BAB II
LANDASAN TEORI

2.1 Game Strategi

Game strategi menurut Melvin Dresher [**4**] digambarkan oleh suatu peraturan. Peraturan ini memiliki spesifikasi secara jelas setiap pemain dapat melakukan sesuatu di berbagai kondisi. Peraturan dijelaskan dengan berbagai informasi, jika salah satu atau suatu pemain menerima. Jika permainan membutuhkan suatu alat kesempatan, aturan yang digambarkan bagaimana kesempatan peristiwa akan dilaksanakan. Memungkinkan juga berakhirnya waktu permainan.

Aturan yang didapat dari properti permainan seperti berapa langkah bergerak, banyak pemain, dan hasilnya seperti apa. Setiap langkah memiliki nilai dan pemain memiliki berapa banyak langkah yang tersedia.

2.2 Definisi Real Time Strategy

Real Time Strategy menurut Sindre Berg Stene adalah salah satu genre dari game strategi dimana pemain memiliki kendali penuh terhadapat militer, aset, ekonomi, dan memanipulasi permainan untuk memperoleh tujuan yang akan dicapai.

2.3 Definisi Turn Based Strategy

Turn Based Strategy menurut Sindre Berg Stene [**5**] adalah salah satu genre dari game strategi dimana pemain memiliki kendali pada saat giliran tertentu. Pemain melakukan manipulasi permainan secara bergantian , dari manipulasi militer, manajemen ekonomi , meningkatkan aset untuk memperoleh tujuan yang akan dicapai.

2.4 Manajemen Sumber Daya

Manajemen menurut Dion Bak Christensen[**1**] adalah proses mengatur bagaimana mendapatkan sumber daya , mengatur bagaimana menambah wilayah kekuasaan, mengatur bagaimana mendapatkan informasi lawan, mengatur membangun bangunan dan pasukan. Pemain pada *game* strategi membutuhkan sumber daya dan bagaimana caranya memperolah mengolah dan memanfaatkan sumber daya secara tepat. Sehingga sumber daya yang diperoleh tidak ada sumber daya yg tidak dipakai sama sekali. Oleh karena itu *game* strategi tidak lepas terhadap manajemen sumber daya.

2.5 Data Envelopment Analysis

Data Envelopment Analysis (DEA) adalah layanan manajemen dan teknik benchmarking awalnya dikembangkan oleh *Charnes, Cooper* dan *Rhodes* (1978) untuk mengevaluasi nirlaba dan masyarakat organisasi sektor. *DEA* telah terbukti untuk menemukan cara untuk meningkatkan pelayanan tidak terlihat dengan teknik lain. Namun ada anomali sekitarnya metodologi pengembangan ini. Salah satu yang terbesar bank-bank *AS* yang terletak lebih dari $ 100 juta personil tahunan kelebihan dan biaya operasi, cukup untuk mempengaruhi pendapatan mereka per saham dan ini Penghematan tidak dapat diidentifikasi dengan teknik lain yang digunakan. Sementara bank lain juga telah menyadari keuntungan ditingkatkan melalui inisiatif didorong oleh *DEA*, tidak dapat menemukan lebih dari 10 bank dalam hal ini . Sementara bisnis memiliki kewajiban untuk melaporkan internal metode mereka, *DEA* belum banyak diadopsi oleh bank. *DEA*, sebuah Metode yang dapat menghasilkan jalur baru untuk keuntungan meningkat tidak digunakan ketika teknik kurang kuat lainnya terus digunakan.

DEA membandingkan unit layanan mempertimbangkan semua sumber daya yang digunakan dan layanan yang diberikan, dan mengidentifikasi unit yang paling efisien atau unit praktik terbaik (cabang, departemen, individu) dan unit efisiensi di mana efisiensi nyata perbaikan yang mungkin. Hal ini dicapai dengan membandingkan campuran dan volume pelayanan dan sumber daya digunakan

oleh masing-masing unit dibandingkan dengan orang-orang dari semua lainnya unit. Singkatnya, DEA adalah pembandingan yang sangat kuat teknik.

Berdasarkan metode efisiensi pada buku [**6**] adalah efisensi adalah suatu nilai rasio output dengan input . Seberapa banyak output perunit dari input dicerminkan seberapa besar efisiensi yang di selaraskan. Perhitungan efisiensi manajemen sumber daya menggunakan metode perhitungan efisiensi *Data Envelopment Analys (DEA).* Persamaan dalam perhitungan efisiensi manajemen sumber daya dapat ditulis sebagai berikut :

$$\eta = \frac{Q_0}{Q_{in}} \text{ x } 100\% \quad \text{.....(2.1)}$$

Keterangan :

η : Efisiensi

Q_0 : Pengeluaran

Q_{in} : Pemasukan

Pada perhitungan menurut *DEA* untuk nilai yang memiliki banyak perameter input dan ouput sebagai berikut :

$$\eta = \sum_n \frac{Q_0}{Q_{in}} \text{ x } 100\% \quad \text{.....(2.2)}$$

Keterangan :

$\sum_n$: Jumlah Parameter

2.6 LISP Programmer Language

Lisp menurut Dana Nau [**7**] ,sejarah *lisp* adalah bahasa pemrograman kedua yang masih digunakan secara meluas (bahasa pertama adalah *Fortran) Lisp* di develop oleh John McCarthy pada tahun 1950an sebagai *tool* program pembuktian kebenaran. Selama beberapa tahun, *Lisp* pada umumnya digunakan oleh kegiatan akademik. Di awal tahun 1970an, *lisp* bekerja dalam *High-powered personal worksations specialize* pada *M.I.T* dan *Xerox Palo Alto Research Center*. Penelitian tersebut menghasilkan dalam mesin *lisp* tersedia secara komersil. *Lisp* adalah bentuk sintaks sederhana, *interpreter* dan berkemampuan menghasilkan struktur data baru pada saat dijalankan dengan didorong kemampuan pengembangan lingkungan *programming* yang *powerful* yang bisa menyediakan bahasa apa saja. Berikut keunggulan *lisp* diantaranya :

a. *Recursion*
   Progarm bisa memanggil diri sendiri sebagai *subroutine.*
b. *Garbage collection*
   Penyimpanan data secara otomatis di dalur ulang
c. *Uniform representation*
   Progam dan data terlihat sama sehingga program dapat memerika programlainnya, program dapat menuliskan program,program dapat memodifikasi sendiri (*learning),* dan struktur datanya bisa mengadung suatu program.
d. *Interaction*
   Pengguna dapat menggabungkan penulisan program, kompilasi, pengujian , *debugging*, menjalankan pada sesi interaktif tunggal.

Berikut penggunaan aplikasi yang menggunakan lisp sebagai berikut :

a. Artificial Intelligence
b. Symbolic Algebraic Manipulation
c. Natural Language Understanding
d. Machine Translation

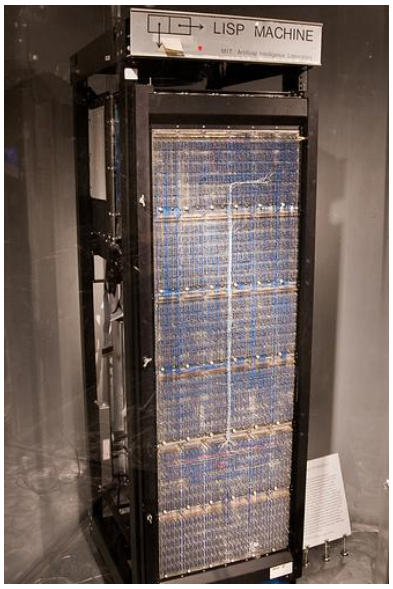
e. Formal Logical Reasoning
f. Expert System (Diagnosis, Identification, Design)
g. Automatic Programming
h. Robotics
i. Perception (Vision, Speech understanding)

Pada *Lisp* , semua komputasi di lakukan oleh pmanggilan fungsi. Suatu fungsi dipanggil dan ditulis seperti urutan, thean fungsi nama pertama .Perbedaan sintaks dari bahasa *Fortran* dan *Lisp* sebagai berikut :

FORTRAN :	LISP :
SQRT(2.0)	(SQRT 2.0)
X = 2	(SETQ X 2)
X= 2 + 2	(SETQ X (+ 2 2))
PRINT 1, X	(PRINT X)
IF(X .GT. Y) Y=3	(COND ((> X Y) (SETQ Y 3)))
GO TO 10	(GO 10)

}

Gambar 2. 1 Gambar Perbedaan Sintaks *FORTRAN* dengan *LISP*



Gambar 2. 2 Gambar Mesin *LISP*



Gambar 2. 3 Mesin *FORTRAN* IBM

2.7 Automated Planning

Planning berdasarkan buku *SHOP : Simple Hierarchical Ordered Planner* [**8**] adalah sisi pemikiran dari suatu aksi. Berbentuk abstrak, proses pertimbangan eksplisit yang memilih dan mengatur aksi dengan mengantipasi hasil yang di duga. Tujuan pertimbangan untuk memperoleh kemungkinan tujuan yang terbaik. Automated planning berada di wilayah *Artifcial Intelligence (AI)* yang mempelajari proses pertimbangan secara komputasi.

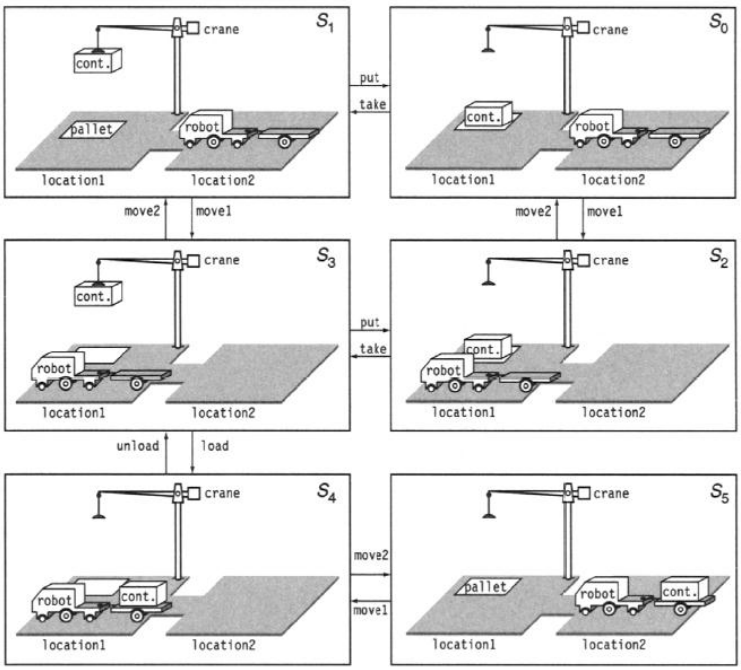
Dikarenakan banyaknya tipe aksi, ada variasi bentuk dari *planning.* Contohnya *path and motion planning, preception planning dan information gathering,navigation planning,manipulation planning ,communication planning* , dan bentuk *social dan economic planning.*

Konsep model dalam mendeskripsikan elemen utama pada masalah. Hal tersebut dapat dipecah secara signifikan dengan pendekatan komputasi dan pendekatan algoritma untuk memecahkan masalah. Namun, hal tersebut sangat berguna untuk menjelaskan konsep dasar untuk mengklarifikasi batasan asumpsi dalam menganalisis kebutuhan representasi dan nilai tukar untuk meningkatkan properti semantik.

Semenjak *planning* menitikberatkan pada pemilihan dan mengatur aksi dalam pergantian *state* pada sistem, konsep model untuk *planning* membutuhkan model umum untuk sistem yang dinamis. Kebanyakan pendekatan *planning* pada buku [**8**] menggunakan mode *state – transition system* secara formal sebagai berikut :

S = {s_1,s_2, . . .} adalah *finite* atau rekursif *enumerable set* dari *state*
A={a_1,a_2, . . .} adalah *finite* atau rekursif *enumerable set* dari aksi
E={e_1,e_1, . . .} adalah *finite* atau rekursif *enumerable set* dari *event*
Y : S x A x E ->2^S adalah fungsi sebuah *state* transisi

Gambar 2. 4 Model *Planning*



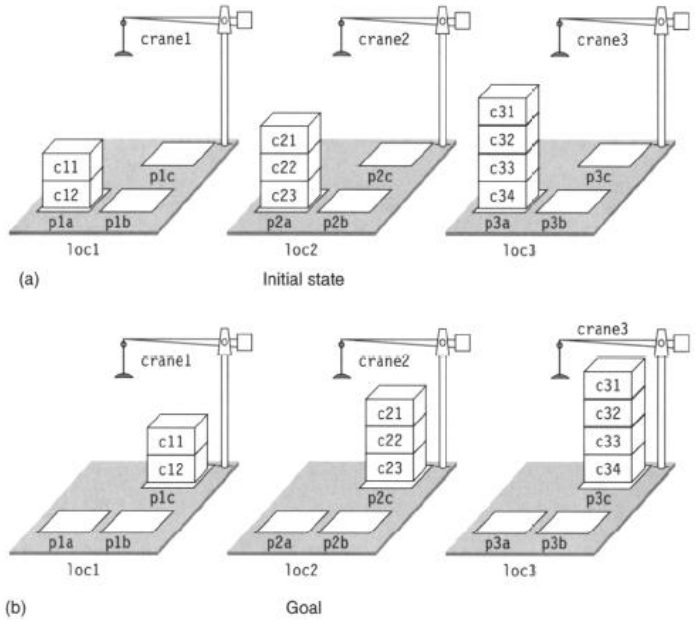
Gambar 2. 5 Gambar transisi *state* untuk *domain* sederhana yang melibatkan mesin derek dan robot dalam mengantarkan barang. [8]

2.8 Hierarchical Task Network Planning

Hierarchical Tas Network Planning menurut buku *SHOP : Simple Hierarchical Ordered Planner* [**8**] seperti *classical planning* yang pada setiap *state*-nya merepresentasi dari *set atom*, dan setiap aksinya menyesuaikan transisi *state* deterministik. Namun, *HTN planners* berbeda dari *classical planners* apa yang mereka tujuan perencanaannya dan bagiamana mereka merencakannya.

Dalam *HTN planners* , secara garis besar bukan untuk mendapatkan hasil dari tujuan tetapi untuk melaksanakan setiap kumpulan *task*-nya. Data masukan pada *planning system* menggunakan operator yang mirip degnan *classical planning* dan juga setiap sekumpulan *method*-nya memecah setiap *task*-nya menjadi *subtask*. *Planning* memproses dengan memecah *non-primitive task* secara rekursif menjadi bagian *task* yang lebih kecil. Setiap *method* merupakan bagian dari spesifikasi cabang fungsi.

HTN planning banyak digunakan lebih luas untuk hal palikasi praktis dari pada *planning* lain dalam buku *SHOP : Simple Hierarchical Ordered Planner* [**8**]. Hal tersebut dikarenakan *method* menyediakan jalan mudah untuk menuliskan "resep" pemecahan masalah menyesuaikan bagaiman manusia bekerja tentang menyelesaikan masalah *planning*. Berikut adalah contoh penerapan *HTN planners* sebagai berikut :



Gambar 2. 6 Gambar *DWR Problem* dengan beberapa *stack* kontener yang harus dipindahkan ke lokasi berbeda dengan *HTN Planners* [8]

2.9 Algoritma *SHOP*

SHOP berdasarkan buku *SHOP : Simple Hierarchical Ordered Planner* [**8**] , Dana Nau [**9**] [**10**] merupakan algoritma *planning* yang saling terhubung secara bertingkat yang lengkap dan saling berseberangan pada setiap wilayah *task*-nya. *task* yang berisikan tugas – tugas yang di urutan yang sama nantinya akan di jalankan. SHOP berdasarkan Dana Nau [**2**] mempunyai keunggulan pada pengetahuan dasarnya. Seperti , SHOP dapat melakukan logika komputasi kompleks.

SHOP (*Simple Hierarchical Ordered Planner*) memiliki wilayah yang generalisasi dari teknik penjadwalan digunakan pada beberapa sistem wilayah khusus. Algoritma *SHOP* termasuk *HTN* (*Hierarchical Task Network*) , tetapi sedikit berbda dari *HTN*. Algoritma ini membuat jadwal dengan mengdekomposisi secara rekursif dari task menjadi bagian task – task kecil atau sub task. Gambaran bagaimana *SHOP* bekerja sebagai berikut :

```
procedure SHOP (S, T, D)
1.  if T = nil then return nil endif
2.  t = the first task in T
3.  U = the remaining tasks in T
4.  if t is primitive (i.e., there is an operator for t) then
5.      nondeterministically choose an operator o for t
6.      P = SHOP (o(S), U, D)
7.      if P = FAIL then return FAIL endif
8.      return cons(p, P)
9.  else if there is a method applicable to t whose
        preconditions can all be inferred from S then
10.     nondeterministically let m be such a method
11.     return SHOP(S, append(m(t,S), U), D)
12. else
13.     return FAIL
14. endif
end SHOP
```

Gambar 2. 7 Algoritma SHOP [2]

Keterangan *input* dan *output* pada *SHOP* sebagai berikut :

1. ***Input***
   a. *State*

      State adalah kumpulan pernyataan awal dimana pernyataan ini akan dimanipulasikan kedalam *task* yang akan dieksekusi. Contoh : Uang Rp.1000

   b. *Task List*

      Task List adalah sekumpulan task secara berurutan yang akan dieksekusi.

c. *Domain*

Domain adalah berisikan axiom ,method, operator ,berupa task yang dibagi – bagi , prekondisi dan pemecahan dari task sebelumnya.

2. Output

Output yang dihasilkan bisa mendapatkan jalur aksi yang dipilih berdasakan jalur pertama yang diambil, semua kemungkinan jalur yang dipilih, jalur yang memerlukan biaya lebih sedikit, jalur yang memerlukan semua biaya paling sedikit dan lainnya.

Meskipun *domain* diatas cukup sederhana, kebanyakan *AI planning system* yang lain tidak memerlukan daya ekspresif yang cukup untuk sepenuhnya mewakili, karena komputasi numerik yang dibutuhkan merupakan bagian proses *planning*. Berikut domain yang diperluas sebagai berikut :

Tabel 3. 1 Tabel Perluasan *Domain SHOP* [9]

Axiom Domain	Maksud sintaks

A_1 = `(:- (have-taxi-fare ?dist)`
`((have-cash ?m)`
`(eval (>= ?m (+ 1.5 ?dist)))))`
	Agar mendapatkan uang yang cukup untuk Taxi, Membutuhkan paling sedikit \$1.5 + \$1 untuk setiap mil jarak yang akan ditempuh.

A_2 = `(:- (walking-distance ?u ?v)`
`((weather-is 'good)`
`(distance ?u ?v ?w)`
`(eval (<= ?w 3)))`
`((distance ?u ?v ?w)`
`(eval (<= ?w 0.5))))`
	Menambil jalan kaki jika cuaca bagus dan jarak $\leq$ 3 mil atau cuaca buruk dan jarak $\leq$ ½ mil.

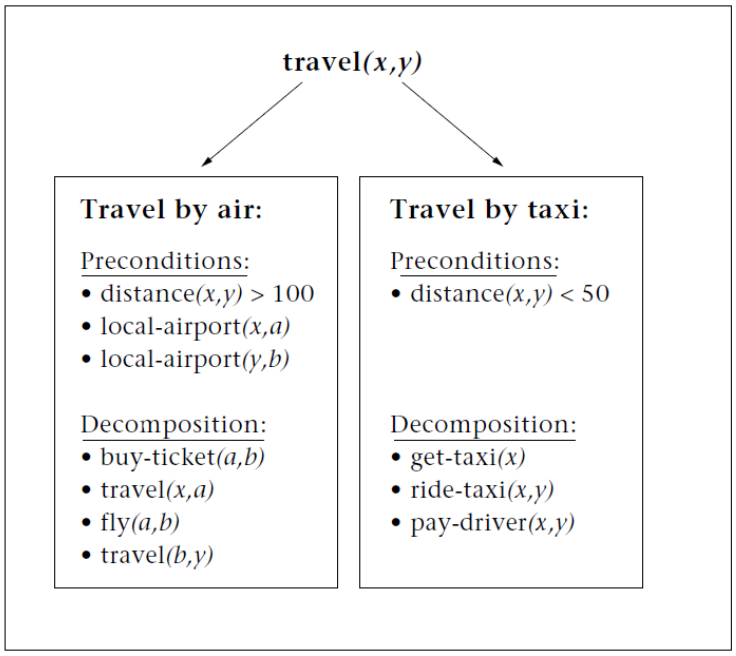
M_1 = `(:method (pay-driver ?fare)`
`((have-cash ?m)`
`(eval (>= ?m ?fare)))`
`` `((!set-cash ?m ,(- ?m ?fare)))) ``
	Jika memiliki uang yang cukup untuk mebayar taxi, kemudia uang di tangan dikurangi dengan biaya taxi.

M_2 = (:method (travel-to ?q) ((at ?p) (walking-distance ?p ?q)) '((!walk ?p ?q)))	Jika q adalah jarak jalan kaki. Kemudian perjalanan langsung saja ke lokasi secara langsung.
M_3 = (:method (travel-to ?y) (:first (at ?x) (at-taxi-stand ?t ?x) (distance ?x ?y ?d) (have-taxi-fare ?d)) `((!hail ?t ?x) (!ride ?t ?x ?y) (pay-driver ,(+ 1.50 ?d))) ((at ?x) (bus-route ?bus ?x ?y)) '((!wait-for ?bus ?x) (pay-driver 1.00) (!ride ?bus ?x ?y)))	Jika kita berada di tempat taksi dan punya cukup uang untuk membayar ongkos taksi, maka kita dapat melakukan perjalanan ke y oleh memanggil taksi pertama di taksi, naik di dalamnya untuk y, dan membayar sopir tarif yang diperlukan. Jika tidak, jika kita berada di rute bus, maka bisa berjalan ke y dengan menunggu bus, membayar sopir $ 1, dan naik bus ke y.
O_1 = (:operator (!hail ?vehicle ?location) () ((at ?vehicle ?location)))	Sintaks ini adalah operator untuk memanggil kendaraan. Sintaks tersebut membawa kendaraan ke lokasi kita saat ini
O_2 = (:operator (!wait-for ?bus ?location) () ((at ?bus ?location)))	Sintaks ini adalah operator untuk menunggu bus. Sintaks ini membawa bus ke lokasi saat ini
O_3 = (:operator (!ride ?vehicle ?a ?b) ((at ?a) (at ?vehicle ?a)) ((at ?b) (at ?vehicle ?b)))	Sintaks ini adalah operator untuk naik kendaraan ke lokasi.Sintaks ini menempatkan kedua kami dan kendaraan pada saat di lokasi.

O_4 = `(:operator (!set-cash ?old ?new)`
`((have-cash ?old))`
`((have-cash ?new)))`
	Sintaks ini adalah operator untuk mengubah berapa banyak uang tunai sisanya.

O_5 = `(:operator (!walk ?here ?there)`
`((at ?here))`
`((at ?there)))`
	Sintaks ini adalah operator untuk berjalan ke lokasi. Sintaks ini menempatkan posisi kita di lokasi itu.

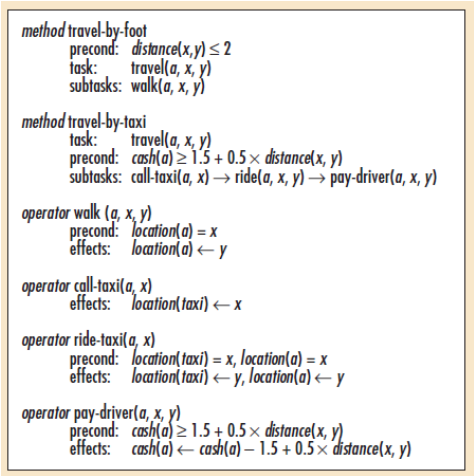
SHOP bekerja sebagai contoh pada masalah pemilihan perjalanan ke suatu tempat memiliki dua metode cara perjalanan , dengan pesawat atau taxi. Gambaran lebih jelas sebagai berikut :



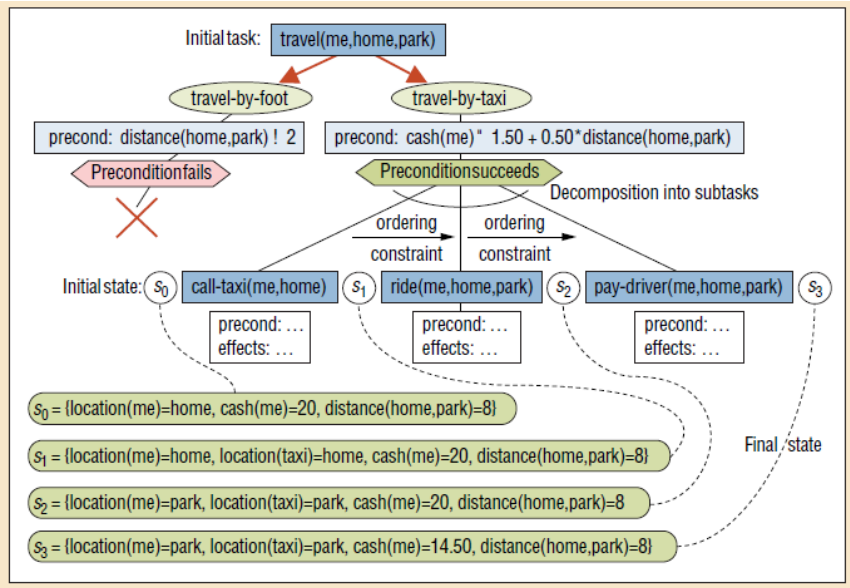
Gambar 2. 8 Contoh Penerapan Algoritma [2]

Berdasarkan gambar tersebut *planner* untuk travel terdiri dari task – task yaitu , menggunakan pesawat udara , atau menggunakan taxi. Setiap *task* yang telah tersedia dibreak down kembali menjadio sub task pada setiap tasknya. Untuk Task travel by Air / Pesawat harus melalui kondisi jika jarak tempuh antara kota x dengna kota y lebih dari 100 KM kemudian task ini di eksekusi , variabel a dan b

menunjukan posisi area yang ditempuh. melakukan buy-ticket(a,b) , kemudian travel (x,a) perjalanan keberangkatan / departure, setelah itu fly(a,b) keadaan diperjalanan, kemudian sampai ditujuan travel(b,y). Sedangkan perjalanan menggunakan taksi dilakukan jika jarak tempat x dengan y kurang dari 50KM. gambaran lebih jelas sebagai pada contoh kasus travel menggunakan jalan kaki atau taxi berikut :



Gambar 2. 9 Perluasan Domain Pada *SHOP* [11]



Gambar 2. 10 Proses kerja SHOP [11]

2.10 OOP(Object Oriented Programming)

OOP adalah teknik memodelkan suatu sistem dunia nyata dalam perangkat lunak berdasarkan objek. Objek tersebut adalah konsep inti. Suatu objek adalah sebuah perangkat lunak entitas atau konsep model dunia nyata. Ketika sebuah program berjalan, objek individu biasanya tidak dapat berdiri sendiri. Mereka termasuk ke sebuah koleksi dari objek lain yang serupa yang merupakan anggota dari grup yang sama, atau class. Sebuah program akan terbentuk dari banyak class yang berbeda, setiap class terbentuk dari objek-objek yang serupa.

Beberapa sistem perangkat lunak orientasi objek akan memiliki sifat berikut:

1. Abstraksi dengan objek

Abstraksi adalah mekanisme yang memungkinkan kompleks, situasi dunia nyata dapat diwakili menggunakan model yang disederhanakan. Orientasi objek abstrak dunia nyata didasarkan pada objek dan interkasi antar objek lainnya.

2. Enkapsulasi class

Enkapsulasi adalah proses menyembunyikan semua bagian rinci sebuah objek dari dunia luar.

3. Interaksi lewat pesan.

Untuk memenuhi suatu perintah, objek butuh berinteraksi dengan objek lain. Interaksi dapat antara objek di class yang sama, atau objek di class lain. Interaksi ini ditangani dengan cara mengirimkan pesan ke objek lain untuk melewati informasi atau meminta aksi.

4. Masa hidup objek

Semua objek memiliki masa hidup. Mereka dibuat dan diinisialisasi sebagaimana mereka dibutuhkan pada saat program dijalankan, hidup dan membawa keluar fungsi mereka, dan akhirnya dihancurkan. Sementara mereka

ada, mereka mempertahankan identitas mereka sendiri dan kondisi. Banyak objek yaitu contoh dari class yang sama bisa hidup pada waktu tertentu. Setiap objek memiliki atribut yang berbeda dari yang lain yaitu contoh objek dari class yang sama.

5. Hirarki class

Dalam desain orientasi objek, class objek tersebut diatur ke dalam hirarki yang memodelkan dan menggambarkan hubungan antar class. Hubungan yang paling sederhana adalah sebuah asosiasi.

6. Polimorfisme

Polimorfisme adalah karakteristik sistem orientasi objek. Ketika warisan digunakan untuk memperpanjang class umum ke class yang lebih khusus, biasanya akan mencakup memperluas beberapa perilaku dari class umum. Class khusus sering akan menerapkan perilaku yang agak berbeda dengan class umum, tetapi nama yang digunakan mendefinisikan perilaku akan menjadi sama. Adalah penting bahwa contoh yang diberikan dari sebuah objek menggunakan perilaku yang benar, dan property polimorfisme memungkinkan ini terjadi secara otomatis dan mulus. Polimorfisme sebenarnya lebih mudah digunakan daripada dijelaskan.

2.11 UML (Unified Modeling Language)

UML adalah penerus gelombang berorientasi objek alnalisis dan desain (OOA & D) metode yang muncul di akhir ,,80-an dan ‟90-an awal. Hal yang paling langsung menyatukan metode Booch, Rumbaugh (OMT), dan Jacobson, tapi jangkauan lebih luas dari itu. UML pergi melalui proses standarisasi dengan OMG (Obyek Manajement Group) dan sekarang menjadi standar OMG.

UML disebut bahasa permodelan, bukan metode. Sebagian besar metode terdiri, setidaknya pada prinsipnya, kedua model sebuah bahasa dan proses. Bahasa pemodelan adalah notasi (terutama grafis) bahwa metode digunakan untukmengekspresikan desain. Proses ini menyarankan mereka apa langkah yang harus diambil dalam melakukan desain.

Bagian proses dalam buku banyak metode yang agak samar. Selain itu,

Kebanyakan orang, ketika mereka mengatakan bahwa mereka menggunakan metode, menggunakan bahasa pemodelan, tapi jarang mengikuti proses. Jadi dalam banyak hal pemodelan bahasa adalah bagian paling penting dari meode ini. Hal ini tentu bagian penting untuk komunikasi. Jika anda ingin mendiskusikandesain anda dengan seseorang, itu adalah bahasa pemodelan yang anda berdua perlu pahami, bukan proses yang digunakan untuk sampai ke desain tersebut.

2.12 Diagram *UML*

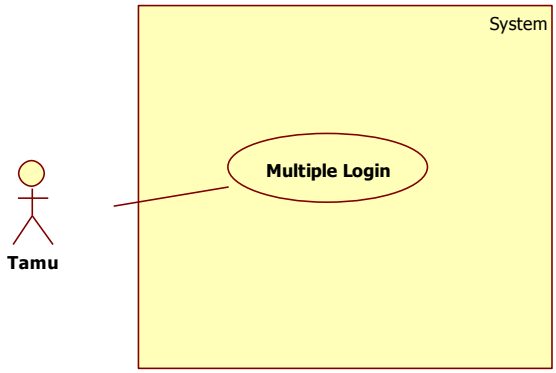
Menggunakan berbagai macam diagram dengan fungsi masing-masing untuk menggambarkan setiap proses dari sistem berorientasi objek. Berikut merupakan beberapa diagram UML diantaranya:

2.11.1 Use Case Diagram

Use Case atau diagram *use case* merupakan pemodelan yang digunakan untuk menggambarkan kelakuan (*behavior*) dari sistem yang akan dibuat. Use case mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem yang akan dibuat. Secara kasar, use case digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut.

Syarat penamaan pada use case adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada use case yaitu pendefinisian apa yang disebut aktor dan use case.

1. Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem yang akan dibuat diluar sistem yang akan dibuat itu sendiri, jadi walaupun simbol dari actor adalah gambar orang, tapi actor belum tentu merupakan orang.
2. Use case merupakan fungsionalitas yang disediakan sistem sebagai unit - unit yang saling bertukar pesan antarunit atau aktor



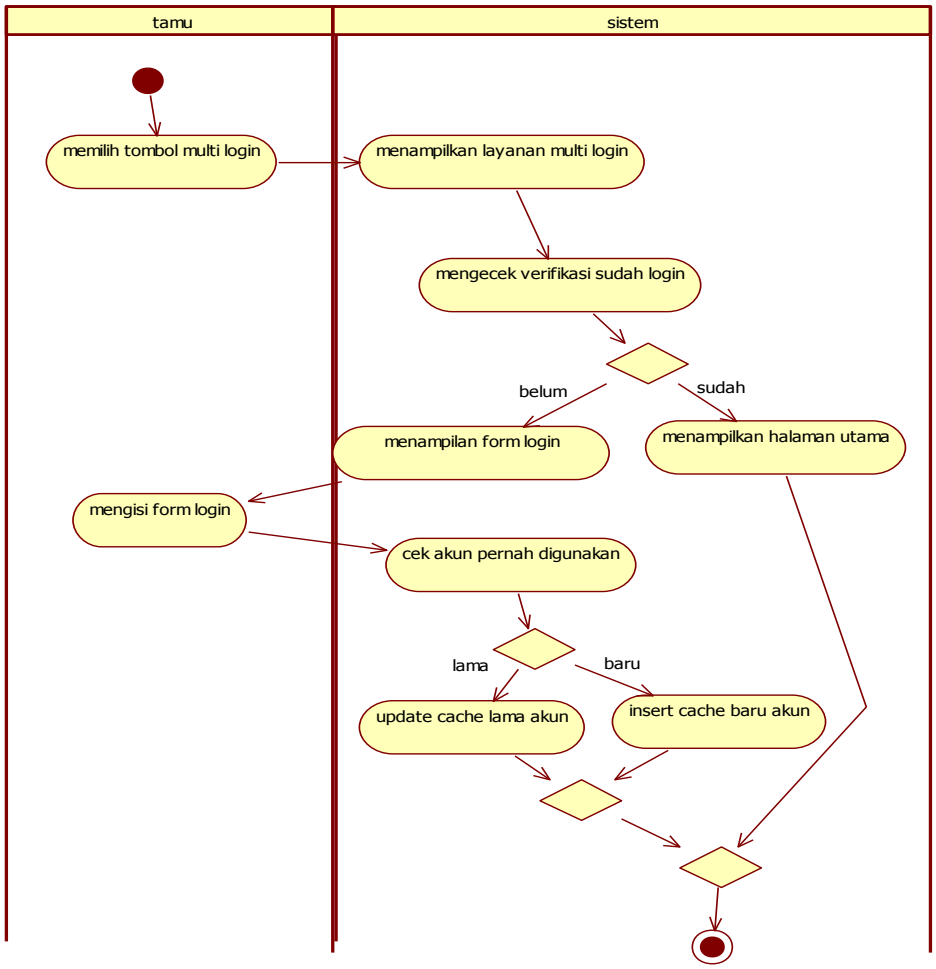
Gambar 2. 11 Contoh *Usecase Diagram*

2.11.2 Activity Diagram

Diagram aktivitas atau activity diagram adalah sebuah diagram yang menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Dalam diagram aktivitas yang perlu diperhatikan adalah bahwa diagram aktivitas menggambarkan aktivitas sistem, bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem.

Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut:

1. Rancangan proses bisnis di mana setiap urutan aktivitas yang digambarkan merupakan proses bisnis system yang didefinisikan.
2. Urutan atau pengelompokan tampilan dari sistem/user interface di mana setiap aktivitas dianggap memiliki sebuah rancangan antaruka tampilan.
3. Rancangan pengunjian di mana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.

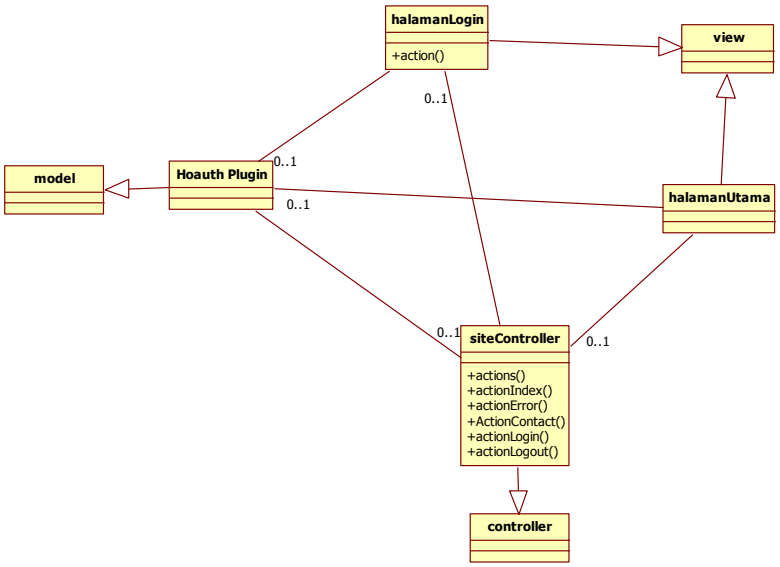


Gambar 2. 12 Contoh *Activity Diagram*

2.11.3 Class Diagram

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi.

1. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas
2. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas

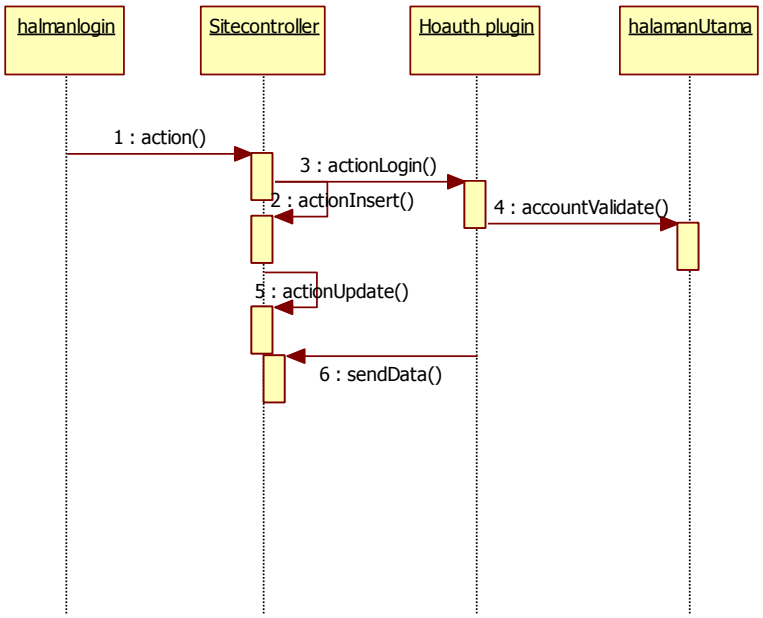


Gambar 2. 13 Contoh *Class Diagram*

2.11.4 Sequence Diagram

Diagram *sequence* adalah diagram yang menggambarkan kelakuan objek pada use case dengan mendeskripsikan waktu hidup objek dan message yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah use case beserta metode-metode yang dimiliki kelas yang substansiasi menjadi objek itu.

Banyaknya diagram sekuen yang harus digambarkan adalah sebanyak pendefinisian *use case* yang memiliki prose situ sendiri atau yang penting semua use case yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram sekuen sehingga semakin banyak use case yang didefinisikan maka diagram sekuen yang harus dibuat juga semakin banyak.



Gambar 2. 14 Contoh *Sequence Diagram*