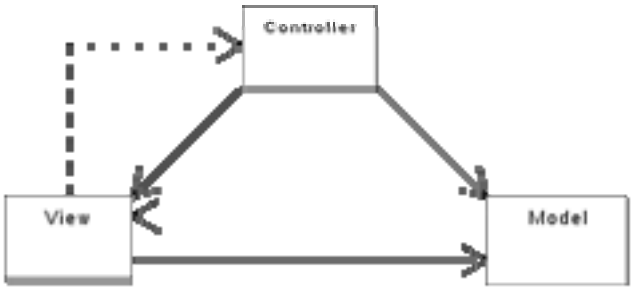
View

View mengatur bagian *user interface* pada aplikasi. Bagian ini mengakses data melalui *Model,* kemudian mengatur bagaimana data tersebut akan dipresentasikan. *View* berperan untuk mengatur konsistensi presentasi tersebut ketika terjadi perubahan pada *Model. Multiple Interfaces* pada aplikasi juga dimungkinkan dengan adanya bagian *View* ini. Jika inti dari aplikasi terletak pada bagian lain (*Model*), maka *Multiple Interfaces* dapat memiliki tampilan yang berbeda namun mengeksekusi komponen *Model* sesuai fungsionalitas yang diharapkan.

Controller

Controller mengatur komunikasi data dan aliran bisnis yang digunakan untuk memanipulasi data baik yang berasal dari *Model* maupun yang menuju ke sana. dengan menggunakan komponen terpisah untuk menampung detail dari transisi *layer*, *View* dapat didesain tanpa harus memperhatikan bagian lain secara berlebih.

Interaksi antar komponen *View* terabstraksi dalam *Controller*. Bila dilakukan *update* terhadap komponen *Model,* detail tersebut dihapus dari *layer* presentasi. Lalu *layer* presentasi akan kembali pada fungsi utamanya untuk menampilkan data. Detail tentang bagaimana data dari *user* mengubah ketetapan aplikasi disembunyikan oleh *Controller*. Hal ini memisahkan dengan jelas antara *presentation logic* dengan *business logic.*



Gambar 2.3 *Model-View-Controller*

2.4 Intelegensia Semu

2.4.1 Pengertian Intelegensia Semu

Intelegensia Semu (IS) atau Kecerdasan Buatan atau *Artificial Intelligence* (AI) merupakan salah satu bagian ilmu komputer yang membuat agar mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang dilakukan oleh manusia.

IS meliputi bagaimana mempelajari manusia dalam melakukan proses suatu gagasan atau pekerjaan dan bagaimana mempresentasikan proses atau gagasan tersebut kepada mesin seperti komputer.

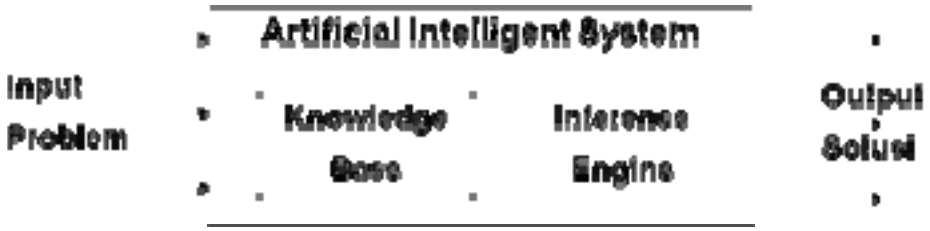
Beberapa definisi dari IS dapat dilihat melalui tabel berikut ini (Russell dan Norvig, 2003, p2):

Tabel 2.1 Beberapa Pengertian Intelegensia Semu

Sistem yang berpikir seperti manusia	**Sistem yang berpikir secara rasional**
"Usaha baru yang menarik untuk membuat komputer berpikir… mesin dengan pikiran, dalam kecerdasan yang lengkap dan akurat." (Haugeland, 1985) "[Proses] aktivitas di mana kita berasosiasi dengan pikiran manusia, aktivitas seperti pengambilan keputusan, pemecahan masalah, pembelajaran…" (Bellman, 1978)	"Ilmu dari fakultas mental melalui penggunaan model komputasional." (Charniak dan McDermott, 1985) "Ilmu mengenai komputasi yang membuatnya mungkin untuk mengerti, memberi alasan, dan beraksi." (Winston, 1992)
Sistem yang beraksi seperti manusia	**Sistem yang beraksi secara rasional**
"Seni membuat mesin yang melakukan fungsi yang membutuhkan kecerdasan ketika dilakukan oleh manusia." (Kurzweil, 1990) "Ilmu tentang bagaimana membuat komputer melakukan sesuatu di mana, pada suatu waktu, manusia melakukannya lebih baik." (Rich dan Knight, 1991)	"Kecerdasan komputasional adalah ilmu tentang desain dari agen kecerdasan." (Poole *et al.*, 1998) "AI… fokus dengan perilaku cerdas dalam artifak." (Nilssom, 1998)

Dalam memecahkan suatu masalah (*input*), ada suatu proses yang perlu dilakukan agar dapat menghasilkan solusi (*output*) tertentu. Pada proses IS, agar komputer bisa bertindak seperti dan sebaik manusia, maka komputer juga harus diberi bekal pengetahuan dan mempunyai kemampuan untuk menalar. Berdasarkan hal tersebut, dapat disimpulkan bahwa ada dua komponen utama yang dibutuhkan dalam melakukan aplikasi IS, yaitu basis pengetahuan (*knowledge base*) dan motor inferensi (*inference engine*). Basis pengetahuan berisi fakta-fakta, teori, pemikiran, dan

hubungan antara satu dengan lainnya. Sedangkan motor inferensi digunakan untuk menarik kesimpulan berdasarkan pengalaman. Proses dalam IS tersebut dapat digambarkan sebagai berikut:



Gambar 2.4 Proses Intelegensia Semu

Pada dasarnya, segala sesuatu, terlebih bila berkaitan dengan metode, memiliki keuntungan dan kelemahan masing-masing. Demikian pula halnya dengan IS. Jika dibandingkan dengan Intelegensia Alami, maka keuntungan dan kelemahan IS adalah:

Tabel 2.2 Keuntungan dan Kelemahan Intelegensia Semu

Keuntungan	Kelemahan
1. Lebih bersifat permanen 2. Lebih mudah diduplikasi dan disebarkan 3. Lebih murah jika dibandingkan dengan Intelegensia Alami 4. Lebih bersifat konsisten 5. Dapat didokumentasikan 6. Dapat mengerjakan pekerjaan lebih cepat 7. Dapat mengerjakan pekerjaan lebih baik	1. Tidak kreatif, karena untuk menambah pengetahuan harus dilakukan melalui sistem yang dibangun 2. Harus bekerja dengan input-input simbolik sehingga tidak memungkinkan untuk menggunakan pengalaman secara langsung 3. Pemikiran sangat terbatas

2.4.2 Bidang-bidang Cakupan Intelegensia Semu

a. Sistem Pakar (*Expert System*): komputer sebagai sarana untuk menyimpan pengetahuan para pakar sehingga komputer mempunyai keahlian dan bisa menyelesaikan permasalahan seperti pakar.

b. Pengolahan Bahasa Alami (*Natural Language Processing*): komputer dapat berkomunikasi dengan manusia dengan menggunakan bahasa sehari-hari. Sehingga komputer dapat menerjemahkan dari satu bahasa ke bahasa lain.

c. Pengenalan ucapan (*Speech Recognition*): komputer dapat melakukan komunikasi dengan manusia melalui suara (ucapan).

d. *Computer Vision*: komputer dapat melakukan interpretasi terhadap gambar atau pola yang tampak.

e. *Intelligent Computer-Aided Instruction*: komputer dapat digunakan sebagai tutor.

f. Robotikan dan sistem sensor (*Robotic & Sensory System*): komputer digunakan sebagai pusat pengendali dalam sistem robot cerdas.

g. *Game playing*: komputer sebagai pemain dan dapat belajar dari permainan yang pernah dilakukan.

h. *Fuzzy Logic*: komputer dapat mengambil keputusan dalam hal-hal yang tidak jelas atau kondisi di antara Ya dan Tidak.

2.5 *Natural Language Processing*

2.5.1 Pengertian *Natural Language Processing*

Natural language atau bahasa alami adalah bahasa yang dipelajari manusia dari lingkungannya dan digunakan untuk berkomunikasi.

Natural Language Processing (NLP) merupakan sebuah bidang dari ilmu komputer yang fokus dengan interaksi antara komputer dengan bahasa (alami) manusia. Sistem *Natural Language Generation* (NLG) mengubah informasi dari *database* komputer ke dalam bahasa manusia yang dapat dibaca. Sistem *Natural Language Understanding* (NLU) mengubah contoh bahasa manusia ke dalam representasi yang lebih formal sehingga memudahkan program komputer untuk melakukan manipulasi. Banyak permasalahan dengan NLP diaplikasikan, baik untuk *generation* maupun *understanding*. Sebagai contoh, sebuah komputer harus mampu membuat model morfologi (struktur kata) supaya mengerti kalimat dalam bahasa Inggris, tetapi sebuah model morfologi juga diperlukan untuk menghasilkan kalimat dalam bahasa Inggris yang benar secara gramatikal.

NLP memiliki *overlap* yang signifikan dengan bidang linguistik komputasional, dan sering dipertimbangkan sebagai sub bidang dari IS. Terminologi bahasa alami digunakan untuk membedakan bahasa manusia (seperti bahasa Indonesia, Inggris, atau Spanyol, dan sebagainya) dari bahasa formal atau bahasa komputer (seperti C++, Java, atau LISP). (http://en.wikipedia.org/wiki/Natural_language_processing).

Menurut Rich dan Knight (1991, p380) ada beberapa tahap-tahap dalam NLP, yaitu:

a. Analisis Morfologi

Kata-kata tunggal dianalisis ke dalam komponennya, dan token non kata, seperti tanda baca, dipisahkan dari kata-kata.

b. Analisis Sintaksis

Rangkaian linear kata-kata diubah ke dalam struktur yang menunjukkan bahwa kata-kata saling berelasi satu sama lain. Beberapa rangkaian kata mungkin ditolak jika melanggar aturan bahasa tentang bagaimana mengkombinasi kata-kata. Misalnya, sebuah penganalisis sintaksis dalam bahasa Inggris akan menolak kalimat "*Boy the go to the to store"*.

c. Analisis Semantik

Struktur yang dibuat oleh penganalisis sintaksis menentukan suatu arti. Dengan kata lain, pemetaan dibuat antara struktur sintaksis dan objek dalam domain tugas. Struktur yang tidak memiliki pemetaan seperti demikian, akan mungkin ditolak. Misalnya, kalimat "*Colorless green ideas sleep furiously*" akan ditolak sebagai ganjil secara semantik.

d. *Discourse Integration*

Arti dari sebuah kalimat tunggal mungkin tergantung pada kalimat yang mendahuluinya dan mungkin mempengaruhi arti dari kalimat yang mengikutinya. Misalnya, kata "*it*" pada kalimat "*John wanted it*", bergantung pada konteks percakapan sebelumnya, sementara kata "*John*" mungkin mempengaruhi kalimat sesudahnya (seperti "*He always had*").

e. Analisis Pragmatis

Struktur yang menggambarkan apa yang dikatakan, diinterpretasi kembali untuk menentukan apa arti sebenarnya. Misalnya kalimat "*Do you know what time is it*?" akan diinterpretasi sebagai permintaan untuk memberitahu tentang waktu.

2.5.2 Hirarki Chomsky

Dalam ilmu komputer, terutama dalam bidang *formal language*, hirarki Chomsky merupakan hirarki yang berisi *class* dari *formal grammar*. Sebuah *formal grammar* terdiri dari:

- Kumpulan terbatas (*finite set*) dari simbol terminal
- Kumpulan terbatas dari simbol nonterminal
- Kumpulan terbatas dari *production rule* dengan sisi kiri dan kanan berisi deretan dari simbol-simbol tersebut
- Sebuah simbol *start*

Sebuah *formal grammar* menghasilkan *formal language*, yang merupakan kumpulan deretan simbol dengan panjang terbatas (misalnya *string*) yang mungkin dibangun dengan mengaplikasikan *production rule* ke deretan simbol lainnya yang pada awalnya hanya berisi simbol *start*. Sebuah aturan mungkin diterapkan pada deretan simbol dengan cara menggantikan simbol yang muncul di sisi kiri dengan simbol yang muncul di sisi kanan. Aplikasi deretan aturan ini disebut *derivation*. *Grammar* yang demikian menghasilkan *formal language*: semua kata semata-mata berisi simbol terminal yang dapat dicapai oleh *derivation* dari simbol *start*.

Nonterminal biasanya direpresentasikan dengan huruf besar, terminal dengan huruf kecil, dan simbol *start* dengan *S*. Contohnya *grammar* dengan terminal {*a,b*}, nonterminal {*S,A,B*}, *production rule*

$S \rightarrow ABS$

$S \rightarrow \varepsilon$ (ε adalah *string* kosong)

$BA \rightarrow AB$

$BS \rightarrow b$

$Bb \rightarrow bb$

$Ab \rightarrow ab$

$Aa \rightarrow aa$

dan simbol *start S*, menghasilkan *language* dari semua kata dengan bentuk $a^n b^n$ (misalnya *n* jumlah dari *a* diikuti oleh *n* jumlah dari *b*).

Hirarki Chomsky terdiri dari beberapa level berikut:

- *Grammar* tipe-0 (*unrestricted grammar*)

  Meliputi semua *grammar*. Ini menghasilkan semua *language* yang dapat diterima oleh sebuah mesin Turing. *Language* ini juga dikenal sebagai *recursively enumerable language*.

- *Grammar* tipe-1 (*context-sensitive grammar*)

  Menghasilkan *context-sensitive language*. *Grammar* ini memiliki aturan dari bentuk $\alpha A \beta \rightarrow \alpha \gamma \beta$ dengan *A* sebagai nonterminal, sementara α, β, dan γ sebagai *string* terminal dan nonterminal. *String* α, β dapat bernilai kosong, sedangkan γ tidak dapat bernilai kosong. Aturan $S \rightarrow \varepsilon$ diijinkan jika *S* tidak muncul di sisi kanan dari aturan apapun.

- *Grammar* tipe-2 (*context-free grammar*)

  Menghasilkan *context-free language*. Ini ditetapkan dengan aturan dari bentuk $A \rightarrow \gamma$ dengan A sebagai nonterminal dan γ sebagai *string* terminal dan nonterminal. *Language* ini merupakan semua *language* yang dapat diterima sebagai *non-deterministic pushdown automaton*. *Context-free language* merupakan dasar teori untuk sintaks dari bahasa pemrograman.

- *Grammar* tipe-3 (*regular grammar*)

  Menghasilkan *regular language*. *Grammar* seperti ini membatasi aturan untuk sebuah nonterminal tunggal di sisi kiri, dan di sisi kanan yang berisi terminal tunggal yang kemungkinan diikuti oleh sebuah nonterminal tunggal. Aturan $S \rightarrow \varepsilon$ juga diperbolehkan di sini jika S tidak muncul pada sisi kanan dari aturan manapun. *Language* ini merupakan semua *language* yang dapat diputuskan oleh sebuah *finite state automaton*.

(http://en.wikipedia.org/wiki/Chomsky_hierarchy).

2.5.3 *Context-free Grammar*

Sebuah *context-free grammar* (CFG) menyediakan mekanisme sederhana dan tepat untuk menggambarkan metode di mana frase dalam beberapa bahasa alami dibangun dari blok yang lebih kecil, menangkap "struktur blok" kalimat dalam cara yang alami. Aspek "struktur blok" yang ditangkap oleh CFG bersifat sangat fundamental terhadap *grammar*, di mana terminologi sintaks dan *grammar* sering diidentifikasi dengan aturan

CFG, terutama dalam ilmu komputer. Batasan formal yang tidak tertangkap oleh *grammar* kemudian dipertimbangkan sebagai bagian dari bahasa semantik.

CFG cukup sederhana untuk memperbolehkan pembentukan algoritma *parsing* yang efisien. Misalnya untuk *string* yang diberikan, CFG menentukan apakah dan bagaimana *string* tersebut dapat dihasilkan dari *grammar*. Earley *parser* merupakan contoh dari algoritma tersebut.

CFG *G* merupakan sebuah 4-*tuple* (deretan nilai):

$G = (V, \sum, R, S)$ di mana

1. *V* adalah kumpulan terbatas dari karakter nonterminal atau variabel, merepresentasikan tipe berbeda dari frase atau klausa dalam kalimat.
2. $\sum$ adalah kumpulan terbatas dari terminal, menggambarkan isi yang sebenarnya dari kalimat.
3. *R* adalah relasi dari *V* sampai $(V \cup \Sigma)^*$ seperti $\forall w \in (V \cup \Sigma)^* : (S, w) \in R$
4. *S* adalah variabel *start*, digunakan untuk merepresentasikan keseluruhan kalimat (atau program). Merupakan elemen dari *V*.

(http://en.wikipedia.org/wiki/Context-free_grammar).

Menurut Russel dan Norvig (2003, pp795-797), sebelum membuat suatu *grammar* (misalnya dalam fragmen Inggris), pertama didefinisikan *lexicon* terlebih dahulu. *Lexicon* adalah daftar dari kata-kata yang diizinkan. Kata-kata ini dikelompokkan ke dalam kategori atau bagian dari

percakapan *familiar* bagi *user*. Berikut contoh *lexicon* dalam fragmen bahasa Inggris:

Noun	→	*stench* | *breeze* | *glitter* | *nothing* | *agent* | *wumpus* | *pit* | *pits* | *gold* | *east* | ...
Verb	→	*is* | *see* | *smell* | *shoot* | *feel* | *stinks* | *go* | *grab* | *carry* | *kill* | *turn* | ...
Adjective	→	*right* | *left* | *east* | *dead* | *back* | *smelly* | ...
Adverb	→	*here* | *there* | *nearby* | *ahead* | *right* | *left* | *east* | *south* | *back* | ...
Pronoun	→	*me* | *you* | *I* | *it* | ...
Name	→	*John* | *Mary* | *Boston* | *Aristotle* | ...
Article	→	*the* | *a* | *an* | ...
Preposition	→	*to* | *in* | *on* | *near* | ...
Conjunction	→	*and* | *or* | *but* | ...
Digit	→	0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Gambar 2.5 Contoh *Lexicon*

Tahap selanjutnya adalah membuat *grammar*. Contoh *grammar*:

S	→ *NP VP*	*I + feel a breeze*
	| *S Conjuction S*	*I feel a breeze + and + I smell a wumpus*
NP	→ *Pronoun*	*I*
	| *Name*	*John*
	| *Noun*	*pits*
	| *Article Noun*	*the + wumpus*
	| *Digit Digit*	*3 4*
	| *NP PP*	*the wumpus + to the east*
	| *NP RelClause*	*the wumpus + that is smelly*
VP	→ *Verb*	*stinks*
	| *VP NP*	*feel + a breeze*
	| *VP Adjective*	*is + smelly*
	| *VP PP*	*turn + to the east*
	| *VP Adverb*	*go + ahead*
PP	→ *Preposition NP*	*to + the east*
RelClause	→ *that VP*	*that + is smelly*

Gambar 2.6 Contoh *Grammar* dengan Contoh Frase di Setiap Aturan

2.5.4 Analisis Morfologi

Untuk menganalisis kalimat *input*-an, program pertama kali memecah kalimat ke dalam kata tunggal. Program memeriksa *input* dan mencari spasi dan tanda baca untuk mengidentifikasi kata tunggal. Pada

tahap ini, kata *input*-an dibagi ke dalam unit yang lebih kecil yang disebut morfem. Morfem merupakan unit terkecil dari bahasa. Sebuah morfem dapat berupa kata itu sendiri, yang disebut dengan morfem bebas. Sebagai contoh, kata "*computer*" adalah sebuah morfem. Di sisi lain, "*computers*" terdiri dari dua morfem, kata dasarnya sendiri "*computer*", dan huruf "*s*" yang ditambahkan di akhir kata yang mengidentifikasikan jamak. Huruf "*s*" merupakan tipe morfem yang disebut *bound morpheme*. *Bound morpheme* biasanya merupakan awalan dan akhiran yang digunakan pada kata dasar untuk memodifikasi arti. (Turban, 1992, p282).

Tidak berbeda jauh dengan pendapat Turban, menurut Rich dan Knight (1991, p381), analisis morfologi harus dapat melakukan hal berikut:

- Memisahkan kata "*Bill's*" ke dalam kata benda yang benar "*Bill*" dan akhiran kepunyaan " *'s*".
- Mengenal deretan "*.init*" sebagai ekstensi *file* yang berfungsi sebagai kata sifat dalam kalimat.

Dalam menganalisis struktur morfologi dari kata-kata dalam bahasa Indonesia, diperlukan suatu algoritma *stemming* yang sesuai. *Stemming* digunakan untuk mengubah variasi kata ke bentuk kata dasarnya dengan mengaplikasikan aturan-aturan morfologi. Tidak seperti bahasa Inggris, di mana peran akhiran mendominasi pembentukan kata asal, bahasa Indonesia bergantung baik pada awalan maupun akhiran untuk menghasilkan kata-kata baru. Karena itu, untuk *stemming* sebuah kata asal dalam bahasa Indonesia untuk memperoleh kata dasarnya, perlu diperhatikan awalan dan akhiran pada kata asal tersebut.

Algoritma Nazief dan Adriani

Berdasarkan penelitian Asian, William, dan Tahaghoghi (2005, pp2-3), disebutkan bahwa algoritma Nazief dan Adriani didasarkan pada aturan morfologi komprehensif yang mengelompokkan dan mengenkapsulasi imbuhan yang diperbolehkan dan yang tidak diperbolehkan, termasuk awalan, akhiran, sisipan, dan imbuhan gabung. Algoritma ini juga mendukung pengkodean ulang, sebuah pendekatan untuk mengembalikan sebuah huruf awal yang telah dibuang sebelumnya dari kata dasar untuk menunda terlebih dahulu sebuah awalan. Sebagai tambahan, pada algoritma ini juga digunakan kamus kata-kata dasar untuk memeriksa jika *stemming* sudah mencapai kata dasar.

Ada tiga komponen dasar dalam algoritma Nazief dan Adriani, yaitu: pengelompokkan imbuhan, penggunaan aturan (serta pengecualiannya), dan sebuah kamus.

Pengelompokkan imbuhan dibentuk menjadi kategori berikut:

a. *Inflection Suffix*

Kumpulan akhiran yang tidak mengubah kata dasar. *Inflection suffix* dibagi lagi menjadi:

- *Particle* (P): -lah, -kah. Misalnya pada kata "duduklah".
- *Possesive Pronoun* (PP): -ku, -mu, -nya. Misalnya pada kata "ibunya".

P dan PP dapat muncul bersama, di mana PP muncul sebelum P. Sebuah kata dapat memiliki lebih dari satu P maupun PP, dan dapat

diaplikasikan secara langsung ke kata dasar atau ke kata yang memiliki *derivation suffix*.

b. *Derivation suffix*

Kumpulan akhiran yang diaplikasikan secara langsung ke kata dasar. Hanya ada satu *derivation suffix* per kata. Sebagai contoh, kata "lapor" dapat diberi akhiran -kan sehingga menjadi "laporkan". Selain itu, dapat pula diberi akhiran dengan *inflection suffix* -lah sehingga menjadi "laporkanlah".

c. *Derivation prefix*

Kumpulan awalan yang diaplikasikan baik secara langsung ke kata dasar maupun ke kata yang memiliki sampai dua *derivation prefix* lainnya. Misalnya *derivation prefix* "mem-" dan "per-" dapat ditambahkan pada kata "indahkannya" sehingga menghasilkan "memperindahkannya".

Ada aturan yang merupakan pengecualian dan batasan dalam algoritma ini:

a. Tidak semua kombinasi mungkin untuk diperbolehkan. Sebagai contoh, setelah kata yang memiliki awalan di-, maka tidak diperbolehkan untuk dipakai akhiran -an. Daftar lengkapnya dapat dilihat di tabel berikut.

Tabel 2.3 Imbuhan Gabung yang Tidak Dibolehkan

Awalan	Akhiran yang tidak dibolehkan
be-	-i
di-	-an
ke-	-i, -kan
me-	-an
se-	-i, -kan
te-	-an

b. Imbuhan yang sama tidak dapat diulang. Misalnya setelah sebuah kata diberi awalan te-, maka tidak mungkin untuk mengulang menggunakan awalan te- lagi pada kata tersebut.

c. Jika sebuah kata memiliki satu atau dua karakter, maka tidak dilakukan *stemming*.

d. Menambah sebuah awalan mungkin mengubah kata dasar atau awalan yang telah digunakan sebelumnya. Pertimbangkan meng-, yang memiliki variasi mem-, meng-, meny-, dan men-. Beberapa di antaranya dapat mengubah awalan dari suatu kata. Misalnya, pada kata dasar "sapu". Jika digunakan awalan meny- untuk menghasilkan kata "menyapu", berarti telah membuang huruf "s".

Proses *stemming* dalam algoritma Nazief dan Adriani adalah sebagai berikut:

1. Kata *input*-an dicari di dalam kamus. Jika ditemukan, maka diasumsikan bahwa kata tersebut merupakan kata dasar.

2. Membuang *inflection suffix* (-lah, -kah, -ku, -mu, -nya). Jika berhasil dan akhirannya adalah P (-lah, -kah), maka tahap ini diulangi lagi untuk membuang *inflection suffix* PP (-ku, -mu, atau -nya).
3. Membuang *derivation suffix* (-i, -an). Jika berhasil, maka dilakukan tahap 4. Jika tahap 4 tidak berhasil:
    a. Jika akhiran -an dibuang, dan huruf terakhir dari kata adalah -k, maka -k juga dibuang dan diulangi lagi tahap keempat. Jika gagal, dilakukan tahap 3b.
    b. Akhiran yang telah dibuang sebelumnya (-i, -an, -kan), dikembalikan.
4. Membuang *derivation prefix*.
    a. Jika akhiran dibuang pada tahap 3, maka dilakukan pemeriksaan imbuhan gabung yang tidak diperbolehkan berdasarkan **Tabel 2.3**. Jika ditemukan yang sesuai, maka algoritma mengeluarkan hasil.
    b. Jika awalan saat ini sesuai dengan awalan apapun sebelumnya, maka algoritma mengeluarkan hasil.
    c. Jika tiga awalan sebelumnya telah dibuang, maka algoritma mengeluarkan hasil.
    d. Jenis awalan ditentukan dengan cara berikut:
        - Jika awalan kata adalah di-, ke-, se-, maka jenis awalannya berturut-turut adalah di, ke, se.
        - Jika awalan adalah te-, be-, me-, pe-, maka dibutuhkan proses tambahan dalam menghasilkan kumpulan karakter untuk

menentukan jenis awalan. Sebagai contoh, aturan untuk awalan te- pada tabel berikut:

Tabel 2.4 Jenis Awalan Untuk Kata Berawalan Te-

Karakter yang mengikuti				**Jenis awalan**
Set 1	**Set 2**	**Set 3**	**Set 4**	
-r-	-r-	-	-	tidak ada
-r-	Vokal	-	-	ter-luluh
-r-	bukan (-r- atau vokal)	-er-	Vokal	ter
-r-	bukan (-r- atau vokal)	-er-	bukan vokal	tidak ada
-r-	bukan (-r- atau vokal)	bukan -er-	-	ter
bukan (vokal atau -r-)	-er-	vokal	-	tidak ada
bukan (vokal atau -r-)	-er-	bukan vokal	-	te

Misalkan pada kata "terlambat". Setelah membuang te-, akan dihasilkan "rlambat", kumpulan pertama karakter dihasilkan dari awalan menurut aturan Set 1. Dalam kasus ini, huruf yang mengikuti awalan te- adalah "r", dan ini cocok dengan lima baris pertama dari **Tabel 2.4** tersebut. Setelah "r" adalah "l" (Set 2), maka cocok dengan baris ketiga sampai kelima. Setelah "l" adalah "-ambat", mengeliminasi baris ketiga dan keempat untuk Set 3 dan menentukan bahwa jenis awalan "ter" ditunjukkan pada kolom paling kanan.

- Jika dua karakter pertama tidak cocok dengan di-, ke-, se-, te-, be-, me-, atau pe-, maka algoritma mengeluarkan hasil.

e. Jika jenis awalan adalah "tidak ada", maka algoritma mengeluarkan hasil. Jika bukan, maka jenis awalan dapat dilihat pada **Tabel 2.5**,

awalan yang dibuang pun ditemukan, dan awalan tersebut dibuang dari kata. **Tabel 2.5** hanya menunjukkan kasus sederhana dan yang sesuai dengan **Tabel 2.4**.

Tabel 2.5 Menentukan Awalan dari Jenis Awalan

Jenis awalan	Awalan yang dibuang
di	di-
ke	ke-
se	se-
te	te-
ter	ter-
ter-luluh	ter-

f. Jika kata dasar tidak ditemukan, lakukan tahap 4 secara berulang untuk pembuangan awalan lebih lanjut. Jika kata dasar ditemukan, maka algoritma mengeluarkan hasil.

g. Lakukan pengkodean ulang. Tahap ini bergantung pada jenis awalan dan dapat menghasilkan awalan yang berbeda pada kata yang di-*stem* dan dicek pada kamus. Misal pada jenis awalan "ter-luluh" pada **Tabel 2.4** dan **Tabel 2.5**. Pada kasus ini, setelah membuang "ter-", sebuah "r" ditambahkan pada kata. Jika kata baru ini tidak terdapat di dalam kamus, maka ulangi Langkah 4 untuk kata baru tersebut. Jika kata dasar tidak ditemukan, maka "r" dibuang dan "ter-" dikembalikan. Awalan di-*set* ke "tidak ada" dan algoritma mengeluarkan hasil.

5. Jika semua tahapan di atas berhasil dilalui, maka algoritma akan mengeluarkan hasil berupa kata dasar yang asli.

2.5.5 Analisis Sintaksis

Dalam ilmu komputer dan linguistik, *parsing*, atau lebih formal disebut analisis sintaksis, adalah proses menganalisis deretan *token* untuk menentukan struktur gramatikal *berdasarkan formal grammar* yang tersedia. Pada tahap ini, akan dilakukan analisis terhadap struktur sintaksis dari kalimat-kalimat. *Parsing* memverifikasi bahwa kalimat-kalimat terbentuk dengan baik secara sintaksis dan juga menentukan struktur bahasa. Dengan mengidentifikasi relasi linguistik yang utama seperti subjek-kata kerja, kata kerja-objek, dan kata benda-*modifier*, *parser* menyediakan sebuah *framework* untuk interpretasi semantik. Ini sering direpresentasikan dengan *parse tree*.

Sebuah *parser* adalah salah satu komponen *interpreter* atau *compiler* yang mengecek sintaks yang tepat dan membangun struktur data (seperti *parse tree*, *abstract syntax tree*, atau struktur hirarki lainnya) secara implisit di dalam *input token*. *Parser* sering menggunakan *lexical analyser* untuk membuat *token* dari deretan *input*-an karakter. *Parser* dapat diprogram sendiri atau dihasilkan semi otomatis dengan menggunakan alat seperti Yacc dari *grammar* yang ditulis dalam Backus-Naur *form*.
(http://en.wikipedia.org/wiki/Parsing).

2.5.5.1 Teknik dasar *parsing*

Ada dua teknik dasar dalam *parsing* (Russel dan Norvig, 2003, pp798-799), yaitu:

a. *Top-down parsing*

Pada teknik ini, dimulai dengan simbol *S* dan mencari *tree* yang memiliki kata-kata pada daunnya (karena *S* digambarkan di paling atas dari *tree*). *Top-down parsing* dapat didefinisikan sebagai berikut:

- *Initial state* adalah *parse tree* yang berisi akar *S* dan anak yang tidak diketahui: [*S*: ?]. Pada umumnya, setiap *state* dalam *search space* adalah sebuah *parse tree*.
- *Successor function* memilih *leftmost node* dalam *tree* dengan anak yang tidak diketahui. Kemudian akan mencari di *grammar* untuk aturan yang memiliki label *root* dari *node* pada sisi kiri. Untuk aturan yang demikian, *successor function* membuat *successor state* di mana ? digantikan oleh sebuah daftar yang sesuai pada aturan di sisi kanan. Sebagai contoh, dari **Gambar 2.6** ada dua aturan untuk *S*. Sehingga *tree* [*S*: ?] akan digantikan oleh dua *successor* berikut:

  [*S*: [*S*: ?][*Conjunction*: ?][*S*: ?]]

  [*S*: [*NP*: ?][*VP*: ?]]
- *Goal test* memeriksa daun dari *parse tree* yang sesuai dengan tepat untuk *input string*, dengan *input* yang tidak diketahui dan tidak ditemukan.

b. *Bottom-up parsing*

Pada teknik ini, dimulai dengan kata-kata dan mencari *tree* dengan akar *S*. Formulasi dari *bottom-up parsing* adalah sebagai berikut:

- *Initial state* adalah daftar dari kata-kata dalam *input string*, masing-masing ditampilkan sebagai *parse tree* yang merupakan *node* daun tunggal – contohnya: [*the, wumpus, is, dead*]. Pada umumnya, setiap *state* dalam *search space* adalah daftar dari *parse tree*.
- *Successor function* mencari pada setiap posisi *i* dalam daftar *tree* dan setiap aturan sisi kanan dari *grammar*. Jika *subsequence* dari daftar *tree* dimulai dengan *i* sesuai dengan sisi kanan, maka *subsequence* digantikan oleh sebuah *tree* baru yang kategorinya merupakan aturan sisi kiri dan anaknya adalah *subsequence*. Dengan mencocokkan, didapatkan bahwa kategori *node* sama dengan elemen di sisi kanan. Sebagai contoh dari **Gambar 2.5**, aturan *Article* → *the*, sesuai dengan *subsequence* yang berisi *node* pertama dari daftar [*the, wumpus, is, dead*], jadi sebuah *successor state* akan berupa [[*Article: the], wumpus, is, dead*].
- *Goal test* memeriksa *state* yang berisi *tree* tunggal dengan akar *S*.

2.5.5.2 Algoritma Earley

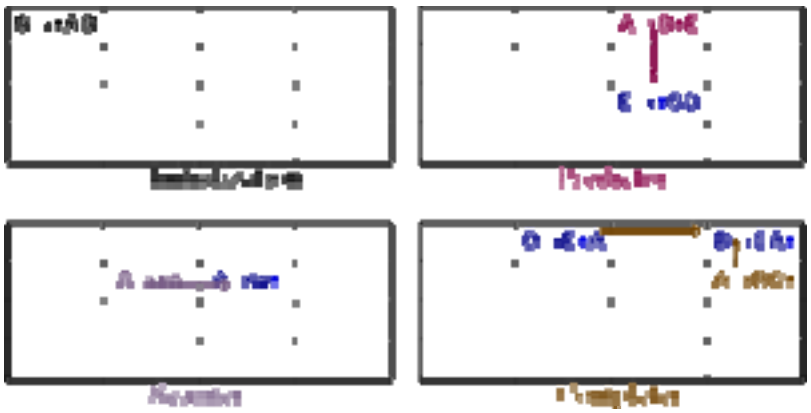
Earley *parser* merupakan jenis dari *chart parser* yang digunakan untuk *parsing* dalam linguistik komputasional. Earley *parser* muncul karena dapat menguraikan semua *context-free language*. *Parser* ini berjalan dalam *cubic time* ($O(n^3)$, di mana *n* adalah panjang dari *string* yang diuraikan) dalam kasus umum, *quadratic time* ($O(n^2)$) untuk *grammar* yang ambigu, dan *linear time* untuk hampir semua LR(k) *grammar*. *Parser* ini bekerja dengan sangat baik ketika aturan dituliskan secara rekursif kiri.

Bergantung pada sumber tertentu, algoritma Earley merupakan baik algoritma *bottom-up* yang menggabungkan beberapa elemen prediksi *top-down*, maupun algoritma prediksi *top-down* yang memiliki pemeriksaan *bottom-up*. Dengan demikian, algoritma Earley terlihat seperti perkawinan antara pendekatan *bottom-up* dan *top-down*. Dari akar *bottom-up*, Earley menjaga *runtime* untuk kasus terburuk pada $O(n^3)$, tetapi karena kekuatan dari elemen prediksi *top-down*, dalam banyak kasus memiliki *runtime* pada $O(n)$. (Sandstrom, 2004, p4).

Berdasarkan algoritma Earley ini, pertama kali dilakukan inisialisasi aturan *top-down* berikut (Loper, 2005, pp30-33):

Untuk setiap aturan *grammar* $S \rightarrow \alpha$, maka tambahkan  ke *queue*[0]. Kemudian, *scan* dari kiri ke kanan dengan menerapkan satu dari 3 aturan ini:

- *Predictor (=top-down rule)*
- *Scanner (=fundamental rule on terminals)*
- *Completer (=fundamental rule on nonterminals)*



Gambar 2.7 Aturan dalam Algoritma Earley

Berikut adalah algoritma sederhana dari algoritma Earley:

```
For each rule S→α in the grammar:
    Add S→•α to chart[0,0]
For i = 0 to N:
    for edge in queue[i]:
        if edge is incomplete and edge.next is a part of speech:
            scanner(edge)
        if edge is incomplete and edge.next is not a POS:
            predictor(edge)
        if edge is complete:
            completer(edge)
```

Gambar 2.8 Algoritma Earley

2.5.6 Analisis Semantik

Salah satu metode analisis semantik pada NLP adalah dengan mencari suatu pola dalam kalimat yang akan mewakili makna dari kalimat tersebut. Metode ini dapat digunakan dalam memproses *input*-an *user* untuk mendapatkan suatu dokumen yang sesuai dengan *input*-an tersebut. Selain itu dapat juga digunakan untuk menganalisis isi dari suatu informasi atau dokumen dan menggunakannya untuk membuat *knowledge base* yang terperinci lengkap dengan indexnya, yang dapat diakses oleh *user* untuk mendapatkan informasi yang tepat.

Untuk bisa mendapatkan hasil yang sesuai, sistem memproses bahasa alami dari *input*-an *user* dan/atau suatu dokumen untuk mendapatkan pola SPO (Subjek-Predikat-Objek) dan menyimpannya. Selain itu, *link* antara pola yang didapatkan dengan sumber informasi juga harus disimpan untuk digunakan dalam tahap selanjutnya, yaitu pencocokan data. Pencarian pola ini bertujuan untuk mendapatkan makna dari suatu bahasa. Tujuannya adalah memperbesar ketepatan hasil pada proses pencocokan, di mana informasi yang dianggap cocok adalah hanya yang memiliki makna serupa.

Pengembangan dari proses ini adalah dengan menggunakan pola yang lebih komprehensif, yaitu perluasan dari pola SPO yang dapat terdiri dari Sifat, Keterangan, Preposisi, dan sebagainya.

Contoh:

Input: *Does the moon always keep the same face towards the Earth*?

Output:

Subjek	:	*moon*
Predikat	:	*keep*
Objek	:	*same face*
Preposisi	:	*towards*
Objek Tak Langsung	:	*Earth*
Sifat	:	-
Keterangan	:	*always*

(Tsourikov et al, 2000, pp14-15; Tsourikov et al, 2001, p8).

2.6 Bahasa Indonesia

2.6.1 Hakikat Bahasa

Bahasa adalah suatu sistem lambang berupa bunyi, bersifat arbitrer, digunakan oleh suatu masyarakat tutur untuk bekerja sama, berkomunikasi, dan mengidentifikasi diri. Sebagai sebuah sistem, maka bahasa terbentuk oleh suatu aturan, kaidah, atau pola-pola tertentu, baik dalam bidang tata bunyi, tata bentuk kata, maupun tata kalimat. Bila aturan, kaidah, atau pola ini dilanggar, maka komunikasi dapat terganggu. (Chaer, 2006, p1).

Menurut Putrayasa (2007, pp19-20), bahasa terdiri dari dua lapisan, yaitu lapisan bentuk dan lapisan makna yang dinyatakan oleh lapisan bentuk tersebut. Bentuk bahasa terdiri atas satuan-satuan yang dapat dibedakan menjadi satuan fonologi dan satuan gramatikal. Satuan fonologi meliputi fonem dan suku. Sedangkan satuan gramatikal meliputi wacana, kalimat, klausa, frase, kata, dan morfem.

2.6.2 Kata

Kata merupakan unsur yang paling penting di dalam bahasa. Tanpa kata, mungkin tidak ada bahasa. Sebab kata itulah yang merupakan perwujudan bahasa. Setiap kata mengandung konsep makna dan mempunyai peranan di dalam pelaksanaan bahasa. Konsep dan peran apa yang dimiliki tergantung dari jenis atau macam kata-kata itu, serta penggunaannya di dalam kalimat.

Dilihat dari konsep makna yang dimiliki dan atau peran yang harus dilakukan, Chaer (2006, pp86-196) membedakan kata-kata menjadi beberapa jenis kata berikut:

a. Kata benda

Kata-kata yang dapat diikuti dengan frase "yang.." atau "yang sangat.." disebut kata benda. Misalnya: jalan (yang bagus), murid (yang rajin), pelayanan (yang sangat memuaskan). Selain itu, yang disebut kata benda turunan atau bentukan dapat dikenali dari bentuknya yang mungkin:

- Berawalan pe-, seperti pemuda, pemenang.
- Berakhiran -an, seperti bendungan, bantuan.
- Berakhiran -nya, seperti besarnya, naiknya.
- Berimbuhan gabungan pe-an, seperti pembangunan, pelebaran.
- Berimbuhan gabungan per-an, seperti pertemuan, persatuan.
- Berimbuhan gabungan ke-an, seperti keadilan, kekayaan.

b. Kata ganti

Kata benda yang menyatakan orang sering kali diganti kedudukannya di dalam pertuturan dengan sejenis kata yang lazim disebut kata ganti. Misal: “Kemarin ayah pergi ke pasar. Dia membeli sebuah cangkul.” Kata “dia” pada kalimat kedua adalah kata ganti. Kata “dia“ menggantikan kedudukan kata “ayah” yang disebutkan pada kalimat pertama.

c. Kata kerja

Kata-kata yang dapat diikuti oleh frase “dengan..”, baik yang menyatakan alat, keadaan, maupun penyerta, disebut kata kerja. Misalnya: pergi (dengan adik), pulang (dengan gembira), menulis (dengan spidol). Dilihat dari strukturnya, ada dua macam kata kerja, yaitu kata kerja dasar dan kata kerja berimbuhan.

Kata kerja dasar adalah kata kerja yang belum diberi imbuhan, seperti pergi, pulang, tulis, tanya. Sementara kata kerja berimbuhan adalah kata kerja yang terbentuk dari kata dasar yang mungkin kata benda, kata kerja, kata sifat, atau jenis kata lain dan imbuhan. Imbuhan yang lazim digunakan dalam pembentukan kata kerja adalah:

- Awalan me-, seperti menulis, melihat.
- Awalan ber-, seperti berdiri, berkuda.
- Awalan di-, seperti ditulis, dibaca.
- Awalan ter-, seperti tertulis, terlihat.

- Awalan per-, seperti perpanjang, percepat.
- Akhiran -kan, seperti tuliskan, damaikan.
- Akhiran -i, seperti diami, tinggali.

d. Kata sifat

Kata-kata yang dapat diikuti dengan kata keterangan "sekali" serta dapat dibentuk menjadi kata ulang berimbuhan gabung "se-nya" disebut kata sifat. Misalnya: indah (indah sekali, seindah-indahnya), besar (besar sekali, sebesar-besarnya), baik (baik sekali, sebaik-baiknya).

e. Kata sapaan

Kata-kata yang digunakan untuk menyapa, menegur, atau menyebut orang kedua, atau orang yang diajak bicara, disebut kata sapaan. Kata-kata sapaan ini tidak mempunyai perbendaharaan kata sendiri, tetapi menggunakan kata-kata dari perbendaharaan kata nama diri dan kata nama perkerabatan. Misalnya: San (bentuk utuh: Hasan), Ti (bentuk utuh: Siti), Nek (bentuk utuh: Nenek).

f. Kata penunjuk

Kata-kata yang digunakan untuk menunjuk benda disebut kata penunjuk. Ada dua macam kata penunjuk, yaitu "ini" dan "itu." Kata penunjuk "ini" digunakan untuk menunjuk benda yang letaknya relatif

dekat dari si pembicara. Sedangkan kata penunjuk "itu" untuk menunjuk benda yang letaknya relatif jauh dari si pembicara.

g. Kata bilangan

Kata-kata yang menyatakan jumlah, nomor, urutan, atau himpunan disebut kata bilangan. Menurut bentuk dan fungsinya, ada dua macam kata bilangan:

- Kata bilangan utama, seperti satu, dua tiga, tiga puluh satu.
- Kata bilangan tingkat, seperti pertama, kedua, kedua puluh satu.

h. Kata penyangkal

Kata-kata yang digunakan untuk menyangkal atau mengingkari terjadinya suatu peristiwa atau adanya suatu hal disebut kata penyangkal. Misalnya: tidak, tak, tiada, bukan, tanpa.

i. Kata depan

Kata-kata yang digunakan di muka kata benda untuk merangkaikan kata benda itu dengan bagian kalimat lain, disebut kata depan. Dilihat dari fungsinya, kata depan dapat dibedakan menjadi kata depan yang menyatakan:

- Tempat berada, yaitu: di, pada, dalam, atas, antara.
- Arah asal, yaitu: dari.
- Arah tujuan, yaitu: ke, kepada, akan, terhadap.
- Pelaku: yaitu: oleh.

- Alat, yaitu: dengan, berkat.
- Perbandingan, yaitu: daripada.
- Hal atau masalah, yaitu: tentang, mengenai.
- Akibat, yaitu: hingga, sampai.
- Tujuan, yaitu: untuk, buat, guna, bagi.

j. Kata penghubung

Kata-kata yang digunakan untuk menghubungkan kata dengan kata, klausa dengan klausa, atau kalimat dengan kalimat, disebut kata penghubung. Dilihat dari fungsinya, ada dua macam kata penghubung, yaitu:

- Kata penghubung setara, yang menggabungkan:
  - Biasa: dan, dengan, serta.
  - Memilih: atau.
  - Mempertentangkan: tetapi, namun, sedangkan, sebaliknya.
  - Membetulkan: melainkan, hanya.
  - Menegaskan: bahkan, malah (malahan), lagipula, apalagi, jangankan.
  - Membatasi: kecuali, hanya.
  - Mengurutkan: lalu, kemudian, selanjutnya.
  - Menyamakan: yaitu, yakni, bahwa, adalah, ialah.
  - Menyimpulkan: jadi, karena itu, oleh sebab itu.

- Kata penghubung bertingkat, yang menyatakan:
  - Sebab: sebab, karena.
  - Syarat: kalau, jikalau, jika, bila, apabila, asal.
  - Tujuan: agar, supaya.
  - Waktu: ketika, sewaktu, sebelum, sesudah, tatkala.
  - Akibat: sampai, hingga, sehingga.
  - Sasaran: untuk, guna.
  - Perbandingan: seperti, sebagai, laksana.
  - Tempat: tempat.

k. Kata keterangan

Kata-kata yang digunakan untuk memberi penjelasan pada kalimat atau bagian kalimat lain, yang sifatnya tidak menerangkan keadaan atau sifat, disebut kata keterangan. Ada dua macam kata keterangan:

- Kata keterangan yang menerangkan keseluruhan kalimat, berfungsi untuk menyatakan:
  - Kepastian: memang, pasti, tertentu.
  - Keraguan atau kesangsian: barangkali, mungkin, kiranya, rasanya, agaknya, rupanya.
  - Harapan: semoga, moga-moga, mudah-mudahan, hendaknya.
  - Frekuensi: seringkali, sekali-sekali, sesekali, sekali-kali, acapkali, jarang.

- Kata keterangan yang menerangkan unsur kalimat, berfungsi untuk menyatakan:
  - Waktu: sudah, telah, sedang, lagi, tengah, akan, belum, masih, baru, pernah, sempat.
  - Sikap batin: ingin, mau, hendak, suka, segan.
  - Perkenan: boleh, wajib, mesti, harus, jangan, dilarang.
  - Frekuensi: jarang, sering, sekali, dua kali.
  - Kualitas: sangat, amat, sekali, lebih paling, kurang, cukup.
  - Kuantitas dan jumlah: banyak, sedikit, kurang, cukup, semua, beberapa, seluruh, sejumlah, sebagian, separuh, kira-kira, sekitar, kurang lebih, para, kaum.
  - Penyangkalan: tidak, tak, tiada, bukan.
  - Pembatasan: hanya, cuma.

l. Kata tanya

Kata-kata yang digunakan sebagai pembantu di dalam kalimat yang menyatakan pertanyaan disebut kata tanya. Misalnya: apa, siapa, mengapa, kenapa, bagaimana, berapa, mana, kapan, bila, bilamana.

m. Kata seru

Kata-kata yang digunakan untuk mengungkapkan perasaan batin, misalnya karena kaget, terharu, kagum, marah, atau sedih, disebut kata seru. Dilihat dari strukturnya, ada dua macam kata seru:

- Kata seru yang berupa kata singkat: wah, cih, hai, o, oh, nah, ha, hah.
- Kata seru yang berupa kata biasa: aduh, celaka, gila, kasihan, bangsat, ya ampun, astaga, masya Allah, alhamdullilah, dan lain-lain.

n. Kata sandang

Kata-kata yang berfungsi menjadi penentu disebut kata sandang. Yaitu: si, sang.

o. Kata partikel

Morfem-morfem yang digunakan untuk menegaskan disebut partikel penegas. Yaitu: kah, tah, lah, pun, per.

2.6.3 Imbuhan

Acapkali sebuah kata dasar atau bentuk dasar perlu diberi imbuhan dulu untuk dapat digunakan di dalam pertuturan. Imbuhan ini dapat mengubah makna, jenis, dan fungsi sebuah kata dasar atau bentuk dasar menjadi kata lain, yang fungsinya berbeda dengan kata dasar atau bentuk dasarnya. Imbuhan mana yang harus digunakan tergantung pada keperluan penggunaannya di dalam pertuturan. Untuk keperluan pertuturan itu malah sering pula sebuah kata dasar atau bentuk dasar yang sudah diberi imbuhan dibubuhi pula dengan imbuhan lain.

Imbuhan yang ada dalam bahasa Indonesia adalah:

a. Akhiran: -kan, -i, -nya, -an.
b. Awalan: ber-, per-, me-, di-, ter-, ke-, se-, pe-.
c. Sisipan: -el, -em, -er.
d. Imbuhan gabung: ber-kan, ber-an, per-kan, per-i, me-kan, me-i, memper-, memper-kan, memper-i, di-kan, di-i, diper-, diper-kan, diper-i, ter-kan, ter-i, ke-an, se-nya, pe-an, per-an.

2.6.4 Kalimat

Menurut Putrayasa (2007, p20), dalam bahasa Indonesia, kalimat ada yang terdiri dari satu kata, dua kata, tiga kata, empat kata, dan seterusnya. Sesungguhnya yang menentukan satuan kalimat bukan banyaknya kata yang menjadi unsurnya, melainkan intonasinya. Dalam wujud lisan, kalimat diucapkan dengan suara naik turun dan keras lembut, disela jeda, dan diakhiri dengan intonasi akhir yang diikuti oleh kesenyapan yang mencegah terjadinya perpaduan asimilasi bunyi ataupun proses fonologis lainnya. Dalam wujud tulisan, kalimat dimulai dengan huruf kapital dan diakhiri dengan tanda titik, tanda tanya, atau tanda seru. Berdasarkan uraian tersebut, dapat disimpulkan bahwa yang dimaksud dengan kalimat adalah satuan gramatikal yang dibatasi oleh adanya jeda panjang yang disertai nada akhir naik atau turun.

Sementara menurut Chaer (2006, p327), kalimat adalah satuan bahasa yang berisi suatu pikiran atau amanat yang lengkap. Lengkap berarti di dalam kalimat itu terdapat unsur atau bagian yang:

- Menjadi pokok pembicaraan, disebut subjek (S). Misalnya: Adik membaca buku.
- Menjadi komentar tentang subjek, disebut predikat (P). Misalnya: Adik membaca buku.
- Merupakan pelengkap dari predikat, disebut objek (O). Misalnya: Adik membaca buku.
- Merupakan penjelasan lebih lanjut terhadap predikat dan subjek, disebut keterangan (K). Misalnya: Adik membaca buku di perpustakaan.

Menurut strukturnya, sebuah kalimat sederhana dalam bahasa Indonesia memiliki pola:

a. Subjek + Predikat

   Contoh: - Ibuku tertawa.

   - Ayahku seorang dokter.

b. Subjek + Predikat + Objek

   Contoh: - Ibu menjahit baju adik.

   - Ayah membaca koran pagi.

c. Subjek + Predikat + Objek + Keterangan

   Contoh: - Ibu menjahit baju adik semalam.

   - Ayah membaca koran di taman.

d. Subjek + Predikat + Objek + Objek

   Contoh: - Ibu membelikan adik baju baru.

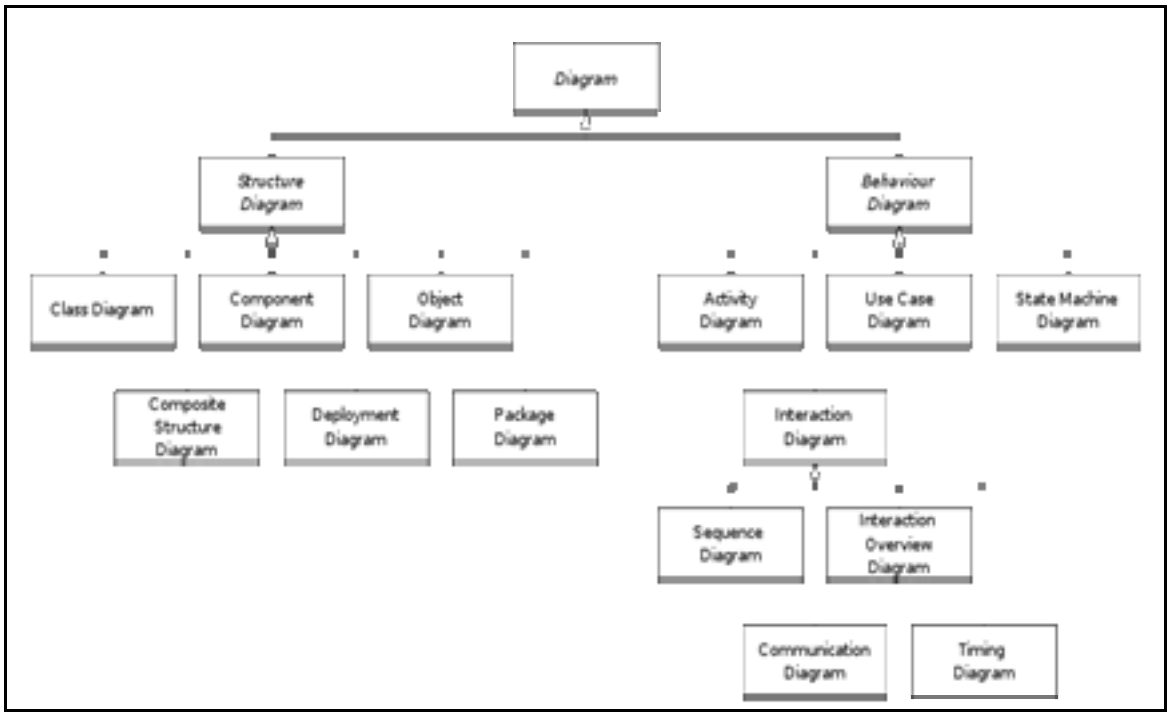
   - Ayah membukakan saya pintu.

2.7 *Unified Modeling Language*

Unified Modeling Language (UML) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem piranti lunak. UML tidak berdasarkan pada bahasa pemrograman tertentu. Standar spesifikasi UML dijadikan standar *defacto* oleh *Object Management Group* (OMG) pada tahun 1997. UML yang berorientasikan objek mempunyai beberapa notasi standar.

Spesifikasi ini menjadi populer dan standar karena sebelum adanya UML, telah ada berbagai macam spesifikasi yang berbeda. Hal ini menyulitkan komunikasi antar pengembang piranti lunak. Untuk itu beberapa pengembang spesifikasi yang sangat berpengaruh berkumpul untuk membuat standar baru. UML dirintis oleh Grady Booch, James Rumbaugh pada tahun 1994 dan kemudian Ivar Jacobson (http://id.wikipedia.org/wiki/UML).

UML mendeskripsikan *Object Oriented Programming* (OOP). OOP merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus dalam *kelas-kelas* atau *objek-objek*. Berbeda dengan logika pemrograman terstruktur. Setiap objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya. UML mendeskripsikan OOP dengan beberapa diagram, yang dapat digambarkan secara hirarki sebagai berikut (http://en.wikipedia.org/wiki/Unified_Modeling_Language):



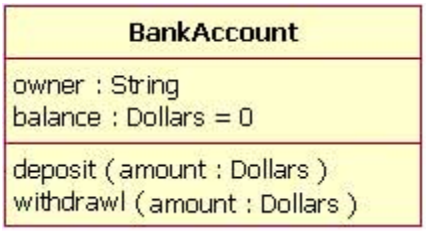
Gambar 2.9 Diagram UML

2.7.1 *Structure Diagram*

Structure diagram menekankan pada apa yang harus dimodelkan dalam sistem. Yang termasuk *structure diagram* adalah:

a. *Class Diagram*

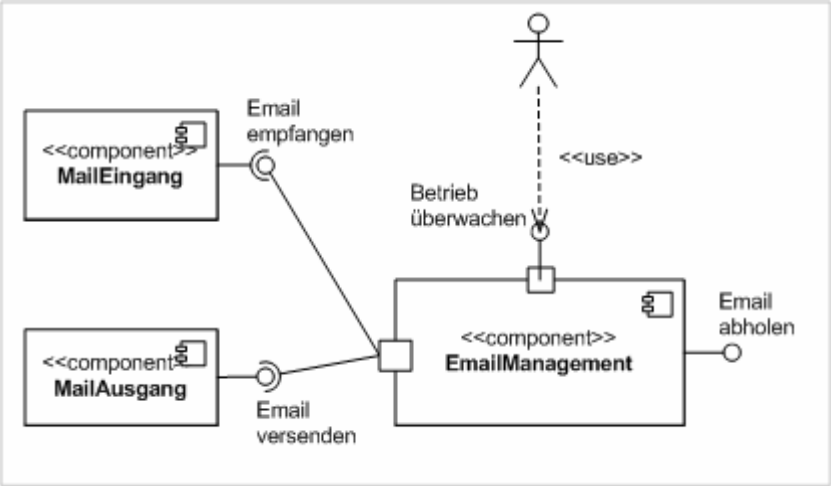
Menggambarkan struktur dari sistem dengan menunjukkan kelas-kelas dari sistem, atributnya, dan hubungan antar kelas.



Gambar 2.10 Contoh *Class Diagram*

b. *Component Diagram*

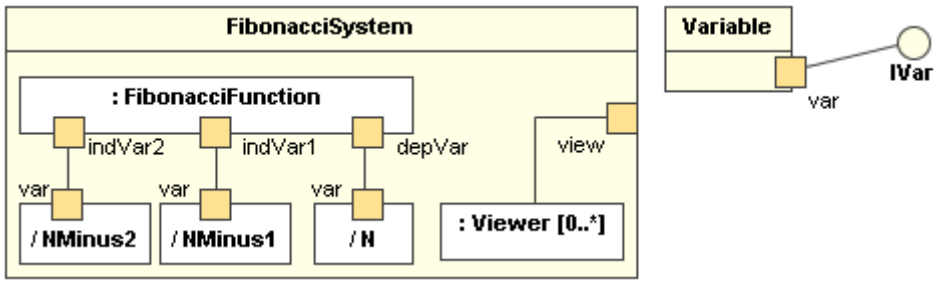
Menggambarkan bagaimana sistem piranti lunak dibagi ke dalam komponen-komponen dan menunjukkan ketergantungan antar komponen.



Gambar 2.11 Contoh *Component Diagram*

c. *Composite Structure Diagram*

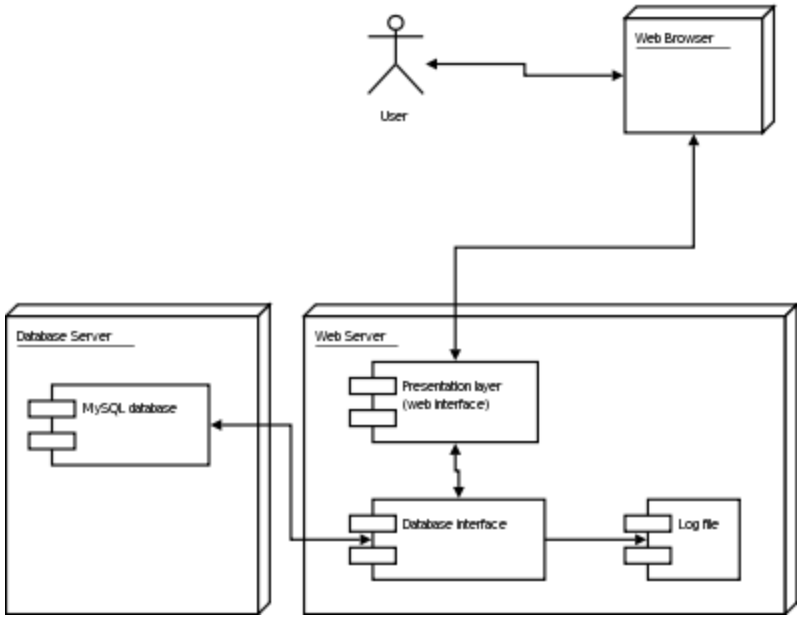
Menggambarkan struktur internal dari kelas dan kolaborasi yang mungkin terjadi dari struktur tersebut.



Gambar 2.12 Contoh *Composite Diagram*

d. *Deployment Diagram*

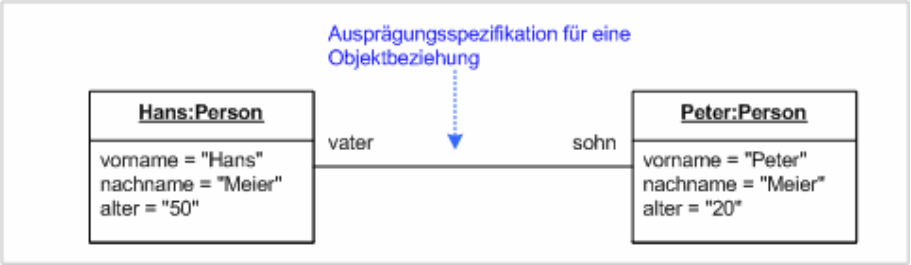
Digunakan untuk membangun model perangkat keras yang digunakan dalam implementasi sistem, komponen yang diatur dalam perangkat keras, dan asosiasi di antara komponen tersebut.



Gambar 2.13 Contoh *Deployment Diagram*

e. *Object Diagram*

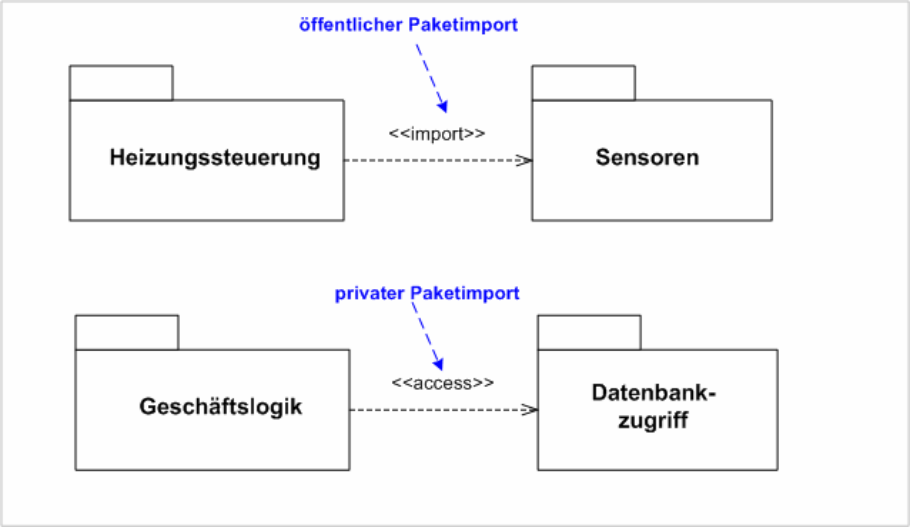
Menunjukkan pandangan lengkap atau sebagian terhadap struktur dari sistem yang telah dibuat modelnya pada waktu tertentu.



Gambar 2.14 Contoh *Object Diagram*

f. *Package Diagram*

Menggambarkan bagaimana sistem dibagi ke dalam kelompok logikal dengan menunjukkan ketergantungan antar kelompok.



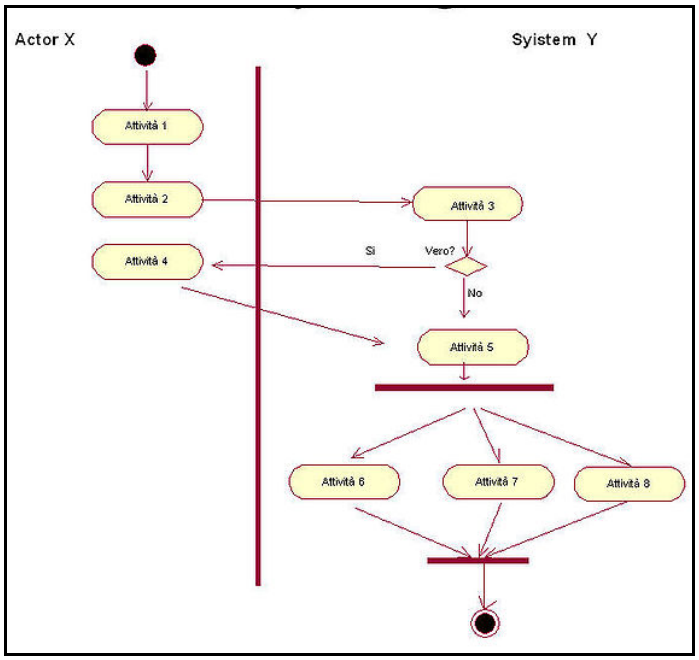
Gambar 2.15 Contoh *Package Diagram*

2.7.2 *Behavior Diagram*

Behavior diagram menekankan pada apa yang harus terjadi ketika membuat pemodelan sistem. Yang termasuk *behavior diagram* adalah:

a. *Activity Diagram*

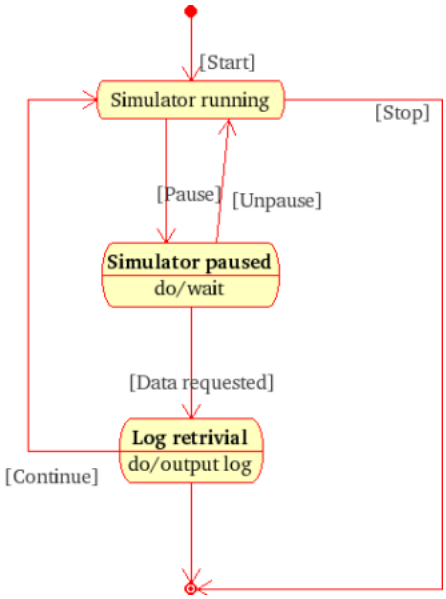
Merepresentasikan aliran bisnis dan operasional langkah demi langkah terhadap komponen dalam sistem. Diagram ini menunjukkan keseluruhan aliran kontrol.



Gambar 2.16 Contoh *Activity Diagram*

b. *State Diagram*

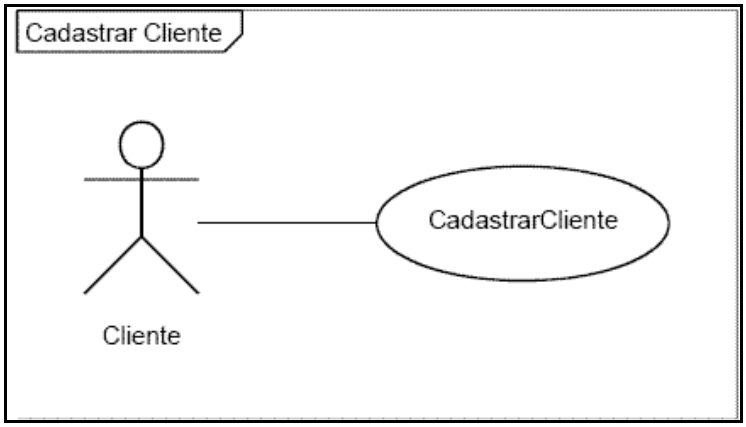
Notasi terstandarisasi untuk menggambarkan banyak sistem, dari program komputer sampai ke proses bisnis.



Gambar 2.17 Contoh *State Diagram*

c. *Use Case Diagram*

Menggambarkan fungsionalitas yang disediakan oleh sistem dari sudut pandang *actor*, tujuan *actor* digambarkan dengan *use case*, dan segala ketergantungan antar *use case*.



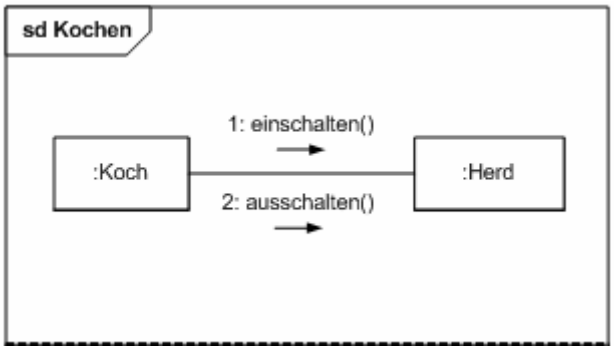
Gambar 2.18 Contoh *Use Case Diagram*

2.7.3 *Interaction Diagram*

Interaction diagram, yang merupakan *subset* dari *behavior diagram*, menekankan pada aliran kontrol dan data di antara hal-hal yang ada ketika membuat pemodelan sistem. Yang termasuk *interaction diagram* adalah:

a. *Communication Diagram*

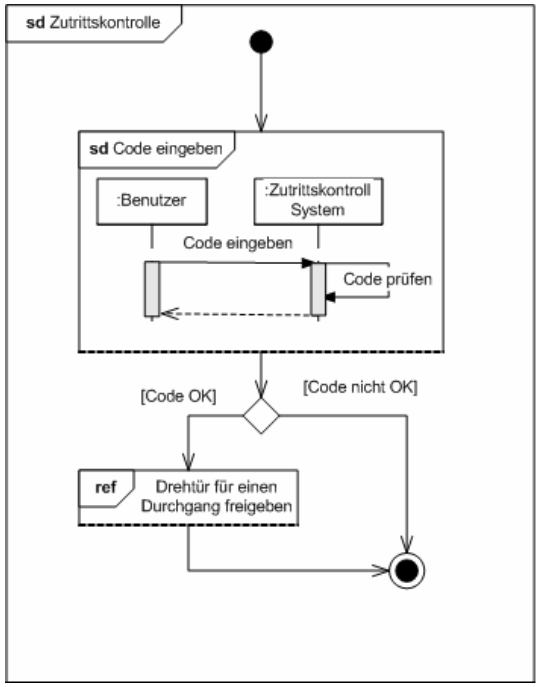
Menunjukkan interaksi antara objek atau bagian dalam hal deretan pesan. Diagram ini merepresentasikan kombinasi informasi yang diambil dari *class, sequence*, dan *use case diagram*, yang menggambarkan baik struktur statis maupun perilaku dinamis dari sistem.



Gambar 2.19 Contoh *Communication Diagram*

b. *Interaction Overview Diagram*

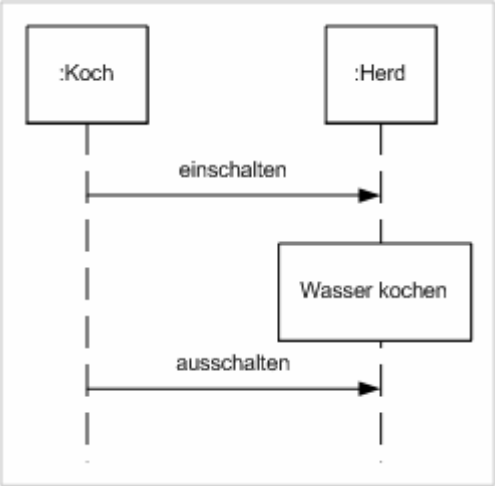
Jenis dari *activity diagram* di mana node-nodenya merepresentasikan *interaction diagram*.



Gambar 2.20 Contoh *Interaction Overview Diagram*

c. *Sequence Diagram*

Menunjukkan bagaimana objek berkomunikasi satu sama lain dalam hal deretan pesan. Diagram ini juga mengindikasikan bahwa umur suatu objek relatif terhadap pesan tersebut.



Gambar 2.21 Contoh *Sequence Diagram*

d. *Timing Diagram*

Jenis spesifik dari *interaction diagram*, di mana fokusnya pada batasan waktu.

2.8 *System Development Life Cycle*

System Development Life Cycle (SDLC) dalam *system engineering* dan *software engineering* berkaitan dengan proses dari mengembangkan sistem, model dan metodologi yang digunakan orang untuk mengembangkan sistem tersebut, biasanya komputer atau sistem informasi.

Dalam *software engineering*, konsep SDLC dikembangkan ke dalam beberapa jenis metodologi pengembangan piranti lunak, *framework* yang digunakan untuk membangun struktur, merencanakan, dan mengontrol proses pengembangan sistem informasi.

SDLC merupakan proses logikal yang digunakan oleh *system analyst* untuk mengembangkan sebuah sistem informasi, termasuk *requirement*, *validation*, pelatihan, dan kepemilikan *user*. Sebuah SDLC seharusnya menghasilkan sistem berkualitas tinggi yang sesuai dengan harapan *customer*, dengan perkiraan waktu dan biaya, bekerja secara efektif dan efisien dalam infrastruktur Teknologi Informasi saat ini dan yang terencana, serta murah untuk dipelihara dan hemat biaya untuk pengembangannya.

Fase Pengembangan Sistem

a. Inisialisasi/Perencanaan

   Untuk menghasilkan pandangan tingkat tinggi terhadap proyek dan menentukan tujuan dari proyek. *Feasibility study* terkadang digunakan untuk menampilkan proyek kepada manajemen tingkat atas dengan maksud mendapatkan pendanaan. Secara tipikal, proyek dievaluasi ke dalam tiga area *feasibility*: ekonomi, operasional, dan teknis. Lebih lanjut, *feasibility* juga digunakan sebagai referensi untuk menjaga proyek tetap pada jalurnya dan untuk mengevaluasi kemajuan tim manajemen sistem informasi. Fase ini disebut juga dengan fase analisis.

b. Pengumpulan dan Analisis Kebutuhan

Tujuan dari analisis sistem adalah untuk menentukan di manakah masalahnya dalam usaha memperbaiki sistem. Langkah ini melibatkan pemecahan sistem ke dalam bagian-bagian yang berbeda dan menggambar diagram-diagram untuk menganalisis situasi. Tujuan proyek analisis, pemecahan fungsi yang dibutuhkan untuk dibuat, dan usaha untuk mengikutsertakan *user* sehingga kebutuhan dapat didefinisikan.

c. Desain

Dalam mendesain sistem, fungsi dan operasi digambarkan secara detail, meliputi rancangan layar, aturan bisnis, diagram proses, dan dokumentasi lainnya. Hasil dari tahap ini akan menggambarkan sistem baru sebagai kumpulan dari modul atau subsistem.

d. *Coding*

Kode programming *modular* dan subsistem akan diselesaikan pada tahap ini. Tahap ini bercampur baur dengan modul individual selanjutnya yang dibutuhkan untuk diuji sebelum integrasi dengan proyek utama. Perencanaan dalam SDLC melibatkan penentuan tujuan, pendefinisian target, penjadwalan, dan perkiraan budget untuk keseluruhan proyek piranti lunak.

e. Pengujian

Kode diuji dalam level yang bervariasi pada pengujian piranti lunak. Unit, sistem, dan pengujian penerimaan *user* dilakukan. Di sini merupakan area abu-abu karena banyak pendapat berbeda yang muncul tergantung tahap pengujian apa yang dilakukan dan seberapa banyak jika iterasi terjadi. Iterasi bukan

merupakan bagian umum dari model *waterfall*, tetapi terkadang terjadi pada tahap ini. Jenis-jenis pengujian:

- *Data set testing*
- *Unit testing*
- *System testing*
- *Integration testing*
- *User acceptance testing*
- *Black box testing*
- *White box testing*
- *Module testing*
- *Regression testing*
- *Automation testing*

f. Operasi dan *Maintenance*

Instalasi sistem mencakup perubahan dan penambahan sebelum menonaktifkan sistem. Memelihara sistem merupakan aspek penting dari SDLC. Seiring dengan perubahan posisi dari personil kunci dalam organisasi, perubahan baru akan diimplementasikan, yang membutuhkan pembaharuan sistem.

(http://en.wikipedia.org/wiki/Systems_Development_Life_Cycle).

2.9 Interaksi Manusia dan Komputer

Interaksi manusia dan komputer (IMK) berkaitan dengan antar muka yang digunakan oleh *user* untuk berkomunikasi dan berinteraksi dengan komputer. IMK merupakan disiplin ilmu yang berhubungan dengan perancangan, evaluasi, dan

implementasi sistem komputer interaktif untuk digunakan oleh manusia, serta studi fenomena-fenomena besar yang berhubungan dengannya.

Menurut Shneiderman (1998, p72), ada delapan aturan emas perancangan antar muka, yaitu:

1. Berusaha untuk konsisten

   Urutan aksi yang konsisten diperlukan pada situasi-situasi yang mirip. Terminologi yang serupa digunakan pada *prompt*, menu, dan layar bantuan. Konsistensi perintah juga dilakukan secara menyeluruh.

2. Memungkinkan penggunaan *shortcut*

   Semakin meningkatnya frekuensi penggunaan, maka meningkat pula keinginan *user* untuk mengurangi jumlah interaksi dan meningkatkan langkah interaksi. Singkatan, *function key*, perintah tersembunyi, dan fasilitas makro sangat berguna untuk *expert user*.

3. Memberikan umpan balik yang informatif

   Untuk setiap langkah operator, diperlukan beberapa umpan balik dari sistem. Untuk aksi yang rutin dan tambahan, respon dapat sederhana. Sementara untuk aksi yang tidak rutin dan utama, respon seharusnya lebih kuat.

4. Merancang dialog yang memberikan penutupan (keadaan akhir)

   Serangkaian aksi diorganisir ke dalam satu grup dengan awal, tengah, dan akhir. Umpan balik yang informatif pada akhir dari aksi grup memberi operator kepuasan penyelesaian, rasa lega, sinyal untuk mengeluarkan perencanaan kemungkinan dan pilihan dari pikiran mereka, serta sebuah indikasi bahwa dapat bersiap untuk sekumpulan aksi selanjutnya.

5. Memberikan pencegahan kesalahan dan penanganan kesalahan yang sederhana

   Sebanyak mungkin, desain sistem sehingga *user* tidak dapat melakukan kesalahan yang fatal. Jika terjadi kesalahan, sistem harus dapat mendeteksinya dan menawarkan mekanisme yang sederhana dan dapat dipahami untuk penanganan kesalahan.

6. Memungkinkan pembalikan aksi yang mudah

   Fitur ini mengurangi ketidaknyamanan, karena *user* tahu bahwa kesalahan dapat diulang. Hal ini memberanikan penjelajahan terhadap pilihan yang tidak dikenal. Kesatuan dari kemampuan untuk pembalikan dapat berupa aksi tunggal, pemasukan data, atau sebuah kumpulan aksi yang lengkap.

7. Mendukung pusat kendali internal (*internal locus of control*)

   Operator yang berpengalaman sangat menginginkan rasa bahwa mereka terlibat di dalam sistem dan sistem dapat merespon terhadap aksi mereka. Desain sistem untuk membuat *user* menjadi inisiator dari aksi daripada hanya sebagai responden.

8. Mengurangi beban ingatan jangka pendek

   Batasan terhadap pemrosesan informasi manusia dalam jangka pendek memerlukan tampilan yang sederhana, menggabugkan tampilan *mutiple page*, frekuensi *window-motion* dikurangi, dan ada waktu yang cukup untuk mempelajari kode, *mnemonic*, dan serangkaian aksi.