GRAHA ILMU

Sistem Pakar Perancangan dan Pembahasan

Metode Chaining, Certainty Faktor, Fuzzy Logik

Linda Marlinda

Sistem Pakar Perancangan dan Pembahasan

Metode Chaining, Certainty Faktor, Fuzzy Logik

Sistem Pakar Perancangan dan Pembahasan

Metode Chaining, Certainty Faktor, Fuzzy Logik

Linda Marlinda

GRAHA ILMU

SISTEM PAKAR PERANCANGAN DAN PEMBAHASAN; Metode Chaining, Certainty Faktor, Fuzzy Logik

oleh Linda Marlinda

Edisi Pertama: Cetakan I ~ 2021

GRAHA ILMU

Ruko Jambusari 7A Yogyakarta 55283
Telp: 0274-889398; 0274-882262; email: info@grahailmu.co.id

ISBN: 978-623-228-832-4

Buku ini tersedia sumber elektronisnya

DATA BUKU:

Format: 17 x 24 cm; Jml. Hal.: xviii + 176; Kertas Isi: HVS 70 gram; Tinta Isi: BW; Kertas Cover: Ivori 260 gram; Tinta Cover: Colour; Finishing: Perfect Binding: Laminasi Doff.

Buku ini aku persembahkan untuk kedua anakku tercinta:
Marysca Shintya Dewi
Shandy Alviano Nugroho

KATA PENGANTAR

Puji syukur ke hadirat Allah SWT, karena atas berkat dan rahmat-Nyalah buku ini dapat diselesaikan dengan baik. Buku ini ditujukan bagi semua mahasiswa dengan program studi sistem informasi, manajemen informatika dan teknik Informatika yang akan menyelesaikan tugas akhir atau skripsi. Sehingga memudahkan bagi para mahasiswa dalam membuat rancangan sistem pakar dengan melihat program yang dibuat sesuai dengan bahasa pemrogramannya yaitu berbasis objek.

Adapun bahasa pemprogramanya yang dapat diimplementasikan dengan menggunakan UML ini adalah pemprograman visual *basic*, visual foxpro dan *web programming*.

Buku ini dipakai sebagai referensi dalam matakuliah sistem pakar di kalangan mahasiswa STIMIK Nusa Mandiri dengan beberapa metode tersebut terdiri dari metode *fordward chaining, backforward chaining,* metode *certainty*, dan metode *Fuzzy Logic*.

Harapan penulis, adanya buku ini dapat memahami analisis dan perancangan sistem pakar dengan mudah bagi pembaca khususnya mahasiswa pemula agar mudah dengan mengimplementasikan rancangan pakar dengan diagram-diagram yang terdapat dalam UML.

Akhirnya penulis mengucapkan banyak terima kasih pada semua pihak yang mendukung dalam menyelesaikan buku ini

Jakarta, Juli 2020

Linda Marlinda, MM, M.Kom

DAFTAR ISI

-oo0oo-

DAFTAR GAMBAR

-oo0oo-

DAFTAR TABEL

-oo0oo-

BAB 1

PENDAHULUAN

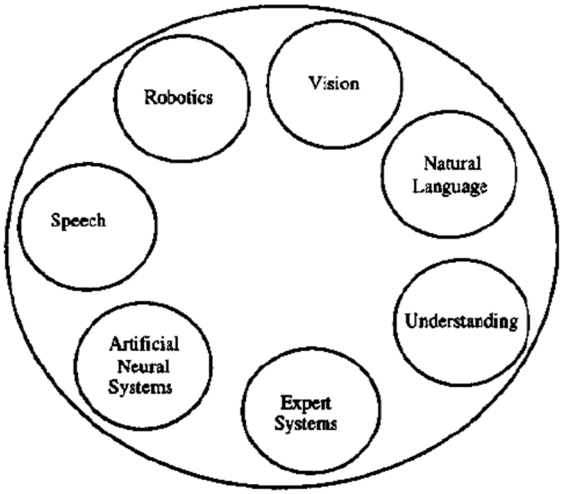
1.1 Pendahuluan

Sistem pakar dalam AI dapat menyelesaikan banyak masalah yang umumnya membutuhkan pakar manusia. Ini didasarkan pada pengetahuan yang diperoleh dari seorang ahli. Kecerdasan Buatan dan Sistem Pakar mampu mengekspresikan dan bernalar tentang beberapa domain pengetahuan. Sistem pakar adalah pendahulu dari sistem kecerdasan buatan dan pembelajaran mesin saat ini.

Sistem pakar atau disebut *expert system* adalah salah satu cabang dari AI atau *Artificial Intelligence* yang menggunakan pengetahuan untuk penyelesaian masalah manusia dengan menggunakan pakar. Seorang pakar adalah orang yang mempunyai keahlian dalam bidang tertentu, yaitu pakar yang mempunyai *knowledge* atau kemampuan khusus yang orang lain tidak mengetahui atau mampu dalam bidang yang dimilikinya. *Knowledge* dalam sistem pakar mungkin saja seorang ahli, atau *knowledge* yang umumnya terdapat dalam buku, majalah dan orang yang mempunyai pengetahuan tentang suatu bidang. Sistem pakar yang baik tentunya dirancang agar dapat menyelesaikan suatu permasalahan tertentu dengan meniru kerja dari ahli. Kecerdasan tiruan (*artificial intelegent*) merupakan bagian dari ilmu komputer yang mempelajari tentang bagaimana membuat mesin (komputer) dapat melakukan pekerjaan seperti dan sebaik yang

dilakukan oleh manusia bahkan bisa lebih baik daripada yang dilakukan manusia.

Kecerdasan tiruan dikembangkan untuk memberikan kemampuan pada komputer agar dapat berpikir, menalar, dan membuat referensi, selain itu juga membuat keputusan berdasarkan fakta-fakta yang ada, maka komputer dapat digunakan sebagai alat bantu dalam mengambil keputusan. Beberapa aplikasi digambarkan sebagai berikut:



Gambar 1.1 *Aplikasi Kecerdasan Tiruan*

1. Sistem Pakar (*Expert System*)
   Komputer digunakan sebagai sarana untuk menyimpan pengetahuan sistem pakar, demikian komputer akan memiliki keahlian untuk menyelesaikan permasalahan dengan meniru keahlian yang dimiliki oleh pakar yang bersangkutan.
2. Robotika dan Sistem Sensor
3. Pengenalan Ucapan (*Speech Recognation*)
   Melalui pengenalan ucapan diharapkan manusia dapat berkomunikasi dengan komputer menggunakan suara.
4. Pengolahan Bahasa Alami (*Natural Language Processing*)
   Dengan pengolahan bahasa alami ini diharapkan pemakai (*user*) dapat berkomunikasi dengan komputer menggunakan bahasa sehari-hari.

5. *Computer Vision*
   Mencoba untuk dapat menginterprestasikan gambar atau objek-objek tampak melalui komputer.

Bagian utama dari kecerdasan tiruan adalah bekal pengetahuan dan mempunyai kemampuan untuk menalar. Basis pengetahuan (*knowledge base*) merupakan suatu informasi yang terorganisasi dan teranalisa agar lebih mudah dimengerti dan bisa diterapkan pada pemecahan masalah, basis pengetahuan berisi fakta-fakta, teori, pemikiran dan hubungan antara satu dengan yang lainnya.

Kemampuan menalar (*inference engine*) merupakan kemampuan menarik kesimpulan berdasarkan pengetahuan. Seorang pakar mempunyai keahlian dalam bidang tertentu, yaitu pakar yang memiliki kemampuan khusus.

Teknik yang digunakan dalam *artificial intellegent* adalah:

a. *Search* (Pencarian)
   Menyediakan cara penyelesaian masalah untuk kasus di mana bila tidak ada lagi pendekatan langsung dapat digunakan dan dipindahkan pada kerangka kerja secara langsung.
b. *Use of Knowledge* (Penggunaan Pengetahuan)
   Menyediakan cara penyelesaian masalah yang lebih kompleks dengan mengekploitasi struktur dari objek yang terkait dengan masalah tersebut.
c. *Abstraction*
   Menyediakan cara untuk memilah atau memisahkan keterangan dan variasi dari sekian banyak untuk mempercepat penyelesaian masalah.

1.2 Definisi Sistem Pakar

Sistem pakar adalah sistem pengambilan keputusan berbasis komputer yang interaktif dan andal yang menggunakan fakta dan heuristik untuk memecahkan masalah pengambilan keputusan yang komplek. Itu dianggap pada tingkat tertinggi kecerdasan dan keahlian manusia. Tujuan

dari sistem pakar adalah untuk memecahkan masalah paling komplek dalam domain tertentu.

Sistem pakar merupakan suatu program yang mengandung pengetahuan pada suatu bidang spesifik didasari oleh para pakar atau ahli. Sistem pakar pertama kali dikembangkan oleh periset kecerdasan buatan pada dasawarsa 1960-an dan 1970-an serta diterapkan secara komersial selama 1980-an. Bentuk umum sistem pakar adalah suatu program yang dibuat berdasarkan suatu set aturan untuk menganalisis informasi (biasanya diberikan oleh pengguna suatu sistem), dan analisis matematis dari masalah tersebut. Tergantung dari desainnya, sistem pakar juga mampu merekomendasikan suatu rangkaian tindakan pengguna untuk menerapkan koreksi. Sistem ini memanfaatkan kapabilitas penalaran untuk mencapai suatu kesimpulan.

Sistem pakar telah memainkan peran besar di banyak industri termasuk dalam layanan keuangan, telekomunikasi, perawatan kesehatan, layanan pelanggan, transportasi, video game, manufaktur, penerbangan dan komunikasi tertulis. Dua sistem pakar awal muncul di ruang perawatan kesehatan untuk diagnosis medis: Dendral yang membantu ahli kimia mengidentifikasi molekul organik, dan MYCIN, yang membantu mengidentifikasi bakteri seperi bakteremia dan meningitis, serta merekomendasikan antibiotik dan dosis.

Sistem pakar adalah suatu program komputer cerdas yang menggunakan *knowledge* (pengetahuan) dan prosedur inferensi untuk menyelesaikan masalah yang cukup sulit sehingga membutuhkan seorang ahli untuk menyelesaikannya (Feigenbaum, 2005).

Sejarah Sistem Pakar

Konsep sistem pakar pertama kali dikembangkan pada tahun 1970-an oleh Edward Feigenbaum, profesor dan pendiri Laboratorium Sistem Pengetahuan di Universitas Stanford. Feigenbaum menjelaskan bahwa dunia sedang bergerak dari pemrosesan data ke ”pemrosesan pengetahuan” sebuah transisi yang dimungkinkan oleh teknologi dan arsitektur komputer.

Sistem pakar dalam AI dapat menyelesaikan banyak masalah yang umumnya membutuhkan pakar manusia. Ini didasarkan pada pengetauan yang diperoleh dari seorang ahli (pakar sebagai asisten yang berpengetahuan misalnya dokter, teknisi) terhadap orang awam yang bukan pakar atau ahli agar dapat mendapatkan pemecahan suatu masalah untuk mendapat solusi.

Sistem pakar adalah pendahulu dari kecerdasan buatan, pembelajaran mendalam dan pembelajaran mesin saat ini.

1972, Newell dan Simon memperkenalkan Teori Logika secara konseptual yang kemudian berkembang pesat dan menjadi acuan pengembangan sistem berbasis AI lainnya. 1978, Buchanan dan Feigenbaum mengembangkan bahasa pemrograman DENDRAL, bahasa pemrograman ini dibuat untuk Badan Antariksa AS (NASA) dan digunakan untuk penelitian kimia di planet Mars. Pada perkembangan selanjutnya studi pada AI difokuskan pada pemecahan masalah sehari-hari atau memberi pertimbangan yang masuk akal (*commonsense reasoning*) terhadap permasalahan yang dihadapi manusia. Hal ini mencakup pertimbangan mengenai suatu objek dan hubungannya dengan objek yang lain

1.3 Mengapa Sistem Pakar Diperlukan

a. Sistem pakar harus mampu membenarkan kesimpulan dalam cara yang sama seorang pakar manusia bisa menjelaskan mengapa kesimpulan tertentu tercapai. Dengan demikian, penjelasan menyediakan fasilitas pemeriksaan dimengerti dari alasan bagi manusia.
b. Memiliki fasilitas penjelasan dalam tahap pengembangan sistem pakar dalam mengkonfirmasi bahwa pengetahuan yang telah diperoleh benar yang digunakan oleh sistem. Fasilitas penjelasan yang baik memungkinkan ahli pakar pengetahuan untuk memverifikasi kebenaran pengetahuan.
c. Memiliki sistem penalaran yaitu aliran pelaksanaan yang tidak memiliki urutan dalam sistem pakar sehingga hanya dapat membaca kode baris demi baris dan memahami sistem beroperasi yang

digunakan, maksudnya urutan aturan yang masuk ke dalam sistem tidak harus berurutan sebelum dieksekusi.

d. Memiliki fasilitas penjelasan sederhana. Fasilitas penjelasan sederhana dalam sistem berbasis aturan sehingga dibuatkan daftar semua fakta yang dibuat.
e. Memiliki metarule, yaitu pengetahuan tentang aturan-aturan (awalan meta berarti "atas" atau "di luar"). Beberapa program seperti Meta-D, telah secara eksplisit peraturan dibuat untuk menyimpulkan hasil yang baru (Buchanan, 78).

1.4 Kelebihan Sistem Pakar

a. *Increased Availability*, Pengetahuan seorang pakar yang sudah diadaptasi ke bentuk *software* dapat diperbanyak dan disebarluaskan dalam jumlah yang tidak terbatas
b. *Reduced cost*, Mengurangi biaya di mana pembuatan sistem pakar bertujuan untuk mengurangi biaya yang harus dikeluarkan untuk membayar pakar atau ahli
c. *Reduced danger* atau mengurangi bahaya, sistem pakar dapat digunakan dalam lingkungan yang mungkin berbahaya untuk manusia
d. *Permanence*, bersifat Tetap di mana *software* sistem pakar dapat digunakan kapan saja tanpa ada batas waktu dan tersimpan di dalam komputer.
e. *Multiple expertise*, Beberapa keahlian. Pengetahuan dari beberapa ahli dapat dibuat tersedia untuk bekerja secara simultan dan terus menerus pada masalah di setiap saat, siang atau malam hari. Tingkat keahlian gabungan dari beberapa ahli dapat melebihi dari ahli manusia tunggal (Harmon, 85)
f. *Increased reliability*, meningkatnya reliabilitas di mana para ahli sistem meningkatkan rasa percaya diri bahwa keputusan yang benar telah dibuat dengan memberikan pendapat kedua pakar manusia.
g. *Explanation*, keputusan yang dibuat oleh sistem pakar bersifat sangat Jelas dan tepat.
h. Sistem pakar secara eksplisit dapat menjelaskan secara rinci alasan yang menuju kesimpulan.

i. *Fast Response*, Cepat respon, Cepat atau *real-time* respon mungkin diperlukan untuk beberapa aplikasi. Tergantung pada perangkat lunak dan *hardware* yang digunakan, sistem pakar dapat merespon lebih cepat dan lebih tersedia daripada seorang pakar manusia, beberapa situasi darurat mungkin memerlukan tanggapan lebih cepat daripada manusia dan sistem pakar *real-time* merupakan pilihan yang baik (Hugh 88; Ennis 86).
j. *Steady, unemotional, complete*, tenang, tanpa emosi, dan respon lengkap setiap saat. Hal ini mungkin sangat penting secara *real-tune* dan situasi darurat, ketika seorang pakar manusia tidak dapat beroperasi pada efisiensi puncak karena stres atau kelelahan.
k. *Intelligent tutor*, *Intelligent* aktor di mana sistem pakar dapat bertindak sebagai guru atau pakar dengan menjalankan program sebagai penalaran penjelasan sistem sistem.
l. *Intelligent database*, *Intelligent database* di mana sistem pakar dapat digunakan untuk mengakses database dengan cara yang cerdas (Kerschberg 86; Schur 88).
m. Proses pengembangan sistem pakar memiliki manfaat tidak langsung juga sejak pengetahuan para ahli manusia harus dimasukkan ke dalam bentuk eksplisit untuk masuk ke dalam komputer.
n. Memperluas jangkauan dari seorang pakar.
o. Dapat mengetahui sampai sejauh mana tingkat kepakaran seorang ahli atau pakar.
p. Merupakan arsip yang terpercaya dari sebuah keahlian.
r. Dapat menyederhanakan suatu pekerjaan.
s. Merupakan efisiensi waktu kerja.
t. Meningkatkan produktifitas karena meningkatnya kualitas hasil pekerjaan.

1.5 Kekurangan Sistem Pakar

a. Daya kerja dan produktivitas manusia menjadi berkurang karena semuanya dilakukan secara otomatis oleh sistem
b. Pengembangan perangkat lunak sistem pakar lebih sulit dibandingkan perangkat lunak konvensional.

c. Pengetahuan seorang pakar yang sudah diadaptasi kebentuk *software* dapat diperbanyak dan disebarluaskan dalam jumlah yang tidak terbatas.

1.6 Pembagian Sistem Pakar Berdasarkan Kelas

a. Konfigurasi, merakit komponen sistem dengan cara yang benar
b. Diagnosa, menarik kesimpulan terhadap masalah yang dihadapi berdasarkan bukti-bukti yang diobservasi
c. Instruksi, metode pengajaran yang cerdas sehingga berfungsi sebagai pakar
d. Interprestasi, menjelaskan data-data yang diobservasi

11.7 Karakteristik Sistem Pakar

a. Tingkat Keahlian Tertinggi (*The Highest Level of Expertise*). Sistem Pakar dalamn AI menawarkan tingkat keahlian tertinggi. Ini memberikan efisiensi, akurasi, dan pemecahan masalah yang imajinatif
b. Reaksi Tepat Waktu (*Right on Time Reaction*): Sistem Pakar dalam Kecerdasan buatan berinteraksi dalam jangka waktu yang sangat wajar dengan dengan pengguna. Total waktu harus kurang dari waktu yang dibutuhkan oleh seorang ahli untuk mendapatkan solusi paling akurat untuk masalah yang sama.
c. Keandalan yang Baik (*Good Reliability*): Sistem pakar dalam AI harus dapat diandalkan, dan tidak boleh membuat kesalahan apapun.
d. Fleksibel: Sangat penting untuk tetap fleksibel karena dimiliki oleh Sistem Pakar.
e. Mekanisme Efektif (*Effective Mechanism*): Sistem Pakar dalam *Artificial Intelligence* harus memiliki mekanisme yang efisien untuk mengelola kompilasi pengetahuan yang ada di dalamnya
f. Mampu menangani keputusan dan masalah yang menantang (*Capable of handling chalenging decision and problems*): Sistem Pakar mampu menangani masalah keputusan yang menantang dan memberikan solusi.

g. Hipotesis, Sistem Pakar dapat melakukan serangkaian hipotesa yang dapat dibandingkan dan tidak bertentangan dengan hipotesa dari seorang pakar dalam masalah yang nyata

1.8 Unsur Penting Pengembangan Sistem Pakar

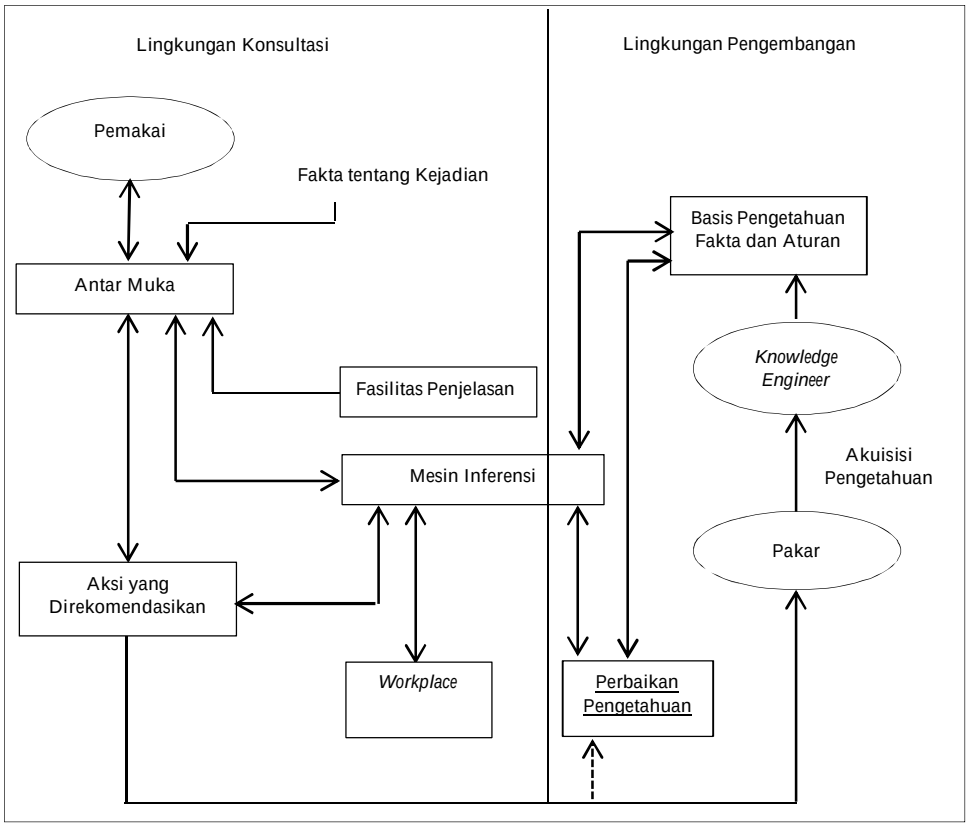
Unsur penting dalam membangun sistem pakar

a. Perancangan sistem pakar berdasarkan pakar atau ahli, dalam merancang sistem pakar diperlukan metode-metode yang akan dipakai seperti metode *forward*, *backforward chaining* dan certain faktor ataupun menggunakan *fuzzy logic* berdasarkan masalah yang didasari oleh pakar atau para ahli.
b. Sistem pendukung baik *hardware* maupun *software*nya, sistem *software* pendukung sistem pakai berbasis object bisa menggunakan phpmyadmin, pemprograman vicual basic, Java, SQL, Visual Foxpro dan lain sebagainya.
c. Pemakai, dalam merancang sistem pakar, permasalahan yang dibuat, harus terfokus ke *object* yang akan dibuat pakar dan ditujukan untuk pemakai siapa.

1.9 Komponen Sistem Pakar

Menurut Turban (2005) sistem pakar disusun oleh dua bagian utama, yaitu lingkungan pengembangan (*development invironment*). Lingkungan pengetahuan pakar ke dalam lingkungan sistem pakar, sedangkan lingkungan konsultasi digunakan oleh pengguna yang bukan pakar untuk memperoleh pengetahuan pakar.

Komponen-komponen sistem pakar dalam kedua bagian tersebut dapat dilihat dalam gambar 1.1 berikut ini:



(Sumber: Turban, 1995)

Gambar 1.2 *Arsitektur Sistem Pakar*

a. Antarmuka pengguna, adalah komponen yang mengambil queri pengguna dalam bentuk yang dapat dibaca dan meneruskannya ke dalam mesin inferensi. Setelah itu, ini akan akan menampilkan hasilnya kepada pengguna. Komponen ini memberikan fasilitas komunikasi antara pemakai dan sistem pakar dengan memberikan berbagai fasilitas informasi dan berbagai keterangan yang bertujuan untuk membantu mengarahkan alur penelusuran masalah sampai ditemukan solusi. Syarat utama membangun antarmuka pemakai adalah kemudahan dalam menjalankan sistem, tampilan yang interaktif, komunikatif dan mudah bagi pemakai sehingga disebut basis pengetahuan.

b. Basis pengetahuan, adalah gudang fakta yang menyimpan semua pengetahuan tentang domain masalah dan merupakan wadah pengetahuan yang diperoleh dari berbagai ahli di bidang tertentu. Ada beberapa cara merepresentasikan data menjadi basis pengetahuan, seperti dalam bentuk atribut, aturan-aturan, jaringan semantik, frame dan logika.

Menurut Firebaugh (1989), terdapat empat teknik yang telah dibuktikan efektif untuk representasi pengetahuan, yaitu:

1) Jaringan semantik
Dalam jaringan semantik, pengetahuan diorganisasikan dengan menggunakan jaringan yang disusun oleh dua komponen dasar, yaitu *node* dan *arc. Node* menyatakan objek, konsep, atau situasi yang ditujukan oleh kotak atau lingkaran. Sedangkan *arc* menyatakan hubungan antar *node* yang ditunjukan oleh tanda panah yang menghubungkan *node-node* dalam jaringan.
2) *Frame*
Frame digunakan untuk merepresentasikan pengetahuan dalam konteks di mana urutan kejadian dan objek muncul. Sebuah *frame* digambarkan dengan menggunakan jaringan dari *node-node* dan hubungan-hubungan. Level teratas dari *frame* menyatakan atribut-atribut sedangkan level terendah memiliki terminal dan slot yang harus diisi oleh data.
3) *Script*
Script menyerupai *frame* dengan informasi tambahan tentang urutan kejadian yang diharapkan serta tujuan dan rencana dari aktor yang terlibat.
4) Kaidah Produksi
Kaidah produksi menjadi acuan yang sangat sering digunakan. Kaidah produksi dituliskan dalam bentuk pernyataan ***IF-Then* (Jika-Maka)**. Pernyaataan ini menghubungkan bagian *premis* (***IF***) dan bagian kesimpulan (***Then***) yang dituliskan dalam bentuk:

IF [premis] THEN [konklusi]

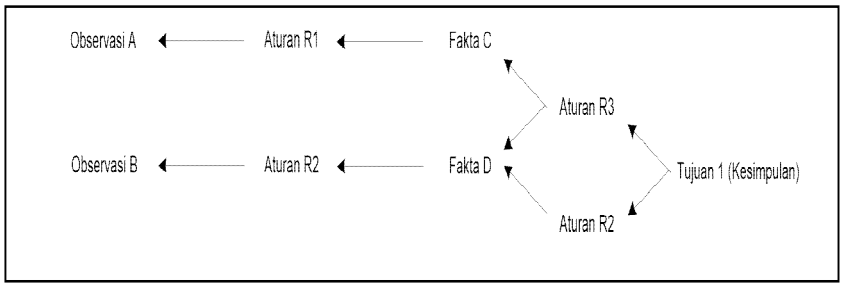
Apabila bagian *premis* dipenuhi maka bagian *konklusi* akan bernilai benar. Sebuah kaidah terdiri dari klausa-klausa. Sebuah klausa mirip dengan sebuah kalimat dengan subjek, kata kerja, dan objek yang menyatakan suatu fakta.

c. Mesin Inferensi, Adalah bagian sistem pakar yang melakukan penalaran dengan menggunakan isi daftar aturan berdasarkan urutan dan pola tertentu. Dalam mesin inferensi pemilihan fakta dan aturan dapat diterapkan pada saat menjawab pertanyaan pengguna sehingga masalah yang dihadapi menemukan solusi yang diinginkan.

Secara umum ada dua teknik utama yang digunakan dalam mekanisme inferensi untuk pengujian aturan, yaitu penalaran maju (*forward reasoning*) dan penalaran mundur (*reverse reasoning*).

1). Pelacakan ke Belakang (*Backward Chaining*)

Menggunakan pendekatan *goal-driven,* dimulai dari tujuan apa yang akan terjadi (hipotesis) dan kemudian mencari bukti yang mendukung (atau berlawanan) dengan harapan kita. Sering hal ini memerlukan perumusan dan pengujian hipotesis sementara.



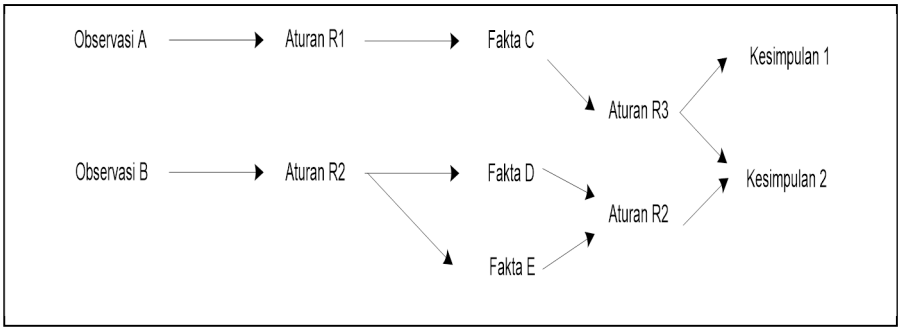
(Sumber: Arhami, 2005)

Gambar 1.3 *Backward Chaining*

2) Pelacakan ke Depan (*Forward Chaining*)

Forward chaining adalah *data-driven* karena inferensi dimulai dengan informasi yang tersedia dan baru konklusi diperoleh. *Forward chaining* merupakan metode inferensi yang melakukan penalaran dari suatu masalah kepada solusinya. Jika *klausa premis*

sesuai dengan situasi (bernilai *TRUE*), maka proses akan menyatakan *konklusi*.

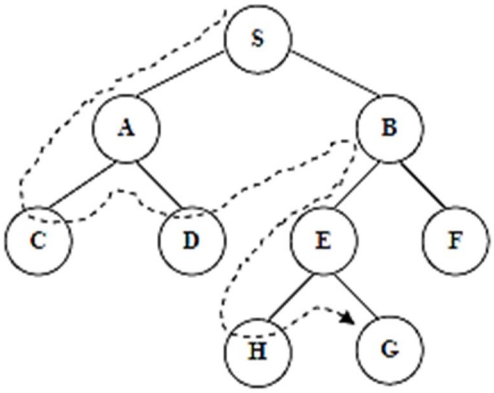


(Sumber: Arhami, 2005)

Gambar 1.4 *Forward Chaining*

Teknik Inferensi dipengaruhi oleh tiga macam penelusuran, yaitu:

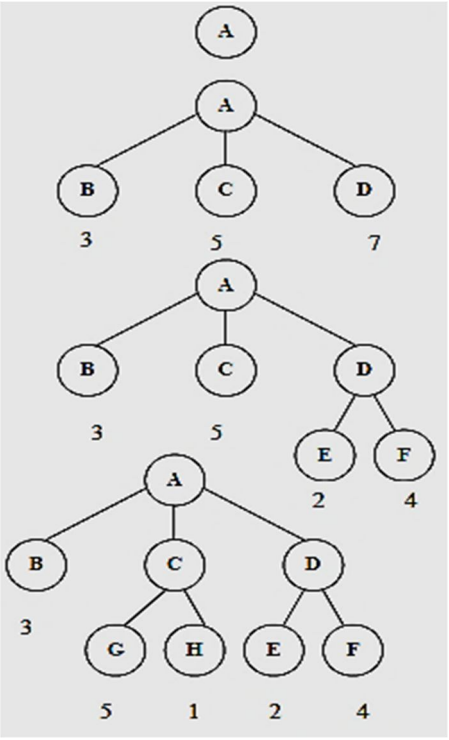
1. Pencarian Mendalam Pertama (*Depth-First Search*)
   Melakukan penelusuran kaidah secara mendalam dari simpul akar bergerak menurun ketingkat dalam yang berurutan. Proses pencarian akan dilaksanakan pada semua anaknya sebelum dilakukan pencarian ke *node* yang selevel. Pencarian dimulai dari *node* akar ke level yang lebih tinggi. Proses ini diulangi terus hingga ditemukannya solusi.



(Sumber: Arhami, 2005)

Gambar 1.5 *Depth First Search*

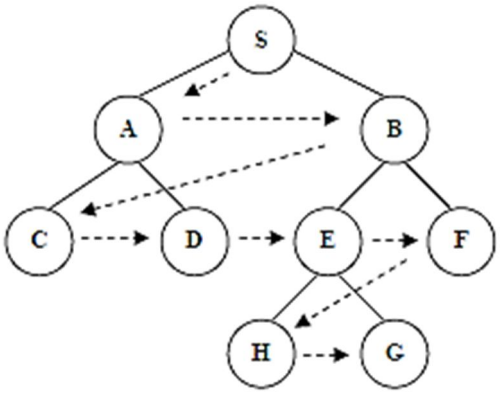
2. Pencarian Melebar Pertama (*Breadth-First Search*)
   Bergerak dari simpul akar, simpul yang ada pada setiap tingkat uji sebelum pindah ketingkat selanjutnya. Semua *node* pada level *n* akan dikunjungi terlebih dahulu sebelum mengunjungi *node-node* pada level *n*+1. Pencarian dimulai dari *node* akar terus ke level ke-1 dari kiri ke kanan, kemudian berpindah ke level berikutnya demikian pula dari kiri ke kanan sampai ditemukannya solusi.



(Sumber: Arhami, 2005)

Gambar 1.6 *Breadth First Search*

3. Pencarian dengan *Best-First Search*
   Bekerja berdasarkan kombinasi dari metode *depth first search* dan *breadth first search* dengan mengambil kelebihan dari kedua metode tersebut. Pada *best first search*, pencarian diperbolehkan mengunjungi *node* di lebih rendah, jika ternyata *node* di level lebih tinggi memiliki nilai heuristik lebih buruk.



(Sumber: Arhami, 2005)

Gambar 1.7 *Best First Search*

1.10 Pemodelan UML (*Unified Modeling Language*)

Pengantar ke Pemodelan Objek

Teknik analisis berorientasi objek merupakan alat terbaik yang dapat digunakan untuk sebuah proyek yang akan mengimplementasikan sistem yang menggunakan teknologi objek untuk membangun, mengelola, dan merakit objek-objek itu menjadi aplikasi komputer yang berguna. Pendekatan berorientasi objek dipusatkan pada sebuah teknik yang sering disebut *object modeling* / pemodelan objek. Sedangkan pendekatan pemodelan objek selama analisis dan desain sistem disebut *object-oriented* analysis / OOA / analisis berorientasi objek.

UML adalah merupakan suatu singkatan dari kata yang berasal dari bahasa orang luar negeri yaitu (*Unified Modeling Language*) yang merupakan suatu sistem arsitektur *software* yang bekerja dalam OOAD (*Object-Oriented Analysis / Design*) dengan satu bahasa yang konsisten yang berguna untuk menentukan, visualisasi, mengkontruksi, dan men-dokumentasikan *artifact* (sepotong informasi yang digunakan atau dihasilkan dalam suatu proses rekayasa *software*, dapat berupa model, deskripsi, atau *software*) yang terdapat dalam sistem *software*. UML merupakan bahasa pemodelan yang paling sukses dari tiga metode OO

yang telah ada sebelumnya, yaitu *Booch*, OMT (*Object Modeling Technique*), dan OOSE (*Object-Oriented Software Engineering*).

Tujuan UML diantaranya adalah:

1. Memberikan model yang siap pakai sebagai bahasa pemodelan visual yang ekspresif untuk mengembangkan serta saling tukar menukar model dengan mudah dan dapat dimengerti secara umum.
2. Untuk mendapatkan bahasa pemodelan yang bebas dari berbagai bahasa pemrograman dan proses rekasaya perangkat lunak.
3. Untuk menyatukan praktik-praktik terbaik yang terdapat dalam pemodelan.

1.11 Konsep Dasar UML

Konsep dasar UML terdiri dari *structural classification, dynamic behavior* dan *model management.* UML mendefinisikan diagram-diagram sebagai berikut:

1. Usecase Diagram
   Usecase diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah apa yang diperbuat sistem, bukan bagaimana sebuah *Usecase* mempresentasikan sebuah interaksi tertentu, misalnya login ke sistem, meng-create daftar belanja dan sebagainya. *Usecase Diagram* dapat sangat membantu apabila kita sedang menyusun *requirement* sebuah sistem mengkomunikasikan rancangan dengan klien dan merancang *test case* untuk semua *feature* yang ada pada sistem.
2. *Class Diagram*
   Class adalah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah obyek dan merupakan inti dari pengembangan dan desain berorientasikan obyek. *Class* merupakan keadaan suatu sistem sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metode / fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class, paskage* dan obyek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi dan lain-lain.

Class memiliki tiga area pokok:

a. Nama (dan *stereotype*)
b. Atribut
c. Metode

Atribut dan metode dapat memiliki salah satu sifat berikut:

a. *Private,* tidak dapat dipanggil dari luar *class* yang bersangkutan.
b. *Protected,* hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
c. *Public,* dapat dipanggil oleh siapa saja.

Class dapat merupakan implementasi dari sebuah *interface,* yaitu *class abstrak* yang hanya memiliki metode. *Interface* tidak dapat langsung diinstansikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class.* Dengan demikian *interface* mendukung resolusi metode pada saat *runtime.*

Hubungan antar *class*:

a. Asosiasi, yaitu hubungan statis *class* umumnya menggambarkan *class* yang memiliki berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *Navigability* menunjukkan arah *query* antar *class.*
b. Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas….").
c. Pewarisan, yaitu hubungan hirarki antar *class. Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metode *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewaris adalah generalisasi.
d. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram* yang akan dijelaskan kemudian.

3. *Statechart Diagram*

   Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu ke *state* lainnya) suatu obyek pada sistem sebagai akibat dari stimuli yang diterima. Pada umumnya *statechart diagram* menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu s*tatechart diagram*). Dalam UML, s*tatechart diagram* berbentuk segi empat dengan sudut membulat dan memiliki nama sesuai dengan kondisinya saat itu. Transisi antar state umumnya memiliki kondisi guard yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. *Action* yang dilakukan sebagai akibat dari event tertentu dituliskan dengan diawali garis miring. Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.

4. *Activity Diagram*

   Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang bagaimana masing-masing alir data berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses *parallel* yang mungkin terjadi pada beberapa eksekusi. *Activity diagram* merupakan *state* diagram khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *Activity diagram* tidak menggambarkan *behavior internal* sebuah sistem dan interaksi antar sub sistem secara nyata, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum. Sebuah aktivitas dapat direalisasikan oleh satu *usecase* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *usecase* menggambarkan bagaimana *actor* menggunakan sistem untuk melakukan aktivitas. Sama seperti *state*, standar UML menggunakan segi empat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan *behavior* pada kondisi tertentu, digambarkan dengan simbol belah ketupat. Untuk mengilustrasikan proses-proses parallel (*frok and join*) digunakan titik sinkronisasi yang dapat berupa titik, garis *horizontal* atau garis *vertical*. *Activity diagram* dapat dibagi menjadi

beberapa obyek *swimlane* untuk menggambarkan obyek mana yang bertanggung jawab untuk aktivitas tertentu.

5. *Sequence Diagram*

   Sequence diagram menggambarkan interaksi antar obyek di dalam dan di sekitar sistem (termasuk pengguna, *display* dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri antar dimensi vertikal (waktu) dan dimensi *horizontal* (obyek-obyek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan *scenario* atau rangkaian langkah-langkah yang dilakukan sebagai respon dari sebuah event untuk menghasilkan *output* tertentu. Diawali dari apa yang meng-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan. Masing-masing obyek, termasuk aktor memiliki *lifeline vertical*. *Message* digambarkan sebagai garis berpanah dari satu obyek ke obyek lainnya. Pada *face* desain berikutnya, *message* akan dipetakan menjadi operasi /metode dari *class*. *Activision bar* menunjukan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message.* Untuk obyek-obyek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk obyek *boundary, controller,* dan *persitent entity.*

6. *Collaboration Diagram*

   Collaboration diagram juga menggambarkan interaksi antar obyek seperti *Sequence diagram,* tetapi lebih menekankan pada peran masing-masing obyek dan bukan pada waktu penyampaian *message.* Setiap *message* memiliki *sequence number,* di mana *message* dari level tertinggi memiliki nomor 1 (satu). *Message* dari level yang sama memiliki *prefix* yang sama pula.

7. *Component Diagram*

   Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) diantaranya. Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binery code,* baik *libery* maupun *executable,* baik yang muncul pada *compile time, link time,* maupun *run time.* Umumnya komponen terbentuk dari beberapa *class* atau *package,* tapi

bisa juga dari komponen-komponen yang lebih kecil. Komponen bisa juga berupa *interface,* yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen yang lain.

8. *Deployment Diagram*
   Deployment atau *physical diagram* menggambarkan detail bagaimana komponen ditempatkan dalam prastruktur sistem, di mana komponen akan terletak (pada mesin server atau piranti keras apa), bagaimana kemampuan jaringan ada lokasi tersebut, spesifikasi server dan hal-hal lain yang bersifat fisikal. Sebuah *node* adalah *server*, *workstation* atau piranti keras lain yang digunakan untuk menempatkan komponen dalam lingkungan yang sebenarnya.
9. *Package Diagram*
   Merupakan sebuah bentuk pengelompokan yang memungkinkan untuk mengambil sebuah bentuk UML dan mengelompokkan elemen-elemennya dalam tingkatan untuk yang lebih tinggi. Kegunaan *package* umumnya adalah mengelompokkan *class*.

Langkah-langkah penggunaan *Unified Modeling Language* (UML) secara umum:

1. Langkah pertama adalah membuat daftar *business process* dari level tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.
2. Selanjutnya *usecase* untuk *business process* dipetakan untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem, *usecase diagram* diperhalus dan dilengkapi dengan *requirement, constraints* dan catatan-catatan lain.
3. Fungsi *deployment diagram* secara kasar untuk mendefinisikan arsitektur fisik sistem.
4. Pendefinisian *requirement* lain (non fungsional, *security* dan sebagainya) yang juga harus disediakan oleh sistem.
5. Berdasarkan *usecase diagram,* mulailah membuat *activity diagram.*
6. Diperlukan adanya definisi obyek-obyek level atas (*package* atau *domain*) kemudian pembuatan *sequence* dan *collaboration diagram* untuk

tiap alir pekerjaan, jika sebuah *use case* memiliki kemungkinan alir normal dan *error,* perlu dibuat satu diagram untuk masing-masing alir.

7. Selanjutnya diperlukan adanya rancangan *user interface model* yang menyediakan antar muka bagi pengguna untuk menjalankan *scenario usecase*.
8. Berdasarkan model-model yang sudah ada, dapat dibuat *class diagram.* Setiap *package* atau *domain* dipecah menjadi *hirarki class* lengkap dengan atribut dan metodenya. Akan lebih baik jika untuk setiap *class* dibuat unit tes untuk menguji fungsionalitas *class* dan interaksi dengan *class* yang lain.
9. Setelah *class diagram* dibuat, kita dapat melihat kemungkinan pengelompokan *class* menjadi komponen-komponen. Karena itu perlu dibuat *component diagram* pada tahap ini. Juga diperlukan adanya definisi tes intergrasi untuk setiap komponen guna meyakinkan bahwa ia berinteraksi dengan baik.
10. Perhalus *deployment diagram* yang sudah dibuat. Detailkan kemampuan dan *requirement* piranti lunak, sistem operasi, jaringan dan lain sebagainya. Petakan komponen ke dalam node.
11. Setelah tahap-tahap di atas barulah dapat dimulai membangun sistem. Ada dua pendekatan yang dapat digunakan:
    a. Pendekatan *usecase* dengan mengangkat setiap *usecase* kepada tim pengembang tertentu untuk mengembangkan unit *code* yang lengkap dengan tes.
    b. Pendekatan komponen, yaitu mengangkat setiap komponen kepada tim pengembang tertentu.
12. Apabila tahap-tahap di atas telah terpenuhi maka diperlukan adanya uji modul dan uji integrasi serta perbaikan model beserta kodenya. Model harus sesuai dengan kode yang aktual.

1.12 Cara Kerja UML adalah dengan Mendefinisikan Notasi dan Sebuah Meta-model

Notasi adalah model-model yang direpresentasikan dalam bentuk grafis, ini adalah *syntax* untuk bahasa permodelan. Untuk contohnya, *notasi class*

diagram mendefinisikan bagaimana item-item dan konsep-konsep seperti *class, association*, dan *multiplicity* direpresentasikan. Tentu saja, ini semua mengarah kepada pertanyaan apakah sebenarnya yang dimaksud dengan sebuah *association* atau *multiplicity* atau bahkan sebuah *class*. Para pengguna umumnya menyarankan beberapa definisi-definisi informal, tetapi banyak orang menginginkan informasi yang lebih daripada itu semua. Ide dari bahasa spesifikasi dan *design* yang tepat adalah sangat relevan dalam bidang sebuah *formal method*. Di dalam sebuah teknik, *design*, dan *specifications* telah direpresentasikan dengan menggunakan turunan dari *predicate* kalkulus. Definisi tersebut telah dimatematikakan dengan tepat dan tidak ada duanya. Walau bagaimanapun nilai dari definisi ini bukanlah merupakan sesuatu yang universal. Walaupun kita dapat membuktikan bahwa program tersebut dapat membuktikan sebuah spesifikasi matematika yang benar, tidak mungkin ada sebuah cara untuk membuktikan kalau spesifikasi matematika itu dapat memenuhi syarat yang dibutuhkan sebuah sistem.

Design adalah tentang menemukan masalah-masalah kunci yang dihadapi pada *development*. *Formal methods* sering kali membuat putus asa dengan cara banyak memberikan detil-detil yang tidak penting. Juga *formal methods* sangat sulit untuk dipelajari dan dimanipulasi, juga lebih sulit dihadapi daripada bahasa pemrograman. Dan kita juga tidak dapat menjalankannya. Kebanyakan *methods object-oriented* mempunyai sedikit kekakuan, notasi mereka lebih berdasarkan intuisi daripada definisi yang formal. Dalam keseluruhannya, ini semua tampak tidak menimbulkan kerusakan. *Methods* ini mungkin informal, beberapa orang menemukan kalau semua ini masih berguna dan kegunaannya itulah yang diperhitungkan.

1.13 Kategori Diagram UML

1. *Structural Diagram*:
   Menggunakan struktural diagram untuk menampilkan blok bangunan dari sistem

2. *Behavioral Diagram*:
   Menggunakan behavioral diagram untuk menampilkan bagaimana sistem merespon permintaan atau apa saja seiring waktu.
3. *Interaction Diagram*: merupakan tipe dari *behavioral diagram*.
   Menggunakan *interaction diagram* untuk melukiskan perubahan dari pesan-pesan dalam suatu kolaborasi (kumpulan dari *object-object* yang sama) sehingga tujuan bisa tercapai.

   a. *Structural diagram* (*Class diagram*): Digunakan untuk menampilkan entiti dunia nyata, elemen dari analisa dan desain, atau implementasi *class* dan relasinya.
   b. *Structural diagram* (*Object diagram*): Digunakan untuk menampilkan suatu contoh spesifik atau ilustrasi dari suatu objek serta *link* nya. Sering digunakan untuk mengindikasikan kondisi dari suatu even, seperti percobaan atau operasi pemanggilan.
   c. *Structural diagram* (*Composite structure diagram*): Digunakan untuk menampilkan bagaimana sesuatu itu dibuat.
   d. *Structural diagram* (*Deployment diagram*): Digunakan untuk menampilkan arsitektur *run-time* dari suatu sistem, kerangka *hardware*, ruang lingkup *software*, dan sebagainya.
   e. *Structural diagram* (*Component diagram*): Digunakan untuk menampilkan organisasi dan hubungan antar sistem.
   f. *Structural diagram* (*Package diagram*): Digunakan untuk mengorganisir elemen model dan menampilkan ketergantungan antara mereka.
   g. *Behavioral diagram* (*Activity diagram*): Digunakan untuk menampilkan arus data dari kebiasaan antar *object*.
   h. *Behavioral diagram* (*Use case diagram*): Digunakan untuk menampilkan layanan yang bisa diminta oleh *actor* dari sistem.
   i. *Behavioral diagram* (*State machine diagram / Protocol state machine diagram*): Digunakan untuk menampilkan urutan proses dari suatu *object* dan kondisinya saat ini.
   j. *Interaction diagram* (*Overview diagram*): Digunakan untuk menampilkan banyak skenario interaksi (urutan dari kebiasaan) bagi

suatu kolaborasi (kumpulan elemen yang sama dan saling bekerja agar tercapai tujuan yang diinginkan).

k. *Interaction diagram* (*Sequence diagram*): Digunakan untuk fokus pada perubahan pesan antara grup dari suatu *object* dan urutan pesan tersebut.
l. *Interaction diagram* (*Communication diagram*): Digunakan untuk fokus pada perubahan pesan antara grup dari suatu *object* dan relasi dari *object-object* tersebut.
m. *Interaction diagram* (*Timing diagram*): Digunakan untuk menampilkan perubahan dan hubungan terhadap waktu nyata atau terhadap proses sistem.
n. *Static diagram*: Menampilkan fitur statis dari sistem. Kategori ini hampir sama dengan *structural diagram*.
o. *Dynamic diagram*: Menampilkan bagaimana proses perubahan yang terjadi dalam sistem sepanjang waktu. Kategori ini mencakup UML *state-machine diagram* dan *timing diagram*.
p. *Functional diagram*: Menampilkan detail dari proses dan algoritma. Kategori ini mencakup *use case, interaction*, dan *activity diagram*.

UML penting sekali untuk para analis programmer untuk mempermudah melakukan *forward* maupun *reserve engineering*. UML menggunakan meta-model sehingga pembacaan alur sebuah aplikasi dapat dipermudah.

-oo0oo-

Sistem Pakar Perancangan dan Pembahasan

Metode Chaining, Certainty Faktor, Fuzzy Logik

Suatu sistem perangkat lunak yang dapat menyami atau meniru kemampuan seorang pakar dalam memecahkan masalah di bidang-bidang spesialisasi seperti sains, perekayasaan, matematika, kedokteran, pendidikan dan sebagainya yang tidak dapat diselesaikan orang awam. Sistem pakar ini mewakili para pakar seperti dokter, mekanik, psikolog dan lain-lain. Di dalam buku ini dijelaskan apa itu sistem pakar, defini, kelebihan, kekurangan, karakteristis dan struktur sistem pakar), representasi pengetahuan (jaringan semantik, frame, script), metode inferensi (Tree, Forward dan Backward Chaining), penalaran dengan ketidakpastian (Uncertainty), dan Fuzzy Logik. Buku ini memberikan gambaran lengkap mengenai sistem pakar beserta aplikasinya misalnya tabel, pohon pakar, metode yang digunakan dan teknik-teknik representasi pengetahuan serta inferensi dipaparkan dengan jelas. Didalam buku ini juga diberikan contoh penelitian sistem pakar yang mudah dimengerti. Buku ini sangat cocok untuk mahasiswa yang mengambil bidang teknik infomatika ataupun sistem informasi di lingkungan STMIK Nusa MAndiri sehingga dapat pembangun rancangan aplikasi sistem pakar yang diinginkan.

Linda Marlinda, Staff Pengajar di STMIK Nusa Mandiri yang memiliki sertifikasi dosen (Serdos) dan karir dibidang Jabatan fungsional akademik (JFA), dengan pangkat LEKTOR KEPALA (LK) Kum 600,50 dan Golongan IV-A, saat ini saya terdaftar sebagai mahasiswa Pasca Sarjana (S3) Ilmu Komputer di Universitas Dian Nuswantoro Semarang. Memiliki pengalaman mengajar selama 25 tahun di bidang Sistem Pakar dan Sistem Basis Data. Dalam melakukan kegiatan Tri Dharma perguruan tinggi menghasilkan sekitar 50 karya tulis dalam bentuk jurnal dan prosiding (nasional dan International) (2009-2020). Mendapatkan penghargaan sebagai the best paper jurnal penelitian di forum seminar nasional (SESSINDO 2013), menjadi dosen berprestasi dilingkungan STMIK Nusa Mandiri (2012), 10 dosen dengan peringkat scopus tertinggi di STMIK Nusa Mandiri (2020, 2021). Saya berharap dapat memberikan kontribusi ilmu dalam buku sistem pakar ini untuk dapat digunakan oleh mahasiswa dan dosen umumnya .

ISBN: 978-623-228-832-4

Buku ini diterbitkan atas kerjasama dengan
STMIK NUSA MANDIRI