BAB 2
TINJAUAN PUSTAKA

1.2. Mobile Robot

Mobile Robot adalah konstruksi robot yang ciri khasnya adalah mempunyai aktuator berupa roda untuk menggerakkan keseluruhan badan robot tersebut, sehingga robot tersebut dapat melakukan perpindahan posisi dari satu titik ke titik lain dengan bantuan navigasi. Ada banyak navigasi yang digunakan *mobile* robot, diantaranya [3]:

1. Manual Remote

Sebuah robot manual benar-benar dibawah kendali seorang sopir dengan joystick atau perangkat kontrol lainnya. Perangkat mungkin dihubungkan langsung ke robot, mungkin *joystick* nirkabel, atau mungkin menjadi aksesori ke komputer nirkabel atau pengendali lainnya. Sebuah robot tele-op'd biasanya digunakan untuk menjaga operator dari bahaya.

2. *Guarded* Tele-op

Sebuah *robot guarded tele-op* memiliki kemampuan untuk merasakan dan menghindari rintangan tetapi sebaliknya akan menavigasi sebagai penggerak, seperti robot di bawah manual *tele-op*. Jika ada beberapa robot *mobile* hanya menawarkan *guarded* tele-op *Sliding.*

3. Next-line

Beberapa *Automated* awal Dipandu Kendaraan (AGVs) adalah baris berikut mobile robot. Mereka mungkin mengikuti garis visual dicat atau tertanam di lantai atau langit-langit atau sebuah kabel listrik di lantai. Mereka tidak bisa mengelilingi hambatan, mereka hanya berhenti dan menunggu ketika sesuatu menghalangi jalan mereka. Banyak contoh dari kendaraan tersebut masih dijual, oleh *Transbotics* , FMC, Egemin, HK *Systems* dan perusahaan lainnya.

4. Autonomously Randomized Robot

Otonomi robot dengan gerakan acak pada dasarnya terpental dinding, baik dinding-dinding yang merasakan dengan bumper fisik seperti pembersih Roomba atau dengan sensor elektronik seperti mesin pemotong rumput Robotika *Friendly*.

Algoritma sederhana *bump* dan putar 30 derajat akhirnya mengarah ke jangkauan sebagian besar atau seluruh permukaan lantai atau halaman.

Robot mobil ini sangat disukai bagi orang yang mulai mempelajari robot. Hal ini karena membuat robot mobil tidak memerlukan kerja fisik yang berat. Untuk dapat membuat sebuah robot mobile minimal diperlukan pengetahuan tentang mikrokontroler dan sensor-sensor elektronik.

1.3. Pencarian Jalur Terpendek

Pencarian jalur/rute (*pathfinding*) adalah salah satu bidang penerapan yang sering ditangani oleh kecerdasan buatan khususnya dengan menggunakan algoritma pencarian. Penerapan yang dapat dilakukan dengan *pathfinding* antara lain adalah pencarian rute dalam suatu *game* dan pencarian jalan/rute pada suatu peta. Algoritma pencarian yang dipakai harus dapat mengenali jalan dan elemen peta yang tidak dapat dilewati. Sebuah algoritma *pathfinding* yang baik dapat bermanfaat untuk mendeteksi halangan/rintangan yang ada pada medan dan menemukan jalan menghindarinya, sehingga jalan yang ditempuh lebih pendek daripada yang seharusnya bila tidak menggunakan algoritma *pathfinding* [2].

1.4. Algortima A*

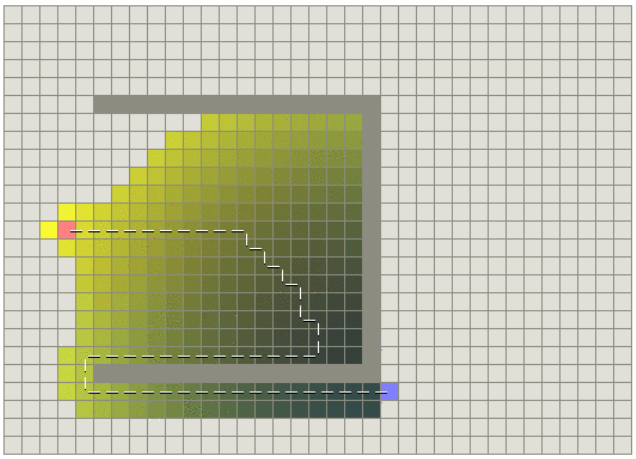
Algoritma A*, dapat juga disebut sebagai algoritma A Star, merupakan salah satu contoh algoritma pencarian yang cukup popular di dunia. Beberapa terminology dasar yang terdapat pada algoritma ini adalah *starting point*, simpul (*nodes*), A, *open list*, *closed list*, harga (*cost*), halangan (*unwalkable*).

Starting point adalah sebuat terminology untuk posisi awal pada sebuah benda. A adalah simpul yang sedang dijalankan dalam algoritma pencarian jalan terpendek. Simpul adalah petak-petak kecil sebagai representasi dari area *pathfinding*. Bentuknya dapat berupa persegi, lingkaran maupun segitiga. *Open list* adalah tempat menyimpan data simpul yang mungkin diakses dari starting point maupun simpul yang sedang dijalankan. *Closed list* adalah tempat menyimpan data simpul sebelum A yang juga merupakan bagian dari jalur terpendek yang btelah berhasil didapatkan. Harga (F) adalah nilai yang diperoleh dari penjumlahan nilai G, jumlah nilai tiap simpul dalam jalur terpendek dari *starting point* ke A, dan H, jumlah nilai perkiraan dari sebuah simpul ke simpul tujuan. Simpul tujuan yaitu simpul yang

dituju. Rintangan adalah sebuah atribut yang menyatakan bahwa sebuah simpul tidak dapat dilalui oleh A. Prinsip algoritma ini adalah mencari jalur terpendek dari sebuah simpul awal menuju simpul tujuan dengan memperhatikan harga (F) terkecil. Diawali dengan menempatkan A pada *starting point*, kemudian memasukan seluruh simpul yang bertetangga dan tidak memiliki atribut rintangan dengan A ke dalam *open list*. Kemudian mencari nilai H terkecil dari simpul-simpul dalam opn list tersebut. Kemudian memindahkan A ke simpul yang memiliki H terkecil. Simpul sebelum A disimpan sebagai parent dari A dan dimasukan ke dalam closed list. Jika terdapat simpul lain yang bertetangga dengan A (yang sudah berpindah) maka masukan simpul-simpul tersebut ke dalam open list. Setelah itu, bandingkan nilai G yang ada dengan nilai G sebelumnya (pada langkah awal, tidak perlu dilakukan perbandingan nilai G). Jika nilai G sebelumnya lebih kecil maka kembali ke posisi awal. Simpul yang pernah dicoba dimasukan ke dalam closed list. Hal tersebut dilakukan berulang-ulang hingga terdapat solusi atau tidak ada lagi simpul yang berada pada open list [2]. Fungsi heuristik untuk menghitung taksiran nilai yang diberikan pada simpul dapat dinnyatakan sebagai berikut:

f(i) = g(i) + hi), dimana
f(i) = ongkos untuk simpul i
g(i) = ongkos mencapai simpul i dari akar
h(i) = ongkos mencapai simpul tujuan dari simpul i

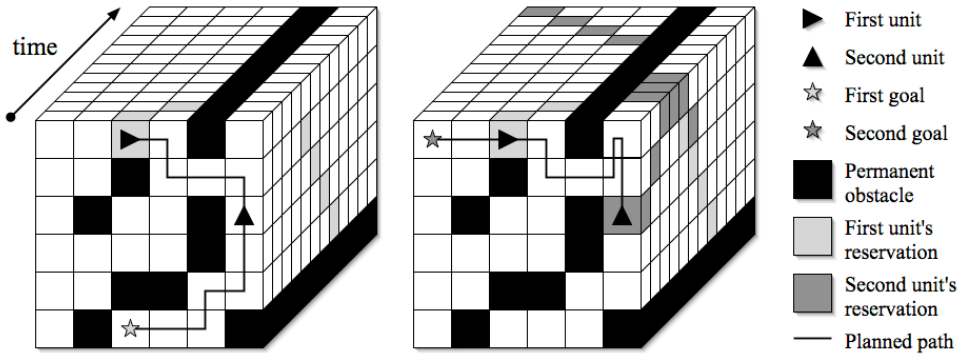
Dibawah ini adalah contoh lingkungan A*.



Gambar 2.1. Contoh lingkungan A* dalam grid [3]

1.5.Cooperative A*

Ketika melakukan pencarian jalur terpendek dengan satu unit berdasarkan peta, pencarian menggunakan dasar A* sangat memadai, tetapi ketika beberapa unit bergerak pada saat yang sama, pendekatan dasar A* tidak bisa terpecahkan. Cooperative A* (CA*) adalah algoritma baru untuk memecahkan pencarian jalur terpendek secara kooperatif. Tugas dipisahkan menjadi serangkaian pencarian agen tunggal. Pencarian individu dilakukan dalam tiga dimensi ruang-waktu dan memperhitungkan rute yang direncanakan oleh agen lainnya. Langkah menunggu termasuk dalam tindakan agen. Setelah rute masing-masing agen dihitung setiap node ditandai ke tabel reservasi. Setelah unit memilih jalan, perlu memastikan bahwa unit lain tahu untuk menghindari sel-sel di sepanjang jalan. Hal ini dicapai dengan menandai setiap sel dalam tabel reservasi. Ini adalah struktur data langsung berisi entri untuk setiap sel dari peta ruang-waktu. Setiap entri menentukan apakah sel yang sesuai tersedia atau dilindungi. Setelah entri dicadangkan adalah illegal untuk unit lain untuk pindah ke sel itu. Tindakan reservasi seperti kendala sementara, menghalangi dari lokasi untuk waktu langkah di masa depan. Menggunakan meja reservasi dan peta ruang-waktu, maka sistem dapat memecahkan masalah kooperatif jalur terpendek. Setiap unit mencari jalur terpendek dengan menggunakan ruang-waktu A* dan kemudian menandai jalan ke dalam tabel reservasi. Unit berikutnya menghindari sel yang telah ditandai.



Gambar 2.2. Contoh 2 unit Cooperative Pathfinding

Tetapi, dengan cara tabel reservasi tidak mencegah dua unit melintas melalui satu sama lain saling berhadapan. Jika satu unit dilindungi (x,y,t) dan (x+1,y,t+1), tidak ada yang menghentikan unit kedua dari pemesanan (x+1, y,t) dan (x, y, t+1). Masalah ini dapat dihindari dengan membuat dua pemesanan untuk

setiap lokasi yang terlibat dalam tindakan, satu per- t dan satu waktu t+1 atau tabrakan bisa diidentifikasi secara eksplisit.

Sebuah pertanyaan alami untuk bertanya tentang merintis pencarian jalur terpendek secara kooperatif adalah seberapa jauh ke depan agen dapat melihat? Untuk bekerja sama dengan sempurna melihat ke masa depan, *agent* perlu melihat sejauh jalan terpanjang. Tapi ini mungkin akan menjadi ratusan *timesteps*, Dan kejadian itu akan menghabiskan berapa banyak waktu. Ruang waktu A* akan perlu mencari peta yang lebih besar, karena semakin lama dimensi waktu, meja reservasi akan perlu menyimpan path untuk semua unit dengan ratusan langkah ke depan. Idealnya, *agent* harus memiliki beberapa batas atas berapa banyak kerjasama diperlukan dan hanya mencari ke masa depan. Perkiraan ini harus mempertimbangkan sifat peta dan jumlah unit di wilayah itu, sebuah untuk melintasi daerah rami, jembatan sempit mungkin akan tertunda jauh lebih lama dari unit dengan melewati jalur dengan dataran yang sepi.

Dalam algoritma Cooperative A* batas kedalaman, dalam pencarian secara kooperatif akan dilambangkan dengan *d.* untuk tujuan dari ilustrasi, diasumsikan bahwa *d* adalah konstan untuk semua unit, tapi ini tidak perlu terjadi. Kedalaman ruang-waktu A* terbatas untuk langkah-langkah d. jika tujuan belum pernah tercapai, maka jalan parsial dikembalikan. Unit dimulai mengikuti jalan parsial sampai diperlukan untuk menghitung yang baru [1].

1.6. HPA* (*Hierarchical Pathfinding* A*)

HPA* adalah studi rinci dari algoritma pathfinding hirarkis untuk merintis jalan pada peta *game* computer. Algoritma ini membagi tugas pencarian jalan menjadi dua tahap yaitu *offline* preprosesing dan *online pathfinding* [4] .

Grafik yang telah disarikan untuk *online* pencarian dibangun menggunakan informasi yang diambil dari masalah labirin untuk lebih detail bagaimana kerangka kerja untuke pencarian hirarkis dibangun (pre-prosesing) dan bagaimana digunakan dalam *online pathfinding*. Awalnya focus pada membangun hirarkis dua tingkat: satu tingkat rendah dan satu tingkat abstrak [5].

2.6.1. *Pre-processing grid*

Langkah pertama dalam membangun kerangka untuk hirarkis pencarian sebuah topologi abstraksi dari labirin. Labirin abstraksi digunakan untuk membangun grafik dalam pencarian hirarkis. Abstraksi topologi meliputi labirin dengan satu set persegi panjang daerah yang disebut *cluster*. Contoh, dalam grid 40 x 40 akan dikelompokan ke dalam 16 cluster dengan ukuran 10 x 10. Untuk setiap garis perbatasan Antara dua kelompok yang berdekatan diidentifikasi set pintu masuk yang menghubungkan mereka. Pintu masuk adalah maksimal hambatan bebas 8 segmen sepanjang perbatasan bersama dua cluster c1 dan c2 berdekatan, secara resmi didefinisikan sebagai berikut. Pertimbangkan dua baris yang berdekatan ubin l1 dan l2 di setiap cluster, yang menentukan tepi perbatasan Antara c1 dan c2. Untuk ubin t2 anggota l1 gabung l2, dengan definisi symm (t) sebagai ubin simetris t terhadap perbatasan antara c1 dan c2. Perhatikan bahwa t dan symm (t) yang berdekatan dan tidak pernah milik cluster yang sama. Pintu masuk adalah seperangkat ubin yang menghormati kondisi berikut:

1. Perbatasan keterbatasan kondisi: l1 dan l2. Kondisi ini menyatakan bahwa pintu masuk adalah de ned bersama dan tidak bisa melebihi perbatasan antara dua cluster yang berdekatan.
2. Kendala kondisi bebas: pintu masuk tidak mengandung ubin kendala
3. Kondisi maximality: pintu masuk diperpanjang di kedua arah sebagai Selama kondisi sebelumnya tetap berlaku.

2.6.2. Online Search

Tahap pertama dari pencarian on-line menghubungkan posisi awal S ke perbatasan cluster yang mengandung S. Langkah ini selesai sementara memasukkan S ke grafik abstrak. Demikian pula, menghubungkan posisi tujuan G ke perbatasan klaster ditangani dengan memasukkan G ke dalam grafik abstrak. Setelah S dan G telah ditambahkan, kita menggunakan A * untuk mencari jalan antara S dan G dalam grafik abstrak. Ini adalah bagian yang paling penting dari pencarian on-line. Ini menyediakan jalur abstrak, bergerak sebenarnya dari S ke perbatasan cluster S, jalan abstrak cluster G, dan bergerak yang sebenarnya dari perbatasan cluster G. Dua langkah terakhir dari pencarian online adalah opsional:

1. Jalan-ulang nement dapat digunakan untuk mengkonversi jalur abstrak ke dalam urutan bergerak pada grid asli.
2. Jalur-smoothing dapat digunakan untuk meningkatkan kualitas jalan-ulang nementukan solusi.

Jalur abstrak dapat kembali ned dalam langkah pasca-pengolahan untuk mendapatkan rinci jalur dari S ke G. Bagi banyak real-time aplikasi nding path-, path lengkap tidak diperlukan hanya beberapa langkah pertama. Informasi ini memungkinkan karakter untuk mulai bergerak ke arah yang benar menuju tujuan. Sebaliknya, A * harus menyelesaikan pencarian dan menghasilkan seluruh jalan dari S ke G sebelum dapat menentukan pertama yang langkah karakter. Pertimbangkan domain di mana perubahan dinamis sering terjadi (misalnya, ada banyak unit mobile berkeliling). Dalam kasus seperti itu, setelah nding abstrak jalan, kita bisa kembali ne secara bertahap sebagai karakter menavigasi menuju tujuan. Jika jalan abstrak saat ini menjadi tidak valid, agen membuang itu dan mencari yang lain jalan abstrak. Tidak perlu untuk kembali ne jalan seluruh abstrak di muka.

1.7.WHCA* (*Windowed Hierarchical Cooperative* A*)

Salah satu masalah dengan algoritma sebelumnya adalah bagaimana algoritma hanya membatasi pada sekali pencarian agen mencapai tujuan. Jika agen duduk pada tujuan, misalnya dalam koridor yang sempit, maka mungkin blok dari bagian peta untuk agen lainnya. Idealnya, agen harus terus bekerja sama setelah mencapai tujuan, sehingga agen dapat bergerak dari tujuan dan memungkinkan orang lain untuk lulus. Isu kedua adalah kepekaan terhadap agen pemesanan. Meskipun kadang-kadang mungkin untuk memprioritaskan agen global solusi yang lebih kuat adalah untuk secara dinamis bervariasi urutan agen, sehingga setiap agen akan memiliki tertinggi prioritas untuk waktu singkat. Solusi kemudian dapat ditemukan yang akan dipecahkan dengan sewenang-wenang agen. Ketiga, algoritma sebelumnya harus menghitung lengkap rute ke tujuan dalam besar, tiga-dimensi ruang *space*. Dengan pencarian agen tunggal, perencanaan dan rencana eksekusi sering disisipkan untuk mencapai efisiensi yang lebih besar dengan menghindari kebutuhan untuk merencanakan kontinjensi jangka panjang yang tidak sebenarnya terjadi. WHCA * mengembangkan ide serupa untuk koperasi pencarian. Sebuah solusi sederhana untuk semua masalah ini adalah untuk jendela pencarian.

Pencarian koperasi terbatas pada kedalaman tetap ditentukan oleh jendela saat. Setiap agen pencarian untuk rute parsial untuk tujuan, dan kemudian mulai mengikuti rute. Secara berkala (misalnya ketika agen adalah setengah jalan melalui rute parsial) jendela digeser ke depan dan rute parsial baru dihitung. Untuk memastikan bahwa agen kepala di arah yang benar, hanya kedalaman pencarian koperasi terbatas pada kedalaman tetap, sementara pencarian abstrak dijalankan untuk kedalaman penuh. Sebuah jendela ukuran w dapat dilihat sebagai abstraksi menengah yang setara dengan negara ruang tingkat dasar untuk langkah-langkah w, dan kemudian setara dengan ruang negara tingkat abstrak untuk sisa pencarian. Dengan kata lain, agen lainnya hanya dipertimbangkan untuk langkah w (melalui meja reservasi) dan diabaikan untuk sisa pencarian.

Untuk mencari ini ruang pencarian baru efisien, trik sederhana dapat digunakan. Setelah langkah langkah w telah berlalu, agen diabaikan dan ruang pencarian menjadi identik dengan pencarian abstrak ruang. Ini berarti bahwa jarak abstrak menyediakan informasi yang sama seperti menyelesaikan pencarian. Untuk setiap node dicapai setelah langkah w tepi terminal khusus diperkenalkan, akan langsung dari N*i* ke tujuan G, dengan biaya sama dengan jarak abstrak dari N*i* ke G. Menggunakan ini trik, pencarian direduksi menjadi jendela w-langkah menggunakan heuristik jarak abstrak diperkenalkan untuk HCA *. Selain itu, pencarian berjendela dapat melanjutkan setelah agen telah mencapai tujuan. Tujuannya agen tidak lebih lama untuk mencapai tujuan, tetapi untuk menyelesaikan jendela melalui tepi terminal. Setiap urutan w bergerak akan demikian mencapai tujuan. Namun, WHCA * pencari akan efisien menemukan urutan biaya terendah. Urutan optimal ini merupakan rute parsial yang akan membawa agen yang paling dekat dengan-nya tujuan, dan sekali di sana untuk tinggal di tujuan untuk sebagai waktu sebanyak mungkin.

Secara umum , fungsi biaya tepi untuk WHCA * adalah:

$$\text{COST}(P,Q) = \begin{cases} 0 & \text{if } P = Q = G, t < w \\ \text{ABSTRACTDIST}(P,G) & \text{if } t = w \\ 1 & \text{otherwise} \end{cases}$$

Gambar 2.3. Biaya Tepi WHCA*

Sebuah keuntungan tambahan dari windowing adalah proses yangwaktu dapat tersebar di semua agen. Dengan mengejutkan jendela tepat, pencarian dapat

lancar disisipkan. Dengan agen n dan ukuran jendela w , menghitung ulang rute pada titik tengah dari setiap jendela, hanya 2n pencarian / w perlu dilakukan per giliran. Jika giliran terdiri dari banyak frame, maka pencarian resumable alami rusak lebih lanjut dan dapat tersebar di beberapa frame [1].

1.8. UML (*Unified Modeling Language)*

UML adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sebuah sistem. UML adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat *tool* untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980- an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM.

UML mendefinisikan diagram-diagram sebagai berikut:

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

2.8.1 Usecase Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang atau sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

Use case diagram dapat sangat membantu menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case*

untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal.

Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

2.8.2 Activity Diagram

Activity diagram menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alur berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, dimana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan *behaviour* internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

2.8.3 Sequence Diagram

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pemain, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan

2.8.4. Class Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).

Class diagram menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

2.8.5. Statechart Diagram

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya) suatu objek pada sistem sebagai akibat dari *stimuli* yang diterima. Pada umumnya *statechart diagram* menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *statechart diagram*).

Dalam UML, *state* digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu.

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.

2.8.6. Collaboration Diagram

Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence diagram*, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*. Setiap *message* memiliki *sequence number*, di mana *message* dari level tertinggi memiliki nomor 1. *Messages* dari *level* yang sama memiliki *prefiks* yang sama.

2.8.7. Component Diagram

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) di antaranya. Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time, link time*, maupun *run time*.

Umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa *interface,* yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.

2.8.8 Deployment Diagram

Deployment/physical diagram menggambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, *server* atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi *server,* dan hal-hal lain yang bersifat fisikal.

Sebuah *node* adalah *server, workstation,* atau piranti keras lain yang digunakan untuk men-*deploy* komponen dalam lingkungan sebenarnya. Hubungan antar *node* (misalnya TCP/IP) dan *requirement* dapat juga didefinisikan dalam diagram ini [6].

1.9.Pengujian Perangkat Lunak

Pengujian perangkat lunak merupakan proses verifikasi dan validasi dari sebuah perangkat lunak [6]. Pengujian perangkat lunak juga mengidentifikasi kelemahan dan kesalahan dalam kode aplikasi yang harus diperbaiki. Pengujian perangkat lunak memiliki tiga tujuan penting yaitu: verifikasi, validasi dan menemukan cacat. Verifikasi adalah proses menegaskan bahwa perangkat lunak sudah memenuhi spesifikasi teknik. Validasi adalah proses menegaskan bahwa perangkat lunak memenuhi kebutuhan bisnis. Menemukan cacat artinya varians Antara hasil yang diharapkan dengan aktual.

2.9.1 Unit Testing

Unit Testing merupakan serangkaian tes yang berdiri sendiri. Setiap tes meneliti komponen individu yang baru atau telah dimodifikasi. Setiap tes memvalidasi modul tunggal, berdasarkan dokumen teknis yang dibangun untuk melakukan tugas tertentu. Unit testing berfokus pada fungsi dan kehandalan sebuah perangkat lunak. Satuan pengujian dilakukan di lingkungan tes sebelum integrasi sistem [6]