BAB 2

LANDASAN TEORI

2.1 Definisi Citra

Secara harfiah, citra *(image)* adalah gambar pada bidang dwimatra (dua dimensi). Ditinjau dari sudut pandang matematis, citra merupakan fungsi menerus *(continue)* dari intensitas cahaya pada bidang dwimatra. Sumber cahaya menerangi objek, dan objek memantulkan kembali sebagian dari berkas cahaya tersebut. Pantulan cahaya ini ditangkap oleh alat-alat optik, misalnya mata pada manusia, kamera, pemindai *(scanner)*, sehingga bayangan objek yang disebut citra tersebut terekam.

Citra sebagai keluaran dari suatu sistem perekaman data dapat bersifat:

1. Optik berupa foto,
2. Analog berupa sinyal video seperti gambar pada montor televisi,
3. Digital yang dapat langsung disimpan pada pita magnetik.

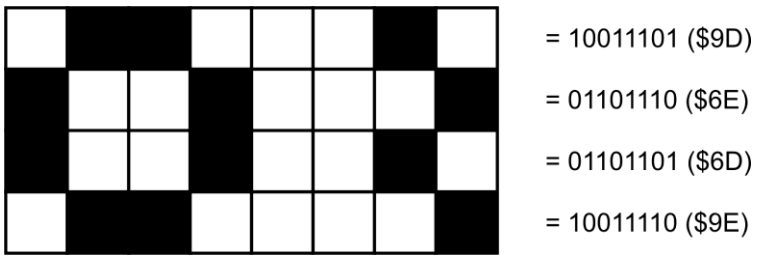
Citra yang dimaksud disini adalah "citra diam" *(still image)*. Citra diam adalah citra tunggal yang tidak bergerak. Untuk selanjutnya, citra diam disebut citra saja. Sedangkan "citra bergerak" *(moving image)* adalah rangkaian citra diam yang ditampilkan secara beruntun *(sequential)* sehingga memberi kesan pada mata kita sebagai gambar yang bergerak. Setiap citra didalam rangkaian itu disebut *frame*. Gambar-gambar yang tampak pada film layar lebar atau televisi pada dasarnya terdiri atas ratusan sampai ribuan *frame*[7].

2.2 Jenis Citra

Nilai suatu pixel memiliki nilai dalam rentang tertentu, dari nilai minimum sampai nilai maksimum. Jangkauan yang digunakan berbeda-beda tergantung dari jenis warnanya. Namun secara umum jangkauannya adalah 0-255. Citra dengan penggambaran seperti ini digolongkan ke dalam citra integer. Berikut adalah jenis-jenis citra berdasarkan nilai pixelnya.

2.2.1 Citra Biner

Pada citra biner, setiap titik bernilai 0 atau 1, masing-masing merepresentasikan warna tertentu. Warna hitam bernilai 0 dan warna putih bernilai 1. Pada standar citra untuk ditampilkan di layar komputer, nilai biner ini berhubungan dengan ada tidaknya cahaya yang ditembakkan oleh *electron gun* yang terdapat di dalam monitor komputer. Angka 0 menyatakan tidak ada cahaya, dengan demikian warna yang direpresentasikan adalah hitam. Untuk angka 1 terdapat cahaya, sehingga warna yang direpresentasikan adalah putih. Setiap titik pada citra hanya membutuhkan 1 bit, sehingga setiap *byte* dapat menampung informasi 8 titik[7].

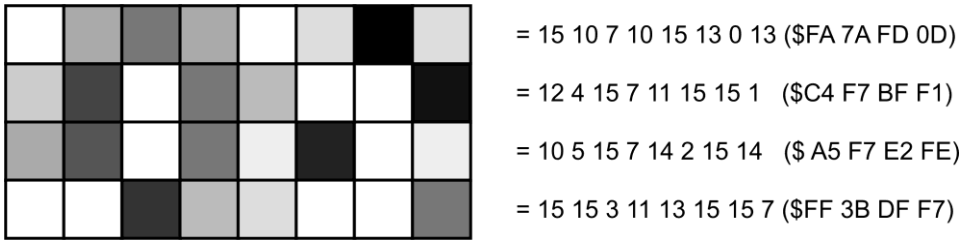


Gambar 2.1 Citra biner dan representasinya dalam data digital

2.2.2 Citra Greyscale

Citra skala keabuan memberi kemungkinan warna yang lebih banyak daripada citra biner, karena ada nilai-nilai lain diantara nilai minimum (biasanya 0) dan nilai maksimumnya. Banyaknya kemungkinan nilai minimum dan nilai maksimumnya bergantung pada jumlah bit yang digunakan. Contohnya untuk skala keabuan 4 bit, maka jumlah kemungkinan nilainya adalah 24 = 16, dan nilai maksimumnya adalah 24 – 1 = 15. Sedangkan untuk skala keabuan 8 bit, maka jumlah kemungkinan nilainya adalah 28 = 256, dan nilai maksimumnya adalah 28 – 1 = 255.

Format citra ini disebut skala keabuan karena pada umumnya warna yang dipakai adalah antara hitam sebagai warna minimal dan warna putih sebagai warna maksimalnya. Namun pada prakteknya, warna yang dipakai tidak terbatas pada warna abu-abu. Sebagai contoh, jika warna minimalnya adalah putih dan warna maksimalnya adalah merah, maka semakin besar nilainya, semakin besar pula intensitas warna merahnya[7].



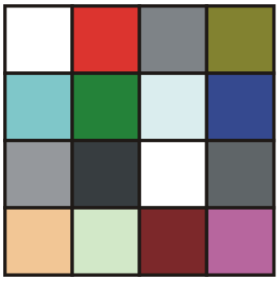
Gambar 2.2 Citra grayscale 4-bit dan representasinya dalam data digital

2.2.3 Citra Warna

Pada citra warna, setiap titik mempunyai warna spesifik yang merupakan kombinasi dari tiga warna dasar, yaitu merah, hijau, dan biru. Format citra ini sering disebut sebagai citra RGB (red-green-blue). Setiap warna dasar mempunyai

intensitas sendiri dengan nilai maksimum 255 (8 bit). Misalnya warna kuning merupakan kombinasi warna merah dan hijau sehingga nilai RGB-nya adalah 255 255 0, sedangkan warna ungu muda nilai RBG-nya adalah 150 0 150. Dengan demikian, setiap titik pada citra warna membutuhkan data 3 byte atau 24 bit.

Jumlah kombinasi warna yang mungkin untuk format citra ini adalah 224 atau lebih dari 16 juta warna, dengan demikian bisa dianggap mencakup semua warna yang ada. Inilah sebabnya format ini dinamakan *true color*[7].



= 255 255 255 - 255 0 0 - 128 128 128 - 128 128 0

= 0 255 255 - 0 128 0 - 204 255 255 - 0 0 255

= 150 150 150 - 51 51 51 - 255 255 255 - 95 95 95

= 255 204 153 - 204 255 204 - 128 0 0 - 255 0 255

Gambar 2.3 Citra warna dan representasinya dalam data digital

2.3 Pre-Processing

Pre-processing perlu dilakukan untuk menyesuaikan apa yang dibutuhkan untuk proses selanjutnya. Yang pertama sistem akan merubah citra berwarna menjadi citra *greyscale* dengan metode *luminosity*.

2.3.1 Metode Luminosity

Metode *luminosity* adalah metode yang lebih canggih dari metode *average*. Metode ini tidak hanya menghitung nilai rata-rata (*average*), tetapi membentuk rata-rata tertimbang sesuai dengan persepsi mata manusia. Mata manusia lebih sensitif terhadap warna hijau dibandingkan dengan warna lain, karena itu warna

hijau diberikan nilai tertimbang paling banyak. Rumus untuk *luminosity* adalah 0,21 R + 0,72 G + 0,07 B[8].

Proses tersebut dilakukan pada setiap piksel pada citra, dengan cara ini maka setiap piksel memiliki satu jenis warna dengan intensitas yang berbeda-beda.

2.4 OCR (Optical Character Recognition)

Optical character recognition (OCR) merupakan aplikasi dari teknologi pengenalan teks, yaitu suatu teknologi yang mampu mengenali teks pada citra digital dan mengalihkannya pada dokumen digital. Dalam perkembangannya, aplikasi OCR sering kali digunakan pada berbagai jenis dokumen, dimana beberapa dokumen memiliki ukuran font yang berbeda satu sama lain. Hal ini menyebabkan aplikasi OCR yang ada menjadi kurang maksimal dalam mengenli teks.

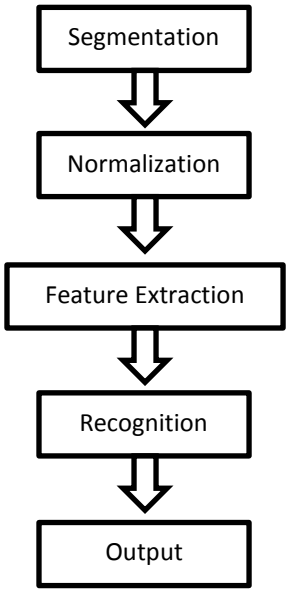
OCR adalah proses yang menerjemahkan gambar karakter (*image character*) menjadi bentuk teks dengan cara mencocokkan pola karakter per-baris dengan pola yang telah tersimpan dalam *database* aplikasi. Hasil dari proses OCR adalah berupa teks sesuai yang tampak pada gambar *output scanner* dimana tingkat keakuratan penerjemahan karakter tergantung dari tingkat kejelasan gambar[9].

Tingkat keberhasilan dari perangkat lunak aplikasi pengenalan teks sangat bergantung dari sejumlah faktor berikut:

1. Kualitas gambar teks yang ada pada dokumen yang dibaca serta tingkat kompleksitasnya (ukuran, format teks, warna, latar belakang).
2. Kualitas alat optik yang dipakai (scanner).

3. Kualitas perangkat lunak aplikasi pengenalan teks itu sendiri[10].

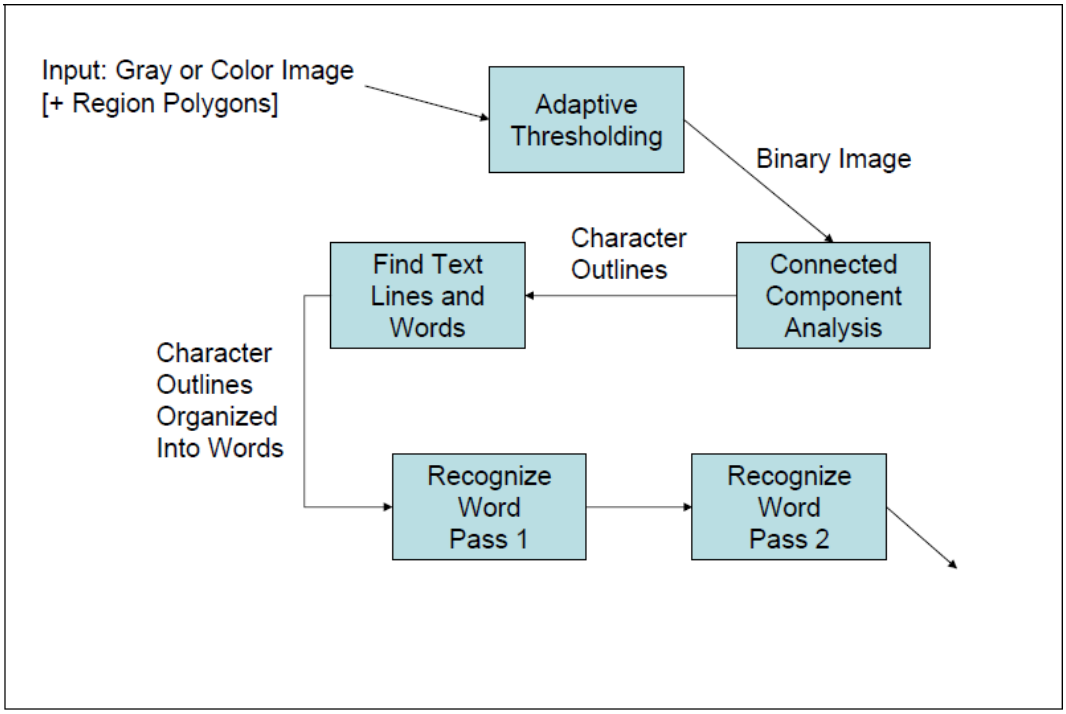
Pada proses pengolahan gambar dengan OCR terdapat beberapa langkah yaitu, *segmentation, normalization, feature extraction, recognition,* dan hasil. Langkah *segmentation* bertujuan untuk memisahkan objek pada gambar dengan latar belakang agar mudah untuk dianalisis. Langkah *normalization* bertujuan untuk mengubah ukuran gambar dan melakukan proses morfologi untuk menghapus piksel *forground* yang terpilih dari gambar biner. Langkah *feature extraction* berfungsi sebagai *image analysist* untuk mengidentifikasi sifat yang melekat pada tiap karakter. Hasil dari langkah *feature extraction* akan digunakan dalam langkah *rocognition* sehingga menghasilkan *output*[11].



Gambar 2.4 Proses Pengenalan Karakter

Tesseract adalah mesin OCR open-source yang dikembangkan oleh HP pada tahun 1984 – 1994. Pada saat ini, untuk pertama kalinya, detail arsitektur dan algoritma dipublikasikan. Tesseract dimulai sebagai sebuah proyek penelitian seorang PhD di HP Labs, Bristol, dan mendapatkan momentum sebagai perangkat lunak dan add-on perangkat keras untuk lini *HP scanner flatbed*. Proyek ini termotivasi oleh fakta bahwa mesin OCR komersial pada saat itu berada di tahap awal, gagal total dan hanya mementingkan kualitas cetak terbaik.

Setelah proyek bersama antara HP Labs Bristol, dan Divisi scanner HP di Colorado, Tesseract memimpin secara signifikan dalam akurasi yang lebih baik dibandingkan dengan mesin OCR komersial, tetapi Tesseract tetap tidak menjadi suatu produk. Tahap selanjutnya dari perkembangannya adalah kembali ke HP Labs Bristol untuk investigasi OCR sebagai alat kompresi. Pengembangan lebih berkonsentrasi pada peningkatan *rejection efficiency* dari pada tingkat akurasi. Pada akhir proyek ini, pada akhir tahun 1994, pembangunan berhenti sepenuhnya. Mesin OCR dikirim ke UNLV untuk melakukan tes akurasi pada tahun 1995, di mana mesin tersebut membuktikan bahwa kualitas Tesseract lebih baik dibandingkan mesin OCR komersial. Pada akhir tahun 2005, HP merilis *open source* Tesseract[12].



Gambar 2.5 Arsitektur Tesseract

1. Adaptive Thresholding

   Proses ini digunakan untuk merubah citra warna atau greyscale yang menjadi citra masukan menjadi citra biner. Tujuannya adalah untuk memisahkan *background* dan *foreground*.

2. Connected Component Analysis

   Proses ini digunakan untuk melakukan proses segmentasi teks yang ada pada citra menjadi karakter yang nantinya akan dikenali.

3. Find Lines and Words

   Proses ini digunakan untuk mencari baris teks pada suatu citra.

4. Recognize Word Pass 1

   Proses ini digunakan untuk proses pengenalan tahap awal, yaitu membandingkan karakter yang didapat dari citra dengan data yang sudah ada.

5. Recognize Word Pass 2

   Proses terakhir adalah proses pengenalan kembali dari keseluruhan halaman pada citra untuk memperbaiki hasil pengenalan tahap pertama.

2.5 Microsoft Translator

Microsoft Translator adalah layanan web yang berfungsi untuk menerjemahkan teks dari bahasa satu ke bahasa lain. Microsoft Translator menyediakan API yang dapat digunakan oleh para *develover* untuk mengambil keuntungan dari teknologi mesin penerjemahan ini dalam aplikasi yang dibuat sendiri, layanan API ini dapat dipanggil menggunakan banyak cara, termasuk HTTP SISA Service, layanan AJAX-callable dan Layanan SOAP Web.

Microsoft Translator API tersedia di *Windows Azure Marketplace*. Untuk menggunakan Microsoft Translator API para *developer* harus berlangganan terlebih dahulu ke *Windows Azure Marketplace* dan membayar layanan tersebut berdasarkan jumlah karakter teks yang dikirimkan ke API setiap bulannya[13].

Contoh dari metode yang disediakan oleh API untuk para *developer* adalah sebagai berikut:

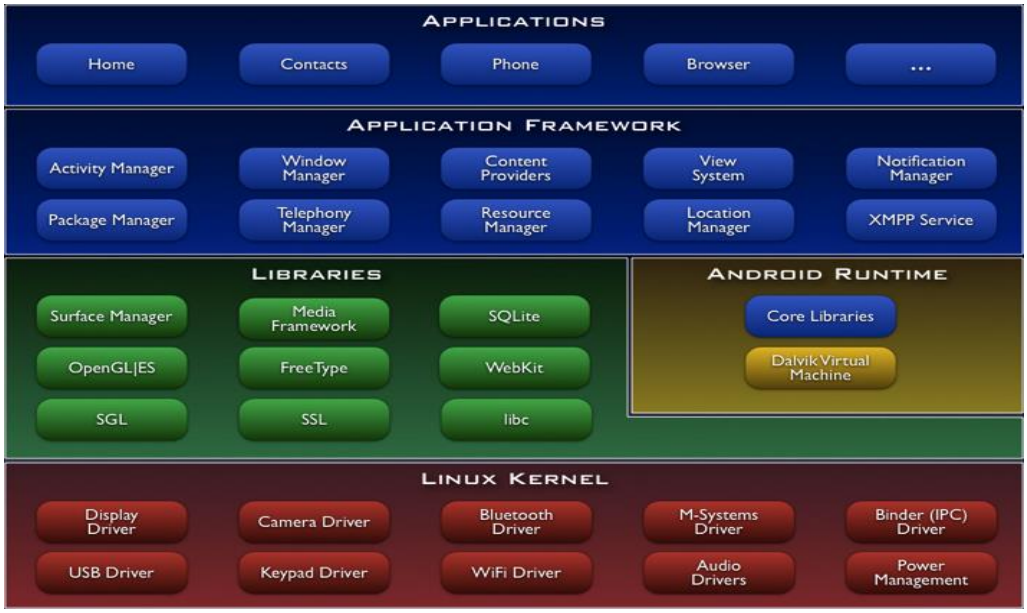
1. Translate: Teks diterjemahkan dari satu bahasa ke bahasa lain
2. TranslateArray: Menerjemahkan berbagai teks dari satu bahasa ke bahasa lain
3. Detect: Mendeteksi bahasa pada string
4. Detect Array: Mendeteksi bahasa pada string array
5. GetTranslations: Mengembalikan sebuah array terjemahan alternatif untuk teks
6. GetTranslationsArray: Mengembalikan sebuah array terjemahan alternatif untuk array teks

2.6 Android

Android adalah sistem operasi untuk telepon seluler yang berbasis Linux. Android menyediakan platform terbuka bagi para pengembang buat menciptakan aplikasi mereka sendiri untuk digunakan oleh bermacam piranti bergerak. Awalnya, Google Inc. membeli Android Inc., pendatang baru yang membuat peranti lunak untuk ponsel. Kemudian untuk mengembangkan Android, dibentuklah *Open Handset Alliance*, konsorsium dari 34 perusahaan peranti keras, peranti lunak, dan telekomunikasi, termasuk Google, HTC, Intel, Motorola, Qualcomm, T-Mobile, dan Nvidia.

2.6.1 Arsitektur Android

Google menggambarkan Android seperti sebuah tumpukan software. Setiap lapisan dari tumpukan ini terdiri dari beberapa program yang mendukung fungsi-fungsi spesifik dari sistem operasi. Dalam paket sistem operasi android tediri dari beberapa unsur seperti tampak pada gambar 2.6. Secara sederhana arsitektur android merupakan sebuah kernel Linux dan sekumpulan pustaka C / C++ dalam suatu Framework yang menyediakan dan mengatur alur proses aplikasi.



Gambar 2.6 Arsitektur android[14]

Berikut adalah penjelasan mengenai masing-masing lapisan pada arsitektur android:

1. Linux Kernel

Android dibangun di atas kernel Linux. Namun secara keseluruhan android bukanlah linux, karena dalam android tidak terdapat paket standar yang dimiliki oleh linux lainnya. Linux merupakan sistem operasi terbuka yang handal dalam manajemen memori dan proses. Oleh karenanya pada android hanya terdapat beberapa servis yang diperlukan seperti keamanan, manajemen memori, manajemen proses, jaringan dan *driver*. Kernel linux menyediakan *driver* layar, kamera, *keypad*, *WiFi*, *Flash Memory*, audio, dan IPC (*Interprocess Communication*) untuk mengatur aplikasi dan lubang keamanan.

2. Libraries

Android menggunakan beberapa paket pustaka yang terdapat pada C/C++ dengan standar Berkeley Software Distribution (BSD) hanya setengah dari yang aslinya untuk tertanam pada kernel Linux. Beberapa pustaka diantaranya:

1. Media Library untuk memutar dan merekam berbagai macam format audio dan video.
2. Surface Manager untuk mengatur hak akses layer dari berbagai aplikasi.
3. Graphic Library termasuk didalamnya SGL dan OpenGL, untuk tampilan 2D dan 3D.
4. SQLite untuk mengatur relasi database yang digunakan pada aplikasi.
5. SSl dan WebKit untuk browser dan keamanan internet.

Pustaka-pustaka tersebut bukanlah aplikasi yang berjalan sendiri, namun hanya dapat digunakan oleh program yang berada di level atasnya. Sejak versi Android 1.5, pengembang dapat membuat dan menggunakan pustaka sendiri menggunakan Native Development Toolkit (NDK).

3. Android Runtime

Pada android tertanam paket pustaka inti yang menyediakan sebagian besar fungsi android. Inilah yang membedakan Android dibandingkan dengan sistem operasi lain yang juga mengimplementasikan Linux. Android Runtime merupakan mesin virtual yang membuat aplikasi android menjadi lebih tangguh dengan paket pustaka yang telah ada. Dalam Android Runtime terdapat 2 bagian utama, diantaranya:

1. Pustaka Inti, android dikembangkan melalui bahasa pemrograman Java, tapi Android Runtime bukanlah mesin virtual Java. Pustaka inti android menyediakan hampir semua fungsi yang terdapat pada pustaka Java serta beberapa pustaka khusus android.
2. Mesin Virtual Dalvik, Dalvik merupakan sebuah mesin virtual yang dikembangkan oleh Dan Bornstein yang terinspirasi dari nama sebuah perkampungan yang berada di Iceland. Dalvik hanyalah interpreter mesin virtual yang mengeksekusi file dalam format *Dalvik Executable* (*.dex). Dengan format ini Dalvik akan mengoptimalkan efisiensi penyimpanan dan pengalamatan memori pada file yang dieksekusi. Dalvik berjalan di atas kernel Linux 2.6, dengan fungsi dasar seperti *threading* dan manajemen memori yang terbatas.

4. Aplication Framework

Kerangka aplikasi menyediakan kelas-kelas yang dapat digunakan untuk mengembangkan aplikasi android. Selain itu, juga menyediakan abstraksi untuk mengakses perangkat, serta mengatur tampilan *User Interface* dan sumber daya aplikasi. Bagian terpenting dalam kerangka aplikasi android adalah sebagai berikut:

1. Activity Manager, berfungsi untuk mengontrol siklus hidup aplikasi dan menjaga keadaan ”*Backstack*“ untuk navigasi penggunaan.
2. Content Providers, berfungsi untuk merangkum data yang memungkinkan digunakan oleh aplikasi lainnya, seperti daftar nama.

3. Resource Manager berfungsi untuk mengatur sumber daya yang ada dalam program. Seperti karakter, grafik dan *file* layout.
4. Location Manager, berfungsi untuk memberikan informasi detail mengenail lokasi perangkat android berada.
5. Notification Manager, mencakup berbagai macam peringatan seperti, pesan masuk, janji, dan lain sebagainya yang akan ditampilkan pada status bar.

5. Aplication Layer

Puncak dari diagram arsitektur android adalah lapisan aplikasi dan widget. Lapisan aplikasi merupakan lapisan yang paling tampak pada pengguna ketika menjalankan program. Pengguna hanya akan melihat program ketika digunakan tanpa mengetahui proses yang terjadi dibalik lapisan aplikasi. Lapisan ini berjalan dalam *Android runtime* dengan menggunakan kelas dan *service* yang tersedia pada Framework aplikasi. Lapisan aplikasi android sangat berbeda dibandingkan dengan sistem operasi lainnya. Pada android semua aplikasi, baik aplikasi inti (*native*) maupun aplikasi pihak ketiga berjalan diatas lapisan aplikasi dengan menggunakan pustaka API (*Application Programming Interface*) yang sama.

2.7 UML

UML adalah kependekan dari Unified Modeling Language yang merupakan suatu cara untuk menyelesaikan suatu masalah dengan mendeskripsikannya yang telah menjadi standar dalam dunia industri untuk memvisualisasikan, merancang dan mendokumentasikan sistem perangkat lunak. Dengan menggunakan UML kita dapat membangun model untuk segala bentuk dan jenis aplikasi perangkat

lunak, yang mana aplikasi yang dibangun dapat berjalan pada perangkat lunak dengan sistem operasi dan jaringan apapun. Selain itu, dengan menggunakan UML, programmer dapat mengerti, memahami dengan jelas maksud, tujuan dan arah rancangan sistem, sehingga sistem perngakat lunak yang dirancang dapat dibangun dengan bahasa program apapun. Namun karena UML merupakan suatu model pengembangan sistem perangkat lunak yang berbasis object oriented sehingga menggnuakan bentuk class dan operation dalam konsep dasarnya, maka lebih cocok dalam pemrogramannya dengan menggunakan bahasa-bahasa pemrograman yang berbasis OOP (*Object Oriented Programming*) seperti C, C++, Java, VB dan lain sebagainya.

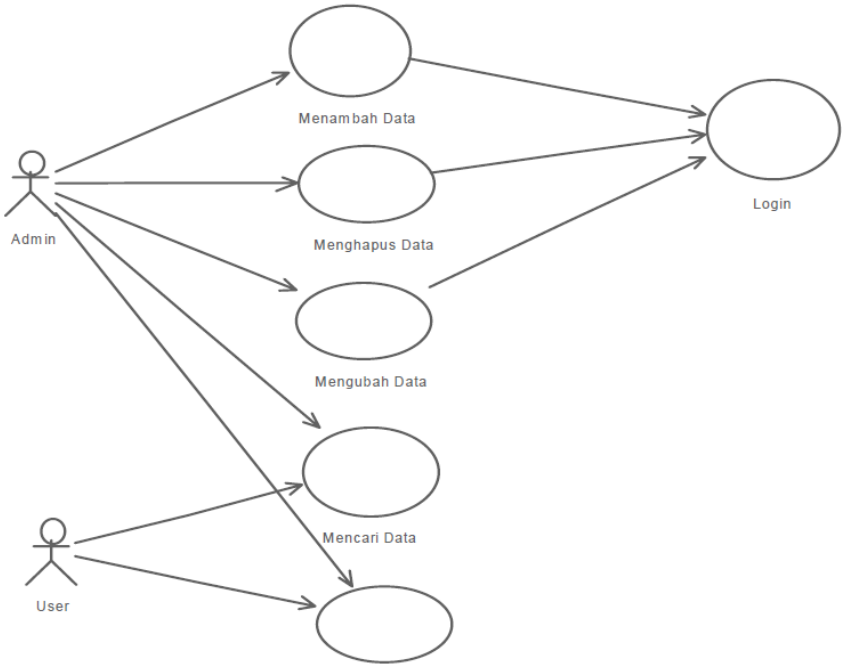
Secara lebih mendalam, UML lebih dari sekedar sebuah standard dan penemuan dari suatu notasi-notasi yang disatukan, tetapi juga berisi konsep-konsep baru yang menarik yang tidak ditemukan secara umum dalam komunitas object oriented. Dalam membangun suatu model perangkat lunak dengan UML, digunakan bentuk-bentuk diagram atau symbol untuk merepresentasikan elemen-elemen dalam sistem. Bentuk diagram yang digunakan untuk merepresentasikannya adalah sebagai berikut[8]:

1. Use-case Diagram
2. Class Diagram
3. Sequence diagram
4. Collaboration Diagram
5. Activity Diagram
6. Component Diagram
7. Deployment Diagram

2.7.1 Use-Case Diagram

Use-case diagram merupakan suatu bentuk diagram yang menggambarkan fungsi-fungsi yang diharapkan dari sebuah sistem yang dirancang. Dalam *Use-case diagram* penekanannya adalah "*apa*" yang diperbuat oleh sistem, dan bukan "*bagaimana*". Sebuah *use-case* akan merepresentasikan sebuah interaksi antara pelaku atau *actor* dengan sistem.

Use-case diagram yang digunakan dalam mercancang suatu sistem dapat sangat membantu pada saat menyusun *requirement* sebuah sistem, mengomunikasikannya dengan klien, dan merancang pengujian untuk semua fitur yang terdapat dalam sistem. Dalam suatu sistem aplikasi database, *use-case diagram* sangat membantu *requierement* apa saja yang diperlukan.



Gambar 2.7 Contoh Use-case Diagram[15]

2.7.2 Class Diagram

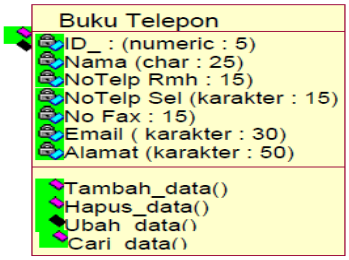
Sebuah *Class Diagram* menunjukkan struktur yang statis dari beberapa class dalam suatu sistem. *Class-class* merepresentasikan suatu keadaan (atribut/properti) dan yang akan dikerjakan oleh sistem (metoda/fungsi). Class memiliki tiga area pokok:

1. Nama (dan *stereotype*)
2. Atribut
3. Metoda

Atribut dan metoda dalam class diagram dapat memiliki salah satu sifat seperti berikut di bawah ini:

1. Private, hanya dapat diakses oleh class itu sendiri.
2. Protected, hanya dapat diakses oleh class itu sendiri dan turunan dari class tersebut.
3. Public, dapat diakses oleh class selain dari class yang bersangkutan.

Class dapat direpresentasikan dalam sebuah *interface* atau sebaliknya merupakan implementasi dari sebuah *interface* yang berupa class abstrak yang hanya tidak memiliki *attribute* dan hanya memiliki metoda. Berikut merupakan bentuk class diagram secara umum:

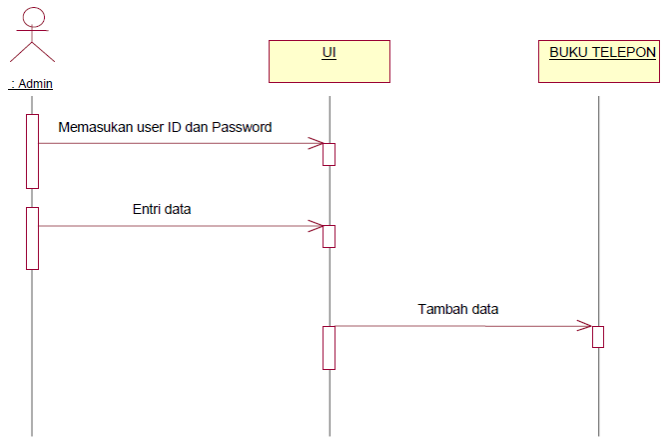


Gambar 2.8 Contoh Class Diagram[15]

2.7.3 Sequance Diagram

Sequence Diagram merupakan diagram yang mengambarkan kolaborasi yang dinamis antara obyek satu dengan yang lain. Kolaborasi ini ditunjukkan dengan adanya interaksi antar obyek di dalam dan di sekitar sistem yang berupa pesan atau instruksi yang berurutan.

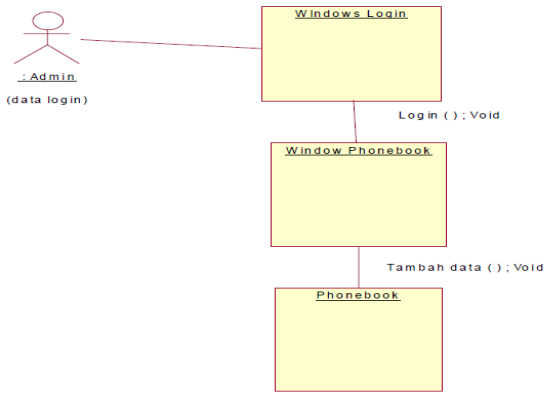
Sequence diagram umumnya digunakan untuk menggambarkan suatu skenario atau urutan langkah-langkah yang dilakukan baik oleh *actor* maupun sistem yang merupakan respon dari sebuah kejadian untuk mendapatkan hasil atau *output*.



Gambar 2.9 Contoh Sequance Diagram[15]

2.7.4 Collaboration Diagram

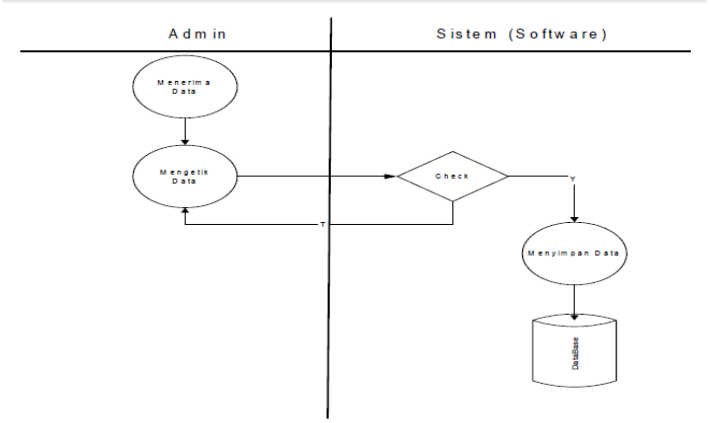
Sebuah *collaboration diagram* menunjukkan kolaborasi yang dinamis yang mirip dengan *sequence diagram*. *Collaboration diagram* digambarkan sebagai sebuah *obiject* diagram dimana sejumlah obyek ditunjukkan disekitarnya dengan hubungan-hubungannya.



Gambar 2.10 Contoh Collaboration Diagram[15]

2.7.5 Activity Diagram

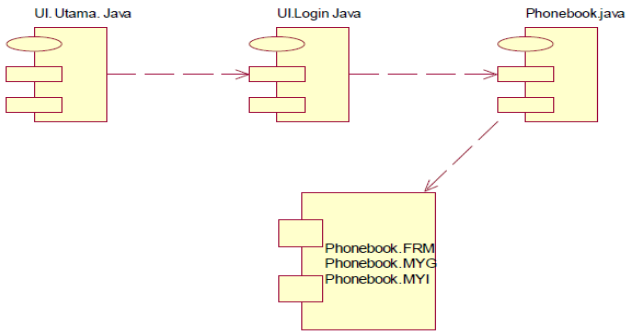
Sebuah *Activity Diagram* menunjukkan suatu alur kegiatan secara berurutan. *Activity Diagram* digunakan untuk mendiskripsikan kegiatan-kegiatan dalam sebuah operasi meskipun juga dapat digunakan untuk mendeskripsikan alur kegiatan yang lainnya seperti *use case* atau suatu interaksi.



Gambar 2.11 Contoh Activity Diagram[15]

2.7.6 Component Diagram

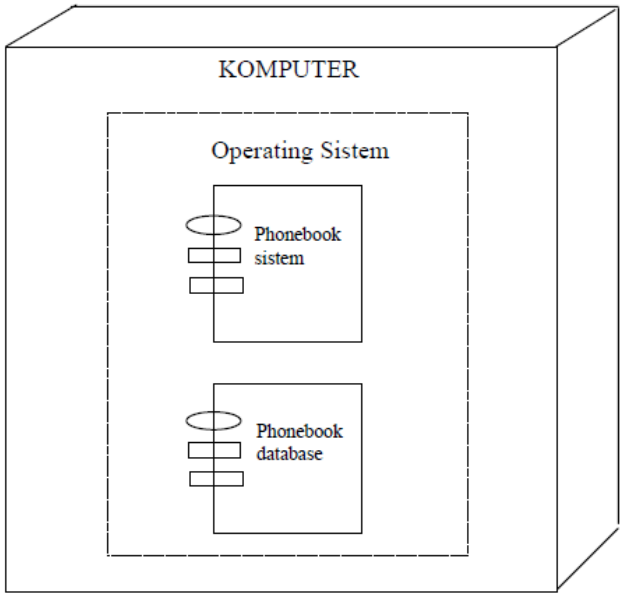
Component Diagram menunjukkan struktur dan hubungan antar komponen *software* termasuk ketergantungan (*dependency*) diantara komponen-komponen tersebut. Komponen pada piranti lunak adalah berupa modul-modul yang berisikan *code*, baik *library* maupun *executable*. Umumnya komponen yang terbentuk dari beberapa *class* dan/atau *package*, atau juga dapat dari komponen-komponen yang lebih kecil.



Gambar 2.12 Component Diagram[15]

2.7.7 Deployment Diagram

Deployment Diagram menunjukkan arsitektur fisik pada *hardware* dan *software* pada suatu sistem yang dirancang. *Deployment diagram* juga dapat menunjukkan perngkat-perangkat dan *nodes* diantara hubungan yang dimilikinya antar komponen.



Gambar 2.13 Contoh Deployment Diagram[15]