PERANCANGAN SIMULASI BERORIENTASI OBJEK PADA MODEL ANTRIAN DENGAN MENGGUNAKAN JAVA DAN DSOL

SKRIPSI

HERI HERYADI
04 04 070344

UNIVERSITAS INDONESIA
FAKULTAS TEKNIK
DEPARTEMEN TEKNIK INDUSTRI
DEPOK
JULI 2008

PERANCANGAN SIMULASI BERORIENTASI OBJEK PADA MODEL ANTRIAN DENGAN MENGGUNAKAN JAVA DAN DSOL

SKRIPSI

Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik

HERI HERYADI
04 04 070344

DEPARTEMEN TEKNIK INDUSTRI
FAKULTAS TEKNIK
UNIVERSITAS INDONESIA
DEPOK
JULI 2008

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama	**: Heri Heryadi**
NPM	**: 04 04 070344**
Tanda Tangan	**:**
Tanggal	**: 10 Juli 2008**

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh	:
Nama	: Heri Heryadi
NPM	: 0404070344
Program Studi	: Teknik Industri
Judul Skripsi	: Perancangan Simulasi Berorientasi Objek Pada Model Antrian Dengan Menggunakan Java Dan DSOL

Telah berhasil dipertahankan di hadapan Dewan Penguji dan di terima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana pada Program Teknik Industri Fakultas Teknik Universitas Indonesia

DEWAN PENGUJI

Pembimbing	: Armand Omar Moeis, ST., MSc	(.....)
Penguji	: Dr. Ir. T. Yuri M. Zagloel, MengSC	(.....)
Penguji	: Ir. Boy Nurtjahyo M, MSIE	(.....)

Ditetapkan di : Depok
Tanggal : 10 Juli 2008

LEMBAR PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama : Heri Heryadi
NPM : 0404070344
Program Studi : Teknik Industri
Departemen : Teknik Industri
Fakultas : Teknik
Jenis karya : Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Non- Eksklusif (*Non-exclusiveRoyalty-Free Right*)** atas karya ilmiah saya yang berjudul :

Perancangan Simulasi Berorientasi Objek Pada Model Antrian Dengan Menggunakan Java Dan DSOL

beserta perangkat yang ada (bila diperlukan). Dengan Hak Bebas Royalti Non-Ekslusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/format-kan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan mempublikasikan tugas akhir saya tanpa meminta ijin dari saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 10 Juli 2008
Yang menyatakan

(Heri Heryadi)

UCAPAN TERIMAKASIH

Segala puji hanya milik Allah dan Shalawat serta Salam semoga tercurah kepada Nabi Muhammad saw. Penulis menyadari bahwa tanpa bimbingan, bantuan, motivasi, inspirasi, dan sumbangan pikiran dari berbagai pihak, sejak awal masa perkuliahan hingga penyusunan skripsi maka skripsi ini tidak akan dapat selesai. Oleh sebab itu, pada kesempatan ini penulis ingin mengungkapkan rasa terima kasih sebesar-besarnya kepada beberapa pihak:

1. Mamah dan Bapak yang telah mendoakan dan memberikan dukungan kepada penulis
2. Roy Elfania, Usep Setiadi, dan Sindi Chintia Dewi atas doa dan semangatnya
3. Bapak Armand Omar Moeis, ST., MSc selaku dosen pembimbing skripsi yang telah banyak memberikan bantuan moral dan material kepada penulis
4. Bapak Ir. Akhmad Hidayatno., MBT selaku dosen pembimbing akademis
5. Alin Priastika (Unsho, Shoun, Ling, SHK), sumber inspirasi sepanjang masa
6. Anggota "Jomblo Corner" (Aqquinaldo, Rian, Eko, dan Arie) yang selalu saling mengisi kekosongan
7. Teman-teman Teknik Industri UI 2004 atas kebersamaannya
8. Jalaludin Rumi, Albert Einstein, Thomas Alfa Edison, Emily Dickinson, Edgar Allan Poe, Thomas Carlyle, Bill Gates serta Larry Page dan Sergey Brin selaku orang-orang hebat yang dapat mengubah wajah dunia termasuk diriku
9. Semuanya, terima kasih banyak

Semoga Allah membalas semua kebaikan semuanya. Dan semoga skripsi ini memberikan manfaat bagi dunia ilmu pengetahuan. Amin.

Depok, 2 Juli 2008

Penulis

RIWAYAT HIDUP PENULIS

Nama : Heri Heryadi

Tempat, Tanggal Lahir : Kuningan, 21 November 1984

Alamat : Jl.Ds.Cilaja No.441 Rt.15/04 Kuningan Jawa Barat 45553

Pendidikan:

1.	SD	SD Negeri 1 Cilaja	1991 - 1997
2.	SLTP	SLTP Negeri 1 Kuningan	1997 - 2000
3.	SMU	SMU Negeri 2 Kuningan	2000 - 2003
5.	S-1	Departemen Teknik Industri Universitas Indonesia	2004 - 2008

ABSTRAK

Nama : Heri Heryadi
Program studi : Teknik Industri
Judul : PERANCANGAN SIMULASI BERORIENTASI OBJEK PADA MODEL ANTRIAN DENGAN MENGGUNAKAN JAVA DAN DSOL

Simulasi dengan menggunakan komputer merupakan alat yang penting untuk memodelkan dan menganalisa sistem yang kompleks. Salah satu masalah yang dapat diselesaikan dengan menggunakan simulasi adalah sistem antrian. Pada sistem antrian terdapat bermacam-macam komponen yang menyusunnya sebagai suatu sistem. Sistem antrian dimulai saat suatu entitas datang dari luar sistem memasuki fasilitas untuk meminta pelayanan oleh server dan berakhir ketika entitas ini selesai melakukan apa yang mereka perlukan dan keluar dari sistem.

Metode simulasi berorientasi objek memiliki keunggulan dibandingkan metode simulasi lainnya karena sistem atau permasalahan dapat dipandang tersusun dari objek-objek seperti di dunia nyata yang dapat saling berkomunikasi sehingga permasalahan yang kompleks dapat dipecah-pecah menjadi bagian-bagian kecil yang bisa diselesaikan dan bersifat lebih alami. Secara konseptual, sistem dimodelkan dalam suatu bahasa diagram yaitu diagram UML yang merupakan standar yang sudah sangat diterima di dalam industri perangkat lunak. Dan dalam implementasinya, penelitian ini menggunakan bahasa pemrograman Java serta *library* simulasi DSOL untuk mengembangkan model yang dibuat.

Penelitian ini menerapkan metode simulasi berorientasi objek dengan menggunakan bahasa pemrograman Java dan *library* simulasi DSOL pada model antrian multi server. Penelitian ini juga membandingkannya dengan metode simulasi berorientasi proses yang selama ini sering digunakan.

Kata kunci:
Simulasi, Simulasi Berorientasi Objek, Model Antrian , Java, UML, DSOL

ABSTRACT

Name : Heri Heryadi
Study Program: Industrial Engineering
Title : OBJECT ORIENTED SIMULATION DESIGN IN QUEUING MODEL USING JAVA AND DSOL

Simulation using computer is an important tool in modeling and analysis of complex systems. One of the problems that can be solved using simulation is queuing system. Queuing system is consisted of components as a whole system's building block. Queuing system run when a customer comes as an input of the system asking for server's services and finish when customer released from system.

Object oriented simulation method has advantages than other simulation method because a system or problem can be viewed arranged from objects communicated each other and can be simplify the problem underlying and more natural and intuitive. Conceptually, a system underlying is modeled in graphical language called UML diagram which is a standard graphical notation accepted by many parties especially in software industry.

This study applied object oriented simulation using Java programming language and simulation library called DSOL in order in multi server queuing model. This study also compares object oriented simulation method with process oriented simulation.

Keywords:
Simulation, Object Oriented Simulation, Model, Queuing Model , Java, UML, DSOL

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

1. PENDAHULUAN

1.1 Latar Belakang Permasalahan

Abad dua puluh satu adalah abad informasi. Kemajuan teknlogi di abad tersebut berkembang sangat pesat yang terdiri dari kecepatan pemrosesan data, kapasitas penyimpanan, dan kecepatan transfer data antar alat yang saling berkomunikasi. Kemajuan tersebut sangat berpengaruh luas terhadap banyak bidang kehidupan manusia seperti telekomunikasi, ekonomi, ilmu pengetahuan, dan kedokteran. Bahkan, manusia kini bisa saling berkomunikasi tanpa memandang tempat dan waktu lagi karena mereka sudah saling terhubung melalui internet.

Salah satu aplikasi terkini dari teknologi komputer dan informasi adalah simulasi. Selain harus menguasai domain masalah terkait (seperti produksi, computer, telekomunikasi, dan sistem logistik), aplikasi simulasi juga membutuhkan kompetensi dalam berbagai bidang yang luas seperti analisis sistem, perancangan sistem, metode statistic, perancangan eksperimental, perancangan perangkat lunak, dan pemrograman.

Simulasi dengan menggunakan komputer merupakan alat yang penting untuk memodelkan dan menganalisa sistem yang kompleks. Aplikasinya sangat luas dan beragam. Bahkan, aplikasi simulasi komputer dalam masalah-masalah komersial menawarkan metode yang fleksibel dan dengan tingkat keberhasilan yang tinggi dalam mengeksplorasi dan mengoptimalkan aliran informasi dan material di dalam suatu perusahaan.

Di kalangan industri banyak sekali perangkat lunak simulasi yang menjadi pilihan baik perangkat simulasi diskret maupun kontinu dan dengan pendekatan yang beragam. Namun, kebanyakan aplikasi simulasi yang ada saat ini harga lisensi untuk memakainya sangat mahal dan biasanya menarget korporasi dan kalangan industri.

DSOL di lain pihak merupakan salah satu alternatif perangkat lunak simulasi yang baik karena menawarkan *platform* simulasi yang bebas biaya dan bersifat *open source* di bawah lisensi GNU/GPL sehingga sangat terbuka untuk didistribusikan, dipelajari, dan dikembangkan tanpa harus khawatir terhadap pelanggaran hak cipta. Walaupun bersifat *open source* bukan berarti DSOL tidak mampu menangani isu skalabitas yang hanya cocok untuk kalangan akademis saja. DSOL dapat diaplikasikan untuk memodelkan sistem apa saja termasuk sistem diskret dan sistem kontinu. Selain itu, karena DSOL ditulis dalam bahasa Java, maka DSOL secara alami memiliki keunggulan untuk dapat digunakan di berbagai *platform* sistem operasi sehingga tidak terbatas pada Microsoft Windows saja. Sekali mengembangkan maka akan dapat berjalan di bawah system operasi Windows, Linux, maupun MacOS tanpa harus merubah *source code.*

Penggunaan aplikasi DSOL di Indonesia masih sangat jarang karena DSOL baru diperkenalkan pada Wintersim Conference 2002. Cara pandang dalam memodelkan sistemnya juga masih terbilang baru yaitu bersifat *object oriented* dan menggunakan bahasa pemrograman Java dan diharapkan bahwa dengan cara pandang tersebut model dapat mudah dibuat terutama dalam tahap proses perancangan, dan pengembangan model itu sendiri.

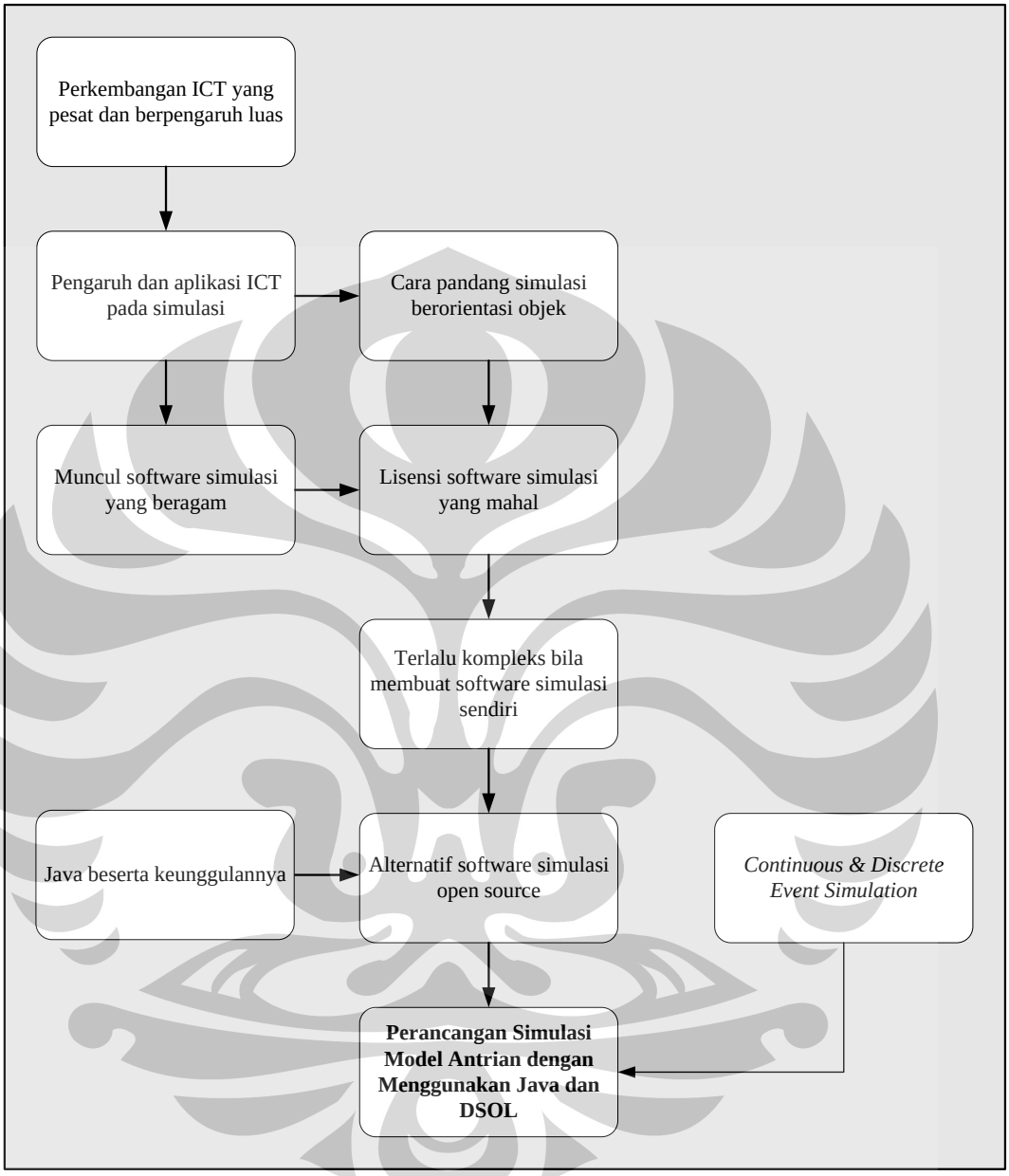
Sementara itu, *Object-oriented simulation* merupakan suatu metode dimana suatu sistem dimodelkan dan diimplementasikan dengan menggunakan *object-oriented design* dan seperangkat alat pemrograman[1]. Dengan menggunakan OOS maka model yang dibuat semakin merepresentasikan kondisi nyata yang ada karena system dipandang tersusun dari objek seperti dunia sebenarnya. Adapun keuntungan menggunakan *object-oriented simulation* adalah[2]:

- Mempercepat proses pengembangan
- Meningkatkan kualitas
- Mempermudah *maintenance*
- Mempermudah modifikasi
- Mempermudah penggunaan kembali *software* dan desain

[1] Chell A. Roberts dan Yasser M. Dessouky, "An Overview of Object-Oriented Simulation", Simulation Councils, 1998, hal.360.
[2] *Ibid.,*

- Mengurangi resiko pengembangan

1.2 Diagram Keterkaitan Masalah



Gambar 1.1 Diagram Keterkaitan Masalah

1.3 Perumusan Masalah

Sesuai dengan latar belakang tersebut maka pokok permasalahan yang akan dibahas pada penelitian ini adalah perlunya membuat suatu model simulasi berdasarkan cara pandang *object oriented*. Adapun model yang digunakan dalam peneilitan ini adalah model sistem antrian multi server. Sistem tersebut terdiri dari banyak *server* yang menerima *customer* yang datang secara saling bebas dan terdistribusi identik. Apabila c*ustomer* yang datang menemukan *server* dalam keadaan *idle* maka akan langsung dilayani. Namun apabila *customer* yang datang menemukan *server* dalam keadaan sibuk, maka *customer* tersebut akan memasuki antrian, di mana jumlah antrian sama dengan jumlah *server* yang tersedia. *Customer* akan dilayani berdasarkan metode *first-in, first-out* (FIFO), di mana *customer* yang masuk pertama akan pertama dilayani. Simulasi dimulai pada saat sistem dalam kondisi kosong dan *idle*. Simulasi dimulai saat *n* buah *customer* memasuki sistem hingga selesai melakukan apa yang mereka perlukan.

1.4. Tujuan Penelitian

1. Memperoleh hasil simulasi dengan menggunakan DSOL dan Java serta memakai pendekatan *object oriented*
2. Memperoleh perbandingan metodologi antara *object-oriented simulation* dengan *flow-based simulation*
3. Memperoleh pedoman mengenai langkah-langkah penggunaan metodologi *object-oriented simulation* dalam membuat model

1.5. Ruang Lingkup Penelitian

Untuk memfokuskan penelitian yang dilakukan, maka diberikan beberapa batasan masalah pada penelitian ini, yaitu:

1. Dalam penelitian ini model dibuat dengan software DSOL dan memakai bahasa pemrograman Java
2. Perancangan model yang dilakukan hanya pada model sederhana karena fokus penelitian ini bukan berdasarkan pada kompleksitas model yang dibuat, namun pada aplikasi *software* DSOL serta pemakaian metode *object*

orientation. Adapun model yang digunakan pada penelitian ini adalah model *multi server queuing system*.

1.6. Metodologi Penelitian

1. Memilih topik dan menentukan tujuan penelitian

   Pada tahap ini dilakukan penentuan masalah yang akan dijadikan topik dalam penelitian serta tujuan dilakukannya penelitian bersama-sama dengan dosen pembimbing
2. Menentukan dan mempelajari dasar teori

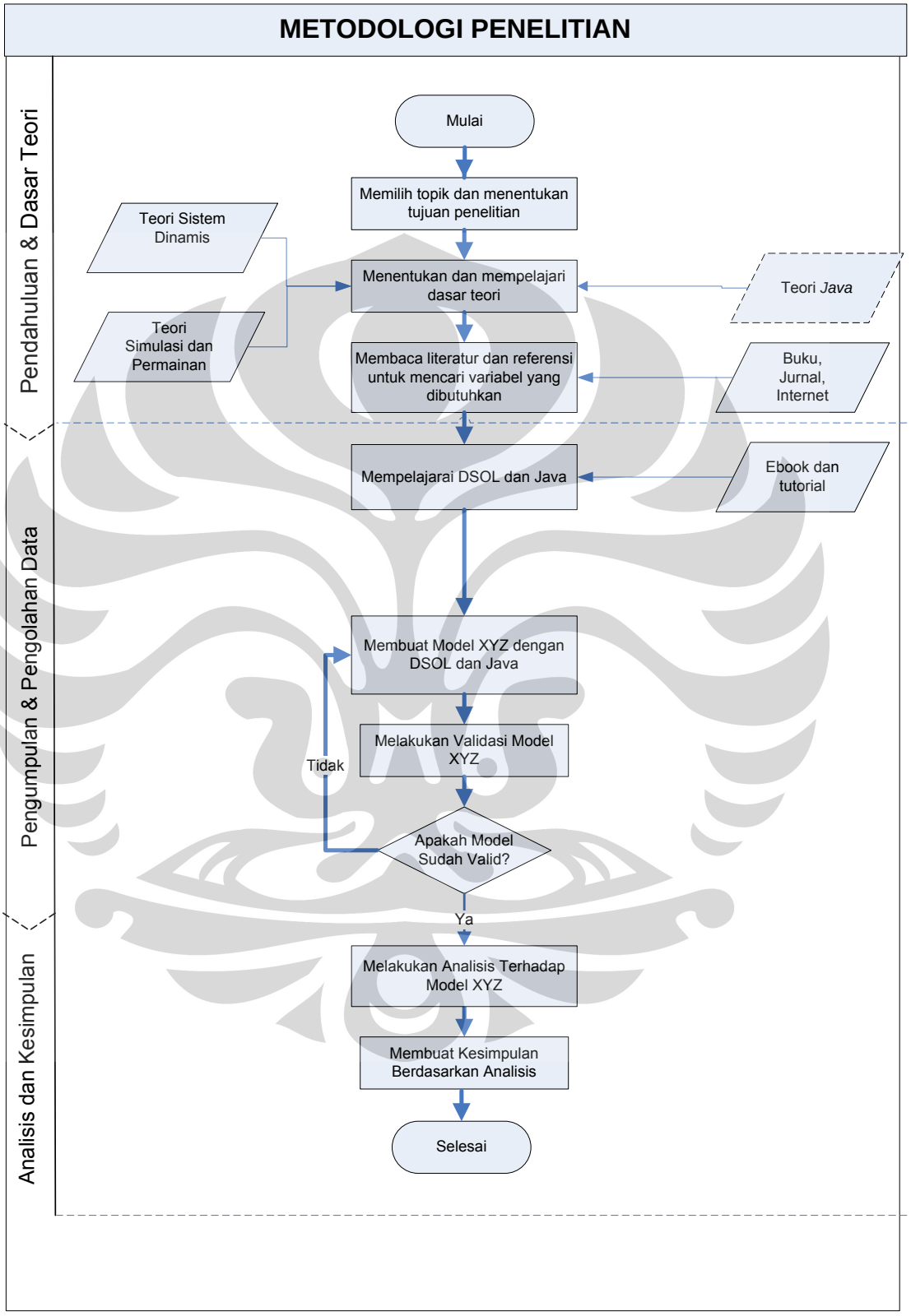
   Pada tahap ini penulis menentukan dasar teori yang dibutuhkan dalam perancangan permainan simulasi berorientasi objek sebagai acuan dalam pelaksanaan tugas akhir. Dasar teori yang dijadikan acuan adalah Sistem Dinamis, Simulasi dan Permainan, DSOL, Java, dan UML
3. Membaca literatur dan referensi untuk mencari variabel yang dibutuhkan

   Tahap selanjutnya adalah mencari literatur dan referensi dari internet untuk mempelajari sistem pembuatan simulasi sistem dinamis berorientasi objek
4. Mempelajarai DSOL dan Java

   Pada tahap ini penulis mempelajari lebih jauh mengenai Java dan DSOL serta implementasinya pada model simulasi
5. Membuat Model dengan DSOL dan Java

   Pada tahap ini dilakukan konstruksi model antrian dengan menggunakan DSOL
6. Melakukan Validasi Model

   Pada tahap ini model simulasi antrian yang telah dibuat mulai dijalankan untuk melakukan perbaikan yang diperlukan sehingga diperoleh simulasi yang benar
7. Melakukan analisis terhadap model antrian dan mendiskusikannya dengan pembimbing skripsi
8. Membuat kesimpulan penelitian berdasarkan hasil analisis



Gambar 1.2 Diagram Alir Penelitian

1.7. Sistematika Penulisan

Pembahasan mengenai penelitian yang diketengahkan oleh peneliti disajikan dalam lima bab. Bab 1 merupakan bab pendahuluan yang akan memberikan penjelasan mengenai latar belakang penelitian, diagram keterkaitan masalah, perumusan permasalahan, tujuan penelitian, batasan masalah, metodologi penelitian, dan sistematika penulisan.

Bab 2 menjelaskan secara terperinci mengenai teori dan konsep yang relevan dengan masalah yang telah dirumuskan untuk mencari pemecahan atas masalah tersebut. Landasan teori yang relevan dengan permasalahan adalah: *dynamic sistems and modeling*, Java, DSOL, dan UML.

Bab 3 membahas mengenai pengumpulan dan pengolahan data. jenis-jenis data apa saja yang dibutuhkan dalam proses pembuatan model dan sumber serta metode pengumpulan data tersebut.

Pada bab 4 penulis melakukan pengolahan data dengan membuat model dengan menggunakan software DSOL. Pada bab ini, penulis juga melakukan verifikasi dan validasi model.

Bab 5 merupakan bab terakhir, akan memuat kesimpulan terhadap keseluruhan penelitian yang telah dilakukan oleh penulis.

2. LANDASAN TEORI

2.1. Simulasi

Simulasi adalah tiruan dari operasi proses atau sistem dunia nyata menurut waktu. Simulasi melibatkan penciptaan sejarah buatan dari sistem dan observasi dari sejarah buatan untuk menarik kesimpulan mengenai karakteristik operasi dari sistem nyata yang diperkenalkan.

Simulasi juga merupakan metode pemecahan masalah yang tak dapat diabaikan dan dapat menjadi jawaban dari banyak persoalan dunia nyata. Simulasi bisa digunakan untuk menjelaskan dan menganalisis perilaku dari suatu sistem,dan dapat membantu mendesain sistem nyata. Jadi, simulasi dapat digunakan untuk memodelkan sistem yang sudah ada maupun sistem konseptual.

Simulasi memiliki beberapa kelebihan, antara lain[1]:

1. Membantu memilih alternatif yang benar

   Simulasi mengizinkan kita menguji setiap aspek dari perubahan yang diusulkan ataupun penambahan lainnya tanpa memusingkan sumberdaya untuk melaksanakannya. Misalnya, ketika suatu sistem penanganan material tengah dipasang, maka perubahan dan koreksi terhadap sistem tersebut akan memakan biaya yang sangat mahal. Dalam hal ini, simulasi dapat menguji desain perubahan sistem tersebut tanpa memusingkan sumber daya atau biaya untuk mendesai sistem tersebut.
2. Fleksibel dalam waktu

   Dengan mengecilkan atau memperluas variabel waktu, simulasi memungkinkan kita memperlambat atau mempercepat fenomena sistem sehingga dapat diamati dengan lebih seksama.
3. Simuasi dapat menjawab pertanyaan ‘mengapa’

   Manajer sering ingin mengetahui mengapa suatu fenomena terjadi dalam sistem nyata. Dengan simulasi, kita menentukan jawaban dari suatu pertanyaan mengapa dengan cara merekonstruksi kejadian sampai ke detail

[1] Banks, Jerry. (1998). *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practices*. John Willey and Son, Atlanta

sistem yang mendalam untuk menentukan mengapa fenomena seperti itu terjadi. Kita tidak dapat mencapai hal seperti itu dengan sistem nyata karena kita tidak dapat mengontrol sistem tersebut secara keseluruhan.

4. Simulasi memungkinkan eksplorasi

   Salah satu manfaat terbesar dari pemakaian aplikasi simualsi adalah sekali kita sudah mengembangkan model simulasi yang valid, kita dapat mengeksplorasi kebijakan baru, prosedur operasi baru, atau metode baru tanpa bereksperimen langsung dengan sistem nyata. Modifikasi dilakukan dalam model, kita mengamati perubahan yang terjadi serta pengaruhnya di dalam suatu komputer daripada bersentuhan langsung dengan sistem sebenarnya

5. Simulasi dapat mendiagnosis masalah

   Simulasi mengizinkan kita memahami secara lebih baik interaksi diantara variabel-variabel yang menyusun sistem kompleks. Sehingga dengan melakukan diagnosis masalah kita dapat mendapatkan peningkatan pemahaman tentang tingkat kepentingan suatu variabel dan pengaruhnya terhadap kinerja sistem secara keseluruhan

6. Simulasi dapat mengidentifikasi batasan

   Dengan menggunkan simulasi misalnya untuk menganalisis *bottleneck* maka kita dapat mencari penyebab *delay* material *work in progress*, informasi, dan proses lainnya.

7. Simulasi dapat mengembangkan pemahaman

   Simulasi dapat menyediakan pemahaman tentang bagaimana suatu sistem benar-benar beroperasi daipada menggantungkan diri pada prediksi seseorang tentang bagaimana suatu sistem akan beroperasi

8. Simulasi dapat memvisualisasikan rencana

   Simulasi memungkinkan kita melihat suatu fasilitas atau organisasi berjalan seperti aslinya dengan bantuan suatu fitur animasi baik dua dimensi maupun tiga dimensi sesuai aplikasi simulasi yang dipakai

9. Simulasi dapat membangun konsensus

   Menggunakan simulasi untuk menunjukan perubahan desain menciptakan opini yang objektif dan mencegah penarikan kesimpulan emosional atau hanya karena opini seseorang saja

10. Simulasi membantu mempersiapkan perubahan

    Masa depan selalu berubah. Dengan mengajukan pertanyaan *what if* akan sangat berguna dalam mendesain sistem baru dan mendesain ulang sistem yang sudah ada. Simulasi memungkinkan kita membangun suatu skenario yang berbeda-beda sehingga kita menjadi siap melihat perubahan suatu sistem

11. Simulasi adalah investasi yang bijak

    Biaya tipikal dari telaah simulasi secara substansi lebih kecil dari 1% dari jumlah total yang dihabiskan dalam implementasi desain dan desain ulang suatu sistem. Karena biaya perubahan atau modifikasi dari suatu sistem setelah sistem dipasang sangat besar, maka simulasi merupakan investasi yang bijak

12. Simulasi dapat menetukan kebutuhan

    Simulasi dapat digunakan untuk menentukan kebutuhan dari suatu desain sistem. Sebagai contoh, spesifikasi dari suatu jenis mesin dalam sistem yang kompleks untuk mencapai suatu tujuan yang dikehendaki mungkin tidak dapat ditentukan. Dengan mensimulasikan kemampuan mesin yang berbeda-beda, kebutuhan tersebut dapat dipenuhi

Namun demikian, simulasi juga memiliki kekurangan yaitu [2]:

1. Pengembangan model membutuhkan pelatihan

   Ketika dua model suatu sistem yang sama dibangun oleh dua orang berkompeten yang berbeda, model tersebut mungkin saja memiliki persamaan, tetapi pada umumnya model tersebut akan berbeda walaupun sistemnya sama

2. Hasil simulasi mungkin sulit diinterpretasi

   Karena kebanyakan output simulasi adalah variabel acak, akan susah menentukan apakah suatu observasi merupakan hasil dari hubungan antar komponen dalam sistem atau hasil dari suatu variabel acak

3. Pemodelan dan analisis simulasi mungkin mahal dan menghabiskan waktu

   Dengan melakukan penghematan terhadap sumberdaya untuk memodelkan dan menganalusis suatu sistem akan dapat menghasilkan model simulasi dan analisis yang salah sehingga ada saatnya penggunaan simulasi memang menghabiskan biaya mahal

[2] *Ibid.,*

4. Penggunaan simulasi bisa tidak tepat

   Simulasi digunakan dalam kasus ketika solusi analitis memungkinkan, atau lebih dipilih. Sering terjadi bahwa penggunaan simulasi untuk memecahkan suatu masalah terlalu berlebihan karena masalah tersebut dapat diselesaikan dengan metode yang tepat, lebih sederhana, dan lebih cepat

2.2. Teori Antrian

Antrian adalah suatu garis tunggu dari entitas yang memerlukan layanan dari satu atau lebih layanan. Antrian dapat diklasifikasikan menjadi beberapa macam. Klasifikasi antrian menurut Hillier dan Lieberman adalah sebagai berikut:

1. Sistem pelayanan komersial

   Sistem pelayanan komersial merupakan aplikasi yang sangat luas dari model-model antrian, seperti restoran, kafetaria, toko, salon, butik, supermarket, dan lain-lain
2. Sistem pelayanan bisnis-industri

   Sistem pelayanan bisnis industri mencakup lini produksi, sistem *material handling*, sistem pergudangan, dan sistem-sistem informasi komputer
3. Sistem pelayanan transportasi

   Sistem pelayanan transportasi contohnya antrian di gerbang tol, apliakasi dalam penanganan kemacetan lalu lintas, penjadwalan kereta api, dan sebagainya
4. Sistem pelayanan sosial

   Sistem pelayanan sosial merupakan sistem-sistem pelayanan yang dikelola oleh kantor-kantor dan jawatan-jawatan lokal maupun nasional, seperti kantor registrasi SIM dan STNK, kantor pos, rumah sakit, puskesmas, dan lain-lain.

Sebagai suatu sistem, antrian dapat dipecah-pecah menjadi beberapa komponen. Komponen dasar proses antrian adalah:

1. Kedatangan *(arrival)*

   Kedatangan atau arrival adalah komponen atau variabel antrian berupa entitas yang datang sebagai input dari sistem antrian. Varaibel kedatangan ini bisa merupakan variabel yang deterministik namun biasanya merupakan variabel

acak yang terdistribusi menurut distribusi peluang baik diskret maupun kontinu

2. Pelayan *(resources)*

   Pelayan bisa berbentuk orang atau mesin ataupun berupa mekanisme pelayanan. Biasanya berupa server-server yang bertujuan melayani dan menjadi sumber *delay* ketika entitas dilayani

3. Antri *(queue)*

   Timbulnya antrian terutama tergantung dari sifat kedatangan dan proses pelayanan. Jika tak ada antrian berarti terdapat pelayan yang menganggur atau kelebihan fasilitas pelayanan

Penentu antrian lain yang penting adalah disiplin antri. Disiplin antri adalah aturan keputusan yang menjelaskan cara melayani pengantri. Ada lima bentuk disiplin pelayanan yang biasa digunakan, yaitu:

1. *First-Come First-Served* (FCFS) atau *First-In First-Out* (FIFO) artinya, lebih dulu datang, lebih dulu dilayani (keluar). Misalnya, antrian pada loket pembelian tiket kereta api
2. *Last-Come First-Served* (LCFS) atau *Last-In First-Out* (LIFO) artinya, yang tiba terakhir yang lebih dulu keluar. Misalnya, sistem antrian dalam elevator untuk lantai yang sama
3. *Service In Random Order* (SIRO) artinya, panggilan didasarkan pada peluang secara random, tidak soal siapa yang lebih dulu tiba
4. *Priority Service* (PS) artinya, prioritas pelayanan diberikan kepada pelanggan yang mempunyai prioritas lebih tinggi dibandingkan dengan pelanggan yang mempunyai prioritas lebih rendah, meskipun yang terakhir ini kemungkinan sudah lebih dahulu tiba dalam garis tunggu. Kejadian seperti ini kemungkinan disebabkan oleh beberapa hal, misalnya seseorang yang dalam keadaan penyakit lebih berat dibanding dengan orang lain dalam suatu tempat praktek dokter

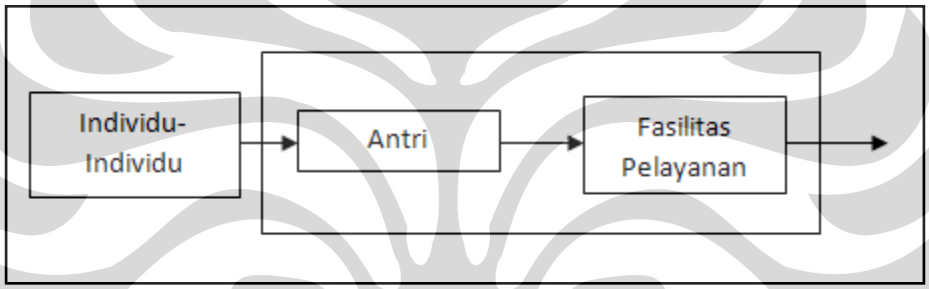
Entitas yang berada dalam garis tunggu tetap tinggal di sana sampai dilayani. Namun hal tersebut bisa saja tidak terjadi karena suatu entitas bisa meninggalkan antrian misalnya karena bosan terlalu lama menunggu. Untuk entitas yang

meninggalkan antrian sebelum dilayani digunakan istilah pengingkaran *(reneging).* Pengingkaran dapat bergantung pada panjang garis tunggu atau lama waktu tunggu. Istilah penolakan *(balking)* dipakai untuk menjelaskan entitas yang menolak untuk bergabung dalam garis tunggu.

Ada empat model struktur antrian dasar yang umum terjadi dalam sistem antrian, yaitu:

1. *Single Channel - Single Phase*

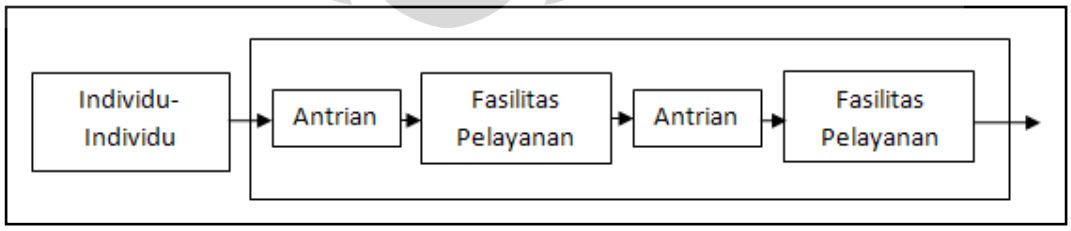
   Single Channel berarti hanya ada satu jalur yang memasuki sistem pelayanan atau ada satu fasilitas pelayanan. *Single Phase* berarti hanya ada satu pelayanan



Gambar 2.1 Antrian *Single Channel - Single Phase*

2. *Single Channel - Multi Phase*

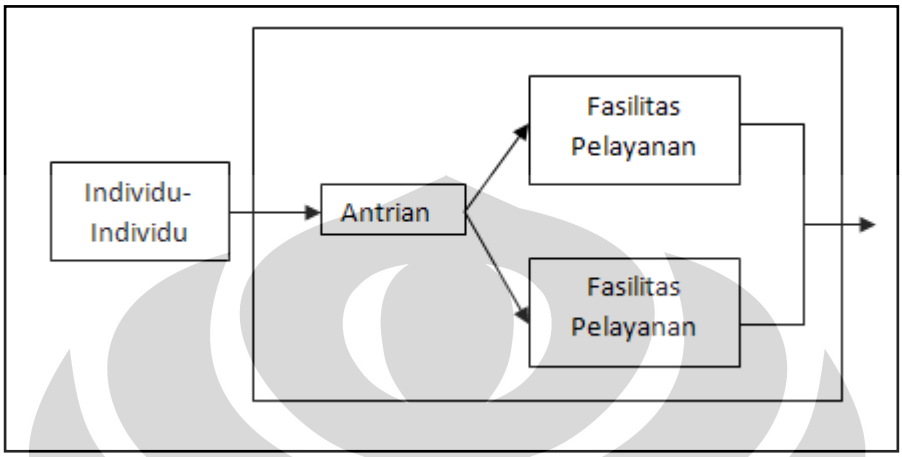
   Istilah *Multi Phase* menunjukkan ada dua atau lebih pelayanan yang dilaksanakan secara berurutan (dalam *phase-phase*). Sebagai contoh: pencucian mobil



Gambar 2.2 Antrian *Single Channel - Multi Phase*

3. *Multi Channel - Single Phase*

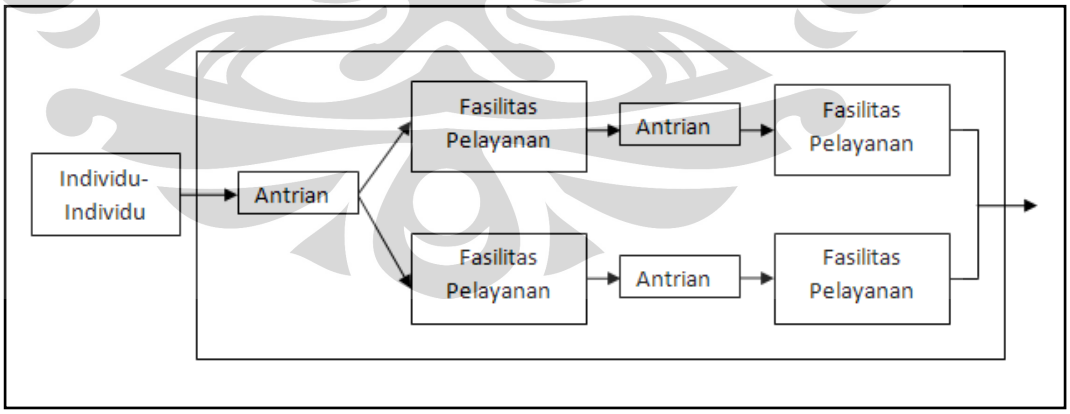
Sistem *Multi Channel - Single Phase* terjadi kapan saja di mana ada dua atau lebih fasilitas pelayanan dialiri oleh antrian tunggal, sebagai contoh model ini adalah antrian pada teller sebuah bank



Gambar 2.3 Antrian *Multi Channel - Single Phase*

4. *Multi Channel - Multi Phase*

Sistem *Multi Channel - Multi Phase* contohnya adalah pelayanan kepada pasien di rumah sakit mulai dari pendaftaran, diagnosa, penyembuhan sampai pembayaran. Setiap sistem-sistem ini mempunyai beberapa fasilitas pelayanan pada setiap tahapnya



Gambar 2.4 Antrian *Multi Channel - Multi Phase*

Untuk memodelkan kelompok antrian yang berbeda dapat digunakan Notasi Kendall. Notasi tersebut sering dipergunakan diantaranya karena notasi itu merupakan alat yang efisien untuk mengidentifikasi tidak hanya model-model antrian, tetapi juga asumsi-asumsi yang harus dipenuhi.

Format umum model antrian adalah: (a/b/c);(d/e/f) di mana:

a = distribusi pertibaan / kedatangan (arrival distribution), yaitu jumlah pertibaan pertambahan waktu
b = distribusi waktu pelayanan / perberangkatan, yaitu selang waktu antara satuan-satuan yang dilayani (berangkat)
c = jumlah saluran pelayanan paralel dalam system
d = disiplin pelayanan
e = jumlah maksimum yang diperkenankan berada dalam sistem (dalam pelayanan ditambah garis tunggu)
f = besarnya populasi masukan

Keterangan:

1. Untuk huruf a dan b, dapat digunakan kode - kode berikut sebagai pengganti :
   M = Distribusi pertibaan Poisson atau distribusi pelayanan (perberangkatan) eksponensial; juga sama dengan distribusi waktu antara pertibaan eksponensial atau distribusi satuan yang dilayani Poisson
   D = Antarpertibaan atau waktu pelayanan tetap
   G = Distribusi umum perberangkatan atau waktu pelayanan.
2. Untuk huruf c, dipergunakan bilangan bulat positif yang menyatakan jumlah pelayanan paralel.
3. Untuk huruf d, dipakai kode - kode pengganti :
   FIFO atau FCFS = Firs in First out atau First Come First Served
   LIFO atau LCFS = Last in First out atau Last Come First Served
   SIRO = Service in Random Order
   G D = General Service Disciplint

4. Untuk huruf e dan f, dipergunakan kode N (untuk menyatakan jumlah terbatas) atau ∞ (tak berhingga satuan - satuan dalam sistem antrian dan populasi masukan)

Misalnya, model (M/M/1);(FIFO/∞/∞), berarti bahwa model menyatakan pertibaan didistribusikan secara Poisson, waktu pelayanan didistribusikan secara eksponensial, pelayanan adalah satu atau seorang, disiplin antrian adalah *first in first out*, tidak berhingga jumlah langganan boleh masuk dalam sistem antrian, dan ukuran (besarnya) populasi masukan adalah tak berhingga. Menurut Siagian (1987), berikut ini adalah beberapa karakteristik dari sistem antrian untuk model (M/M/1);(FIFO/∞/∞):

1. Intensitas Lalu – Lintas

$$\rho = \frac{\lambda}{\mu}$$

Lambang ρ disebut intensitas lalulintas yakni hasil bagi antara laju pertibaan dan laju pelayanan. Makin besar harga ρ makin panjang antrian dan sebaliknya

2. Periode Sibuk

Bila $f(b)$ merupakan fungsi peluang periode sibuk, maka:

$$f(b) = \rho = \frac{\lambda}{\mu}$$

3. Distribusi Peluang dari Langganan dalam Sistem

Bila ρ merupakan peluang bahwa sistem antrian adalah sibuk, maka tentu 1–ρ merupakan peluang bahwa sistem tidak dalam keadaan sibuk pada sebarang waktu. Artinya 1-ρ merupakan peluang bahwa sistem antrian tidak mempunyai langganan

4. Jumlah Rata – rata dalam Sistem

$$E(nt) = \frac{\rho}{1-\rho}$$

5. Jumlah rata-rata dalam antrian

$$E(nw) = \frac{\rho^2}{1-\rho}$$

6. Jumlah rata-rata yang menerima layanan

$$E(ns) = \rho$$

7. Waktu rata-rata dalam system

$$E(Tt) = \frac{1}{\mu - \lambda}$$

8. Waktu rata-rata dalam antrian

$$E(Tw) = \frac{\lambda}{\mu(\mu - \lambda)}$$

9. Waktu pelayanan rata-rata

$$E(Ts) = \frac{1}{\mu}$$

2.3. Java

Java adalah suatu bahasa pemrograman yang aslinya dikembangkan oleh Sun Microsystems dan dirilis pada tahun 1995 sebagai komponen utama dari *Java Platform* milik Sun. Sun secara resmi mengumumkan Java pada konferensi utama pada bulan Mei tahun 1995. Java memperoleh perhatian dari komunitas bisnis karena fenomena ketertarikan terhadap dunia web. Java sekarang digunakan untuk mengembangkan aplikasi enterprise skala besar, untuk meningkatkan fungsionalitas suatu web server, menyediakan aplikasi bagi peralatan konsumen seperti cell phones, pager, dan PDA dan tujuan penggunaan lainnya.[3]

Bahasanya sendiri meminjam sintaks dari bahasa-bahasa pemrograman sebelumnya seperti C dan C++ tetapi memiliki model objek yang lebih sederhana dan lebih sedikit fasilitas level bawah untuk mengakses prosesor atau mesin. James Gosling menciptakan Java pada Juni 1991 untuk digunakan dalam proyek aplikasi televisi. Awalnya bahasa tersebut diberi nama Oak, dan dirubah menjadi Java karena nama Oak sudah ada sebagai nama sebuah bahasa pemrograman.

Aplikasi Java biasanya dikompilasi ke dalam suatu bytecode yang akan berjalan dalam *Java Virtual Machine* tanpa bergantung pada arsitektur komputer. Kompiler Java asli beserta virtual mesinnya serta class library dari Java itu sendiri

[3] Deitel, H.M & Deitel, P.J. (2004). *Java How to Program*. New Jersey: Prentice Hall.

dikembangkan oleh Sun sejak 1995 sampai 2007 dan diwadahi dalam suatu spesifikasi bernama Java Community Process. Namun sejak tahun 2007 Sun telah membebaskan Java dan melisensikannya secara open source dibawah lisensi GNU.

Berdasarkan *white paper* resmi dari Sun, Java memiliki karakteristik sebagai berikut [4]:

- Sederhana *(Simple)*

  Bahasa pemrograman Java menggunakan Sintaks mirip dengan C++ namun sintaks pada Jav telah banyak diperbaiki terutama menghilangkan penggunaan pointer yang rumit dan multiple inheritance. Java juga menggunakan *automatic memory allocation* dan *memory garbage collection*
- Berorientasi objek *(Object Oriented)*

  Bahasa pemrograman *object-oriented* membagi program ke dalam modul yang terpisah yang dinamakan objek yang mengenkapsulasi data program dan operasinya. Sehingga, *object oriented programming* dan *object oriented design* mengacu pada cara tertentu dalam mengorganisasi program, yang sekarang lebih dipilih terutama dalam membangun sistem aplikasi perangkat lunak yang kompleks. Tidak seperti C++, Java didesain dari awal sebagai bahasa yang murni *object oriented*.
- Terdistribusi (Distributed)

  Ini berarti bahwa program yang dibuat dapat dijalankan di dalam suatu jaringan komputer. Sebagai tambahan, Java hadir dengan koleksi library yang di desain untuk secara langsung digunakan untuk tipe aplikasi yang berbeda-beda sehingga akan sangat mudah dalam mengembangkan aplikasi internet dan web. Ini merupakan alasan mengapa Java sangat cocok dalam mendukung aplikasi jaringan perusahaan
- Diinterpretasi *(Interpreted)*

[4] http://www.java.sun.com

Program Java dijalankan menggunakan interpreter yaitu *Java Virtual Machine (JVM)*. Hal ini menyebabkan *source code* Java yang telah dikompilasi menjadi *Java bytecodes* dapat dijalankan pada platform yang berbeda-beda

- Liat *(Robust)*

  Berarti bahwa suatu kesalahan atau error yang terjadi dalam aplikasi Java tidak akan menyebabkan sistem crash seperti kesalahan atau error yang terjadi dalam bahasa pemrograman lainnya. Bahkan, berbagai fitur di dalam bahasanya memungkinkan banyak kesalahan atau error untuk dideteksi lebih dini pada saat program dijalankan
- Aman *(Secure)*

  Didesain untuk berjalan dalam suatu jaringan, java mengandung fitur untuk melindungi dari kode yang tidak dapat dipercaya yang mengandung virus dan dapat mengkorupsi sistem dalam suatu cara tertentu.
- Bebas Arsitektur *(Architecture Neutral*)

  Java trademark *"write once, run anywhere"* mengandung arti bahwa aplikasi yang dibuat dalam bahasa java dapat berjalan tanpa perubahan ketika dijalankan dalam platform yang berbeda.Untuk alasan ini maka Java secara penuh sangat cocok untuk aplikasi web
- Mudah dibawa *(Portable)*

  Source code maupun program Java dapat dengan mudah dibawa ke platform yang berbeda-beda tanpa harus dikompilasi ulang
- Berkinerja *(Performance)*

  Kinerja pada Java sering dikatakan kurang tinggi. Namun kinerja Java dapat ditingkatkan menggunakan kompilasi Java lain seperti buatan Inprise, Microsoft ataupun Symantec yang menggunakan *Just In Time Compilers* (JIT)
- *Multithreaded*

  Java mempunyai kemampuan untuk membuat suatu program yang dapat melakukan beberapa pekerjaan secara sekaligus dan simultan
- Dinamis *(Dynamic)*

Java didesain untuk dapat dijalankan pada lingkungan yang dinamis. Perubahan pada suatu *class* dengan menambahkan properties ataupun method dapat dilakukan tanpa menggangu program yang menggunakan *class* tersebut

2.4. DSOL

DSOL (*Distributed Simulation Object Library*) merupakan aplikasi *open source* yang dibuat dengan menggunakan bahasa program Java, yang bertujuan sebagai *library* untuk digunakan dalam mensimulasikan baik model diskret maupun kontinu. DSOL pertama kali diperkenalkan pada saat IEEE's Winter Simulation Confrence 2002. Di era komputerisasi sekarang membuat perkembangan DSOL semakin mudah karena kita dibekali dengan seperangkat *tools* dan teknik untuk mendefinisikan secara spesifik konsep dari simulasi itu sendiri. Mulai dari langkah konseptualisasi, formulasi, verifikasi, sampai validasi. Adapun beberapa kelemahan *tools* simulasi yang sekarang sering dialami adalah[5]:

1. *Usefulness*

   Kelemahan pada simulasi yang ada adalah kesulitan dalam menghubungkan *tools* pada simulasi yang mendasari transaksi pada sistem informasi. Karena konsep utama *usefulness* pada sistem pengambilan keputusan adalah menambah nilai pada proses pengambilan keputusan tersebut. Dengan demikian hal tersebut berhubungan dengan model analitik, yang mendasari *knowledge* dan sumber informasi tersedia dalam model atau *tools* yang ada. Syarat utama dari *usefulness* adalah seberapa cocok simulasi yang kita buat dengan kondisi nyata.

2. *Usability*

   Usability menghubungkan antara manusia, proses, dan teknologi. *Usability* bertanggung jawab dalam mengkoordinasikan antara konseptualisasi dan spesifikasi dengan representasi model yang akan diteliti. Hal ini bergantung pada tampilan antara pengguna dan teknologi dalam pengambilan keputusan (Keen dan Sol, 2005). Inilah yang menjadi syarat

[5] Peter H.M. Jacobs, "The DSOL Simulation Suite: Enabling Multi-Formalism Simulation In A Distributed Context", Technische Universiteit Delft, 2005, hal.5.

dari *usability* itu sendiri, seberapa besar pengguna dapat memahami dan memakai simulasi yang dibuat.

3. *Usage*

   Usage memperlihatkan fleksibilitas, adaptivitas, dan suitabilitas dalam konteks perusahaan, teknis, ataupun sosial. Apakah *tools* simulasi dapat diintegrasikan dan dipakai oleh sistem informasi dalam suatu perusahaan.

Adapun beberapa syarat utama dari penggunaan DSOL, yaitu[6]:

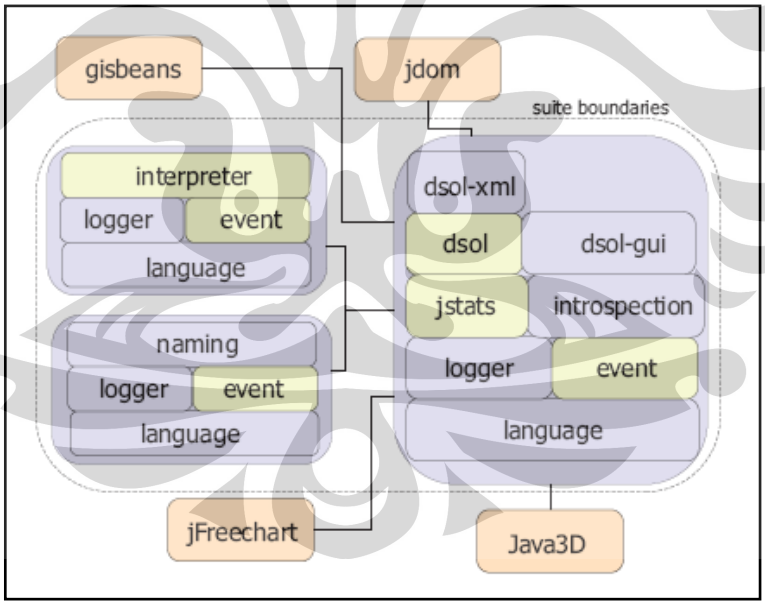
1. N buah pengambil keputusan yang didukung melalui internet
2. N buah formalisasi model (diskrit, kontinyu)
3. N buah ahli yang secara simultan dapat mendukung dalam proses spesifikasi model
4. *Service* dari simulasi harus dibekali dengan kemampuan untuk malakukan hubungan dengan *web-enabled service* lainnya (laporan, database)
5. N buah orang aksesor yang akan menjalankan eksperimen

Pelayanan utama dari DSOL adalah menyediakan *interface* dan *class* untuk simulasi yang berisi formalisasi diskrit dan kontinyu, spesifikasi dari eksperimen dengan menggunakan DSOL, distribusi diskrit dan kontinyu, statistik, dan *class* yang mendukung animasi 2D atau 3D. Dalam masa yang akan datang, DSOL akan dikembangkan dengan *multi-actor conceptual modeling environment*. Hal berikut adalah *service* yang dapat dilakukan oleh DSOL sehingga cocok digunakan untuk simulasi[7]:

- *Language*; servis ini menyediakan bahasa pemrograman dasar yang tidak tersedia pada Java *development kit.*
- *Event*; servis ini menyediakan *distributed asynchronous event mechanism* sehingga terdapat *interface* dan *class* untuk setiap *event*.
- *Logger*; servis ini memperluas Java's *logging mechanism* dengan menyediakan beberapa *logging classes* untuk memfasilitasi *package based logging* untuk DSOL.

[6] *Goals of DSOL*, 2005 <http://sk-3.tbm.tudelft.nl/simulation>
[7] Peter H.M. Jacobs, *Op.Cit.*, hal.68.

- *Naming*; servis ini menghubungkan DSOL dengan kerangka JNDI (*Java Naming and Directory Interface*) sehingga objek model dan output data statistik dapat disimpan dan didistribusikan.
- *Jstats*; servis ini menyediakan fungsi distribusi diskrit dan kontinyu dan berhubungan dengan DSOL dengan *library* matematis eksternal dan grafik.
- *Instropection*; servis ini memeriksa kesesuaian objek Java berdasarkan *Java's bean model*.
- *Interpreter*; mesin virtual Java yang mengimplementasikan Java sehingga memungkinkan interaksi proses *single threaded*
- *DSOL*; servis inti yang menyediakan *interface* dan *class* untuk simulasi.
- *DSOL-xml;* servis ini memparse eksperimen dari xml menjadi representasi dari Java-nya.
- *DSOL-gui*; *graphical user interface* pada DSOL



Gambar 2.5 DSOL *Services* (Sumber: Peter H. M. Jacobs, 2005)

2.5. UML

Unified Modelling Language (UML) merupakan notasi grafis yang membantu dalam menjelaskan dan mendesain sistem piranti lunak terutama perangkat lunak yang dirancang dalam suatu paradigma Object Oriented. UML menawarkan

sebuah standar untuk merancang sebuah sistem yang standarnya dikontrol oleh suatu badan konsorsium terbuka yang bernama Object Management Group (OMG) yang lebih dikenal sebagai standar CORBA (Common Object Request Broker Object).

UML lahir dari penyatuan berbagai bahasa pemodelan grafis yang telah ada sejak tahun 1980-an sampai awal tahun 1990-an seperti Object oriented Design (OOD), Object Oriented Software Engineering (OOSE), Object Modelling Technique (OMT), dan sebagainya dimana masing-masing metodologi tersebut membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila perlu kerjasama dengan grup/perusahaan lain yang menggunakan metodologi yang berlainan.

Notasi UML terutama diturunkan dari tiga notasi yang telah ada sebelumnya, yaitu: Grady Booch dengan OOD, Jim Rumbaugh dengan OMT, dan Ivar Jacobson dengan OOSE. Usaha penggabungan dimulai sejak tahun 1994, dan pada tahun 1995 *draft* pertama UML telah dirilis. Sejak tahun 1996 pengembangan UML dikoordinasikan oleh Object Management Group (OMG - http://www.omg.org).

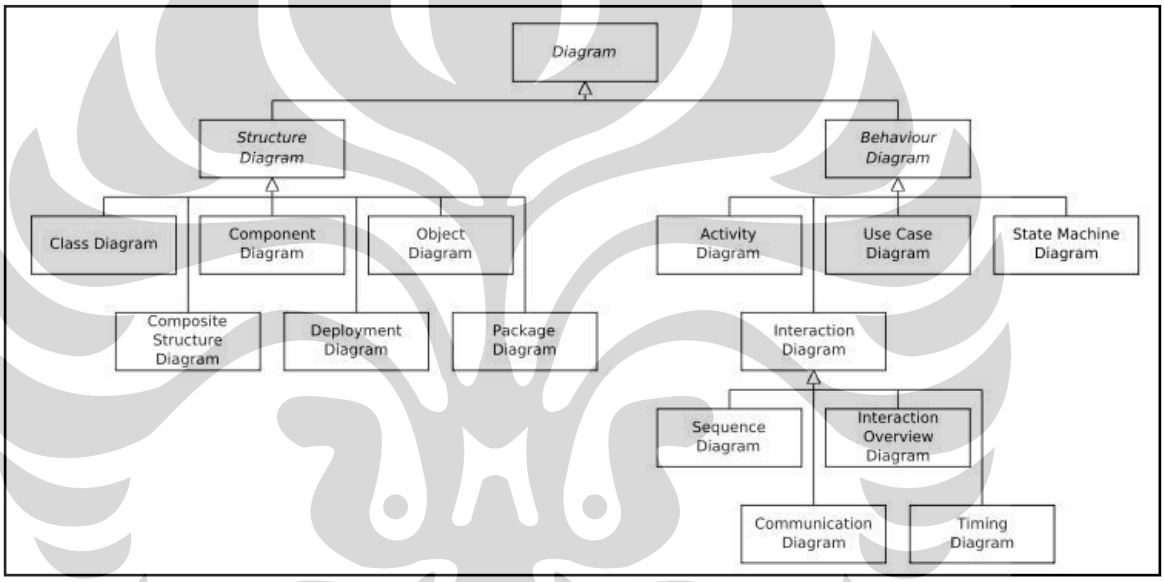
Versi terbaru dari UML saat ini adalah versi 2.x yang memiliki tiga belas jenis diagram secara keseluruhan. Secara umum, diagram UML dapat dikategorikan ke dalam tiga jenis diagram utama yaitu[8]:

1. *Structure diagrams*
    - *Class diagram*
    - *Component diagram*
    - *Composite structure diagram (UML 2.x)*
    - *Deployment diagram*
    - *Object diagram*
    - *Package diagram*
2. *Behavior diagrams*

[8] Fowler, Martin. (2004). *UML Distilled: A Brief Guide ToThe Standard Object Modeling Language*, Third Edition. Boston: Addison-Wesley.

- *Activity diagram*
- *State Machine diagram*
- *Use case diagram*

3. *Interaction diagrams*
   - *Communication diagram*
   - *Interaction overview diagram (UML 2.x)*
   - *Sequence diagram*
   - *Timing diagram (UML 2.x)*



Gambar 2.6 Diagram Pada UML 2.0 (Sumber: Martin Fowler, 2004)

2.5.1. Class Diagram

Class diagram adalah diagram UML yang paling dikenal dan mewakili UML itu sendiri karena mayoritas diagram UML yang sering dibuat adalah *class diagram. Class diagram* tidak hanya digunakan secara luas tetapi juga merupakan pemain utama dan terbesar dalam konsep pemodelan.

Class diagram menjelaskan tipe objek di dalam sistem sekaligus menjelaskan hubungan statis yang muncul diantara objek-objek tersebut. *Class diagram* juga menunjukan kepemilikan operasi dari *class* bersangkutan dan batasan-batasan bagaimana objek-objek tersebut saling terhubung.

Dalam *object oriented programming*, suatu *class* dapat didefinisikan sebagai suatu *blueprint* dari suatu objek dan menjadi suatu spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek serta merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu objek serta bagaimana objek tersebut beroperasi dan saling berkomunikasi melalui metode/fungsi. Secara umum, suatu *class* memiliki tiga area pokok:

1. Nama
2. Properties / atribut/ field

   Atribut dan metode dapat memiliki salah satu sifat berikut:

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

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

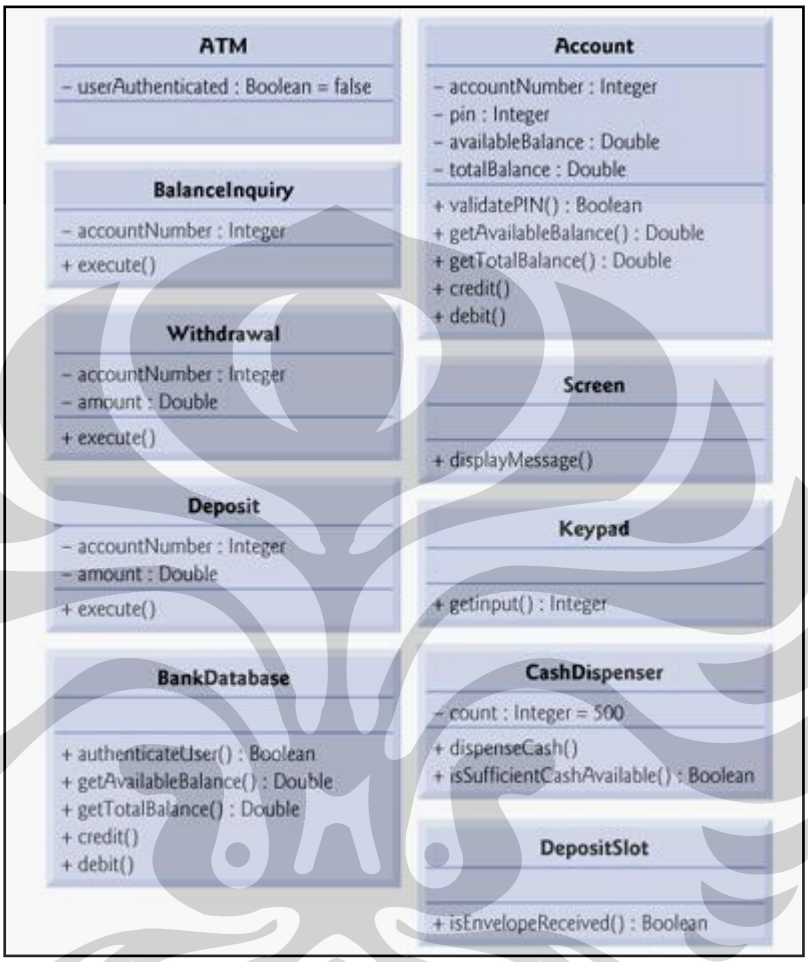
Dalam *class diagram*, terdapat beberapa jenis hubungan antar *class*. Hubungan-hubungan tersebut adalah sebagai berikut:

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah menunjukkan arah *query* antar *class*.

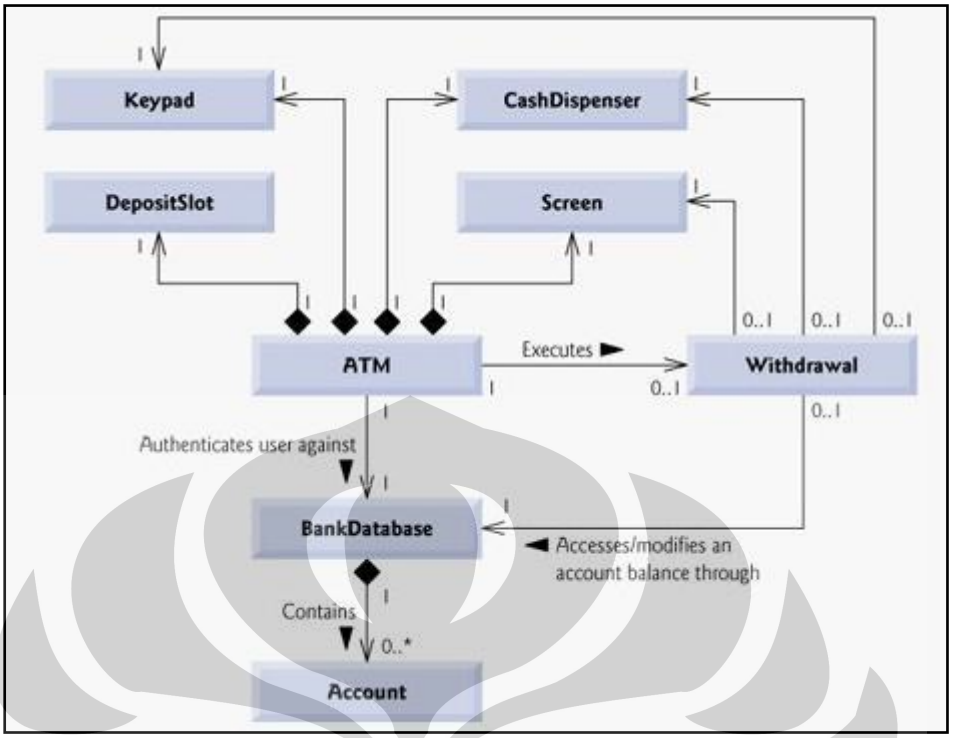
2. Agregasi, yaitu hubungan yang menyatakan suatu *class* terdiri atas *class* lain.

3. Pewarisan, yaitu hubungan hierarki 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 pewarisan adalah generalisasi.

4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang dikirim dari satu *class* ke *class* lain. Hubungan dinamis dapat digambarkan juga dengan menggunakan *sequence diagram.*



Gambar 2.7 *Class Diagram* Pada UML 2.0 (Sumber: Deitel & Deitel, 2004)

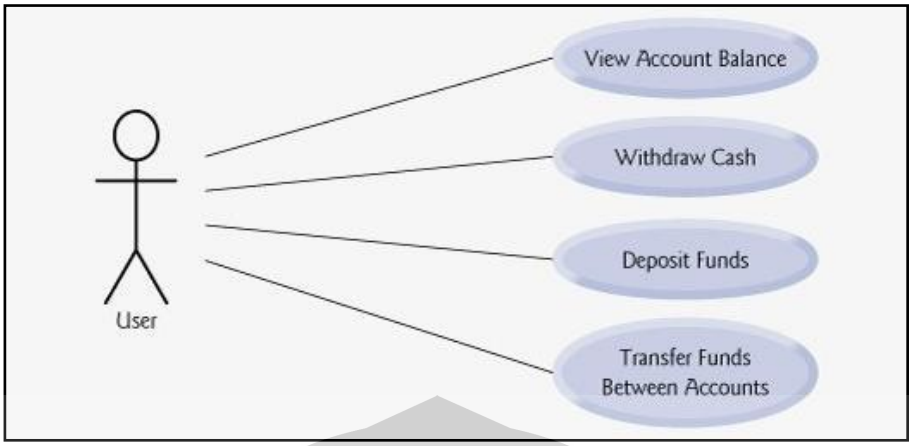


Gambar 2.8 *Class Diagram* Pada UML 2.0 Beserta Hubungan Antar *Class* (Sumber: Deitel & Deitel, 2004)

2.5.2. Use Case Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah apa yang diperbuat sistem, bukan bagaimana sistem berjalan. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

Use case diagram dapat sangat membantu bila kita sedang menyusun kebutuhan sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang skenario tes untuk semua fitur yang ada dalam sistem.



Gambar 2.9 *Use Case Diagram* (Sumber: Deitel & Deitel, 2004)

2.5.3. Statechart Diagram

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya) suatu objek pada sistem sebagai akibat dari stimulan 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. Transisi antar *state* umumnya memiliki kondisi *guard* yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. Pekerjaan yang dilakukan sebagai akibat dari *event* tertentu dituliskan dengan diawali garis miring. Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.

2.5.4. Activity Diagram

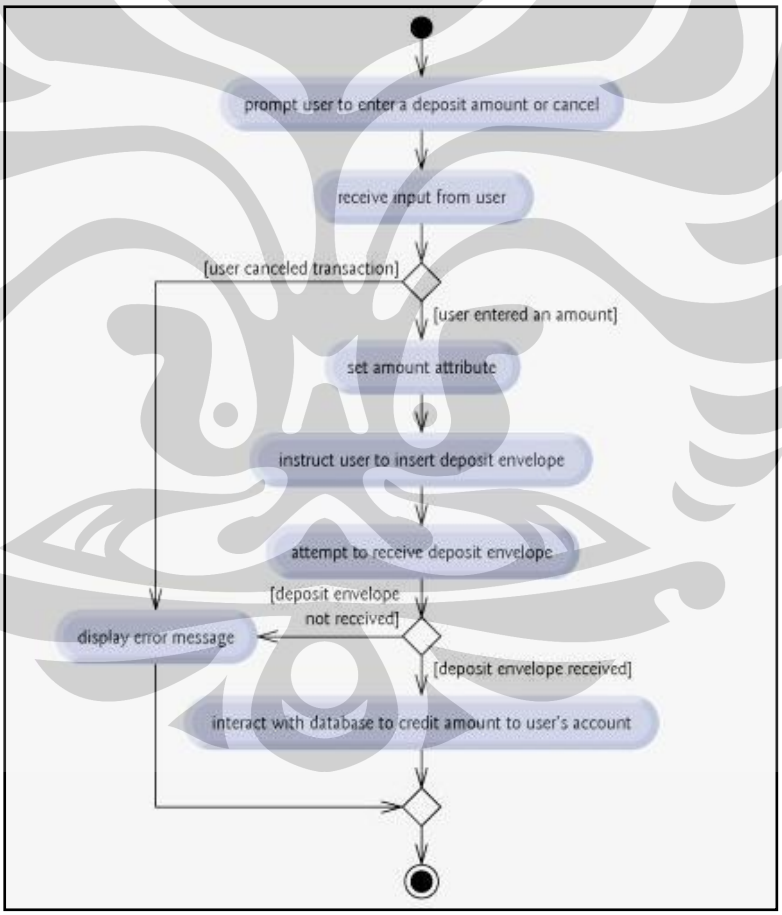
Activity diagram mengambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan state diagram khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi dipicu oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu, *statechart diagram* tidak menggambarkan kelakuan internal sebuah sistem (dan interaksi antar sub-sistem) secara eksak,

tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Tata cara penggambaran dalam UML hampir sama dengan penggambaran *state diagram*, menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan kelakuan pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal.



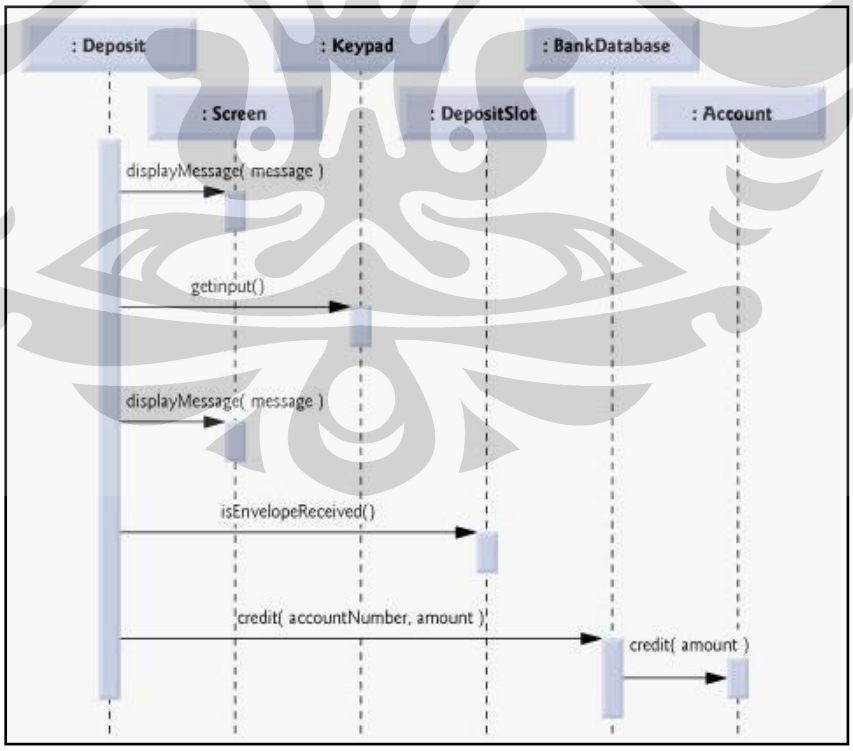
Gambar 2.10 *Activity Diagram* (Sumber: Deitel & Deitel, 2004)

2.5.5. Sequence Diagram

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

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

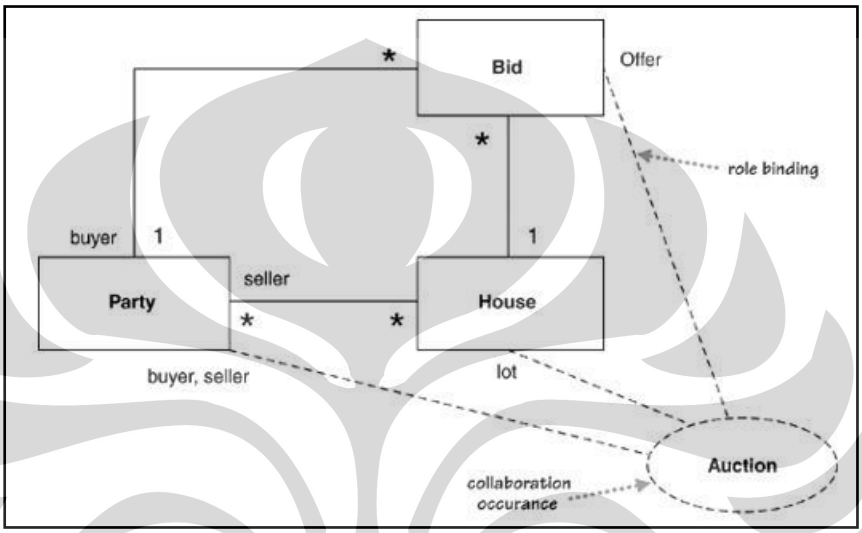
Masing-masing objek, termasuk aktor, memiliki garis hidup vertikal. Pesan (*message*) digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, pesan akan dipetakan menjadi metode dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya seubah pesan.



Gambar 2.11 *Sequence Diagram* (Sumber: Deitel & Deitel, 2004)

2.5.6. Collaboration Diagram

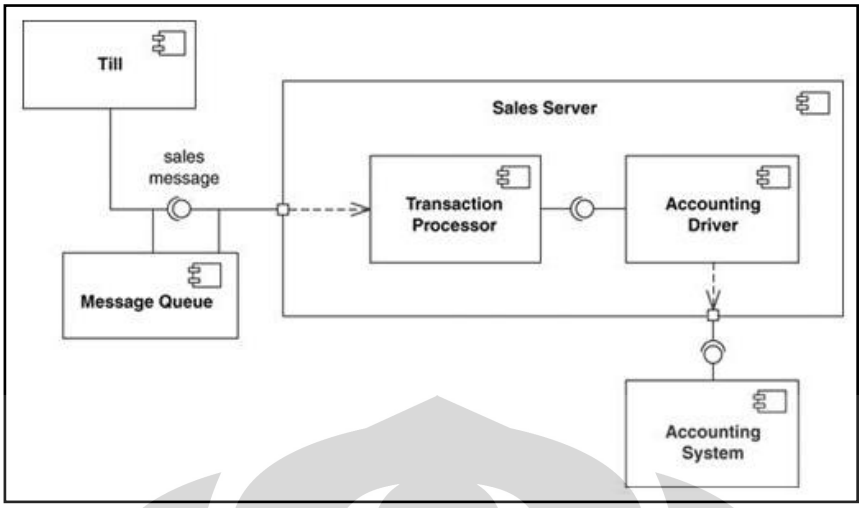
Hampir sama dengan *sequence diagram*, *collaboration diagram* juga menggambarkan interaksi antar objek, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian pesan. Setiap pesan memiliki nomor urut, di mana pesan dari level tertinggi memiliki nomor 1. Pesan dari level yang sama memiliki prefiks yang sama.



Gambar 2.12 *Collaboration Diagram* (Sumber: Martin Fowler, 2004)

2.5.7. Component Diagram

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan di antaranya. Komponen piranti lunak adalah modul berisi kode program, 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*, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen juga dapat berupa *inteface*, yaitu kumpulan layanan yang disediakan oleh sebuah komponen untuk komponen lain.



Gambar 2.13 *Component Diagram* (Sumber: Martin Fowler, 2004)

2.5.8. Deployment Diagram

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

Sebuah *node* adalah *server*, *workstation*, atau piranti keras lain yang digunakan untuk memasang komponen dalam lingkungan sebenarnya. Hubungan antar node (misalnya TCP/IP) dan kebutuhan *node* juga didefinisikan dalam diagram ini.

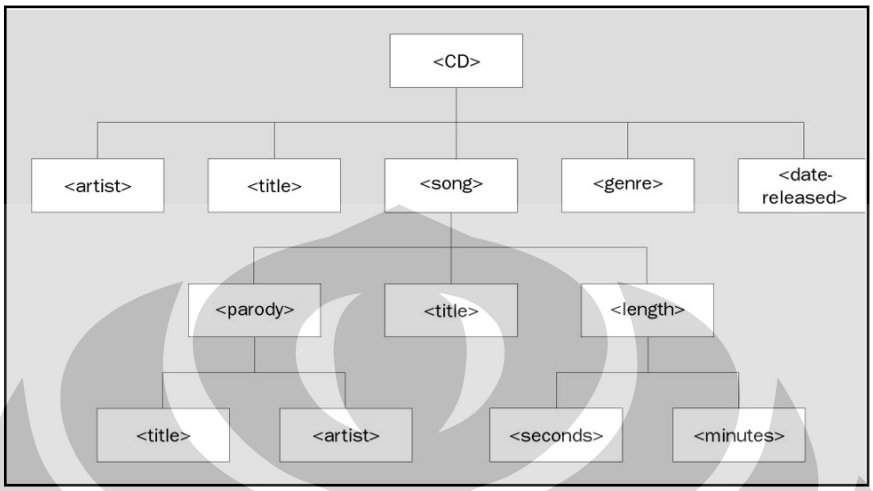
2.5. XML

XML yang merupakan singkatan dari Extensible Markup Language didefinisikan sebagai subset dari SGML yang sasarannya adalah memungkinkan SGML generik dapat dilayani, diterima, dan diproses pada web dengan cara yang sekarang dipakai oleh HTML. XML dirancang untuk kemudahan implementasi dan untuk operabilitas dengan SGML dan HTML.[9]

Sesuai definisinya, XML merupakan bahasa markup yang dirancang khusus untuk penyampaian informasi melalui *World Wide Web* persis seperti HTML yang telah menjadi bahasa standar untuk membuat halaman *web* sejak kehadiran awal dunia *web*.

[9]< http://w3c.org/tr/rec-xml>

XML merupakan format umum untuk mendefinisikan suatu data sehingga data bias dibagi dan bebas *platform*. Data didefinisikan dan dibagi kedalam *tags* menurut urutan seperti diagram pohon.



Gambar 2.14 *XML Diagram* (Beginning XML, 2nd Edition: XML Schemas, SOAP, XSLT, DOM, and SAX 2.0)

3. PENGUMPULAN DAN PENGOLAHAN DATA

3.1. Konseptualisasi

Konseptualisasi merupakan langkah pertama dalam pembuatan model sistem dinamis. konseptualisasi bertujuan untuk memetakan ide simulasi ke dalam suatu bentuk nyata yang akan menjadi sebuah sketsa yang dapat membantu dalam mendesain model yang dibuat. Konseptualisasi juga berfungsi membatasi ruang lingkup model yang akan dibuat sehingga model yang dihasilkan dapat merepresentasikan permasalahan yang ada. Model yang digunakan dalam penelitian ini adalah model antrian multi server sederhana sehingga memudahkan dalam memahami konsep dari *object-oriented simulation* itu sendiri.

3.1.1. Identifikasi Masalah dan Batasan Model

Sebagai sebuah sistem, antrian terdiri dari komponen penyusun yang terdiri dari entitas, *server*, sumber(*source*), fasilitas, dan antrian itu sendiri. Entitas berasal dari suatu sumber di luar sistem yang berfungsi sebagai input sistem antrian. Entitas yang datang ke suatu fasilitas yang tidak sibuk dapat segera dilayani oleh *server*. Sedangkan jika fasilitas sibuk, entitas menunggu di antrian. Apabila suatu tugas atau pelayanan sudah diselesaikan maka entitas yang menunggu akan segera dilayani secara otomatis. Tapi jika antrian kosong, fasilitas menjadi *idle* sampai entitas baru datang.

Banyaknya entitas yang datang ke dalam suatu sistem antrian dapat didekati dengan suatu distribusi probabilistik tertentu atau bahkan deterministik misalnya dalam suatu pengaturan atau undangan. Slain itu juga terdapat waktu antar kedatangan diantara dua entitas yang juga bersifat probabilistik atau deterministik. Entitas yang datang akan dilayani dengan memakan waktu pelayanan per entitas sesuai *service time*-nya

Implementasi sistem antrian *multi server* pada DSOL dapat dijelaskan sebagai berikut. Sistem antrian dari beberapa *server* yang menerima *customer* yang datang secara independen dan terdistribusi secara probabilistik. Apabila c*ustomer* yang datang menemukan *server* dalam keadaan *idle* maka akan langsung dilayani.

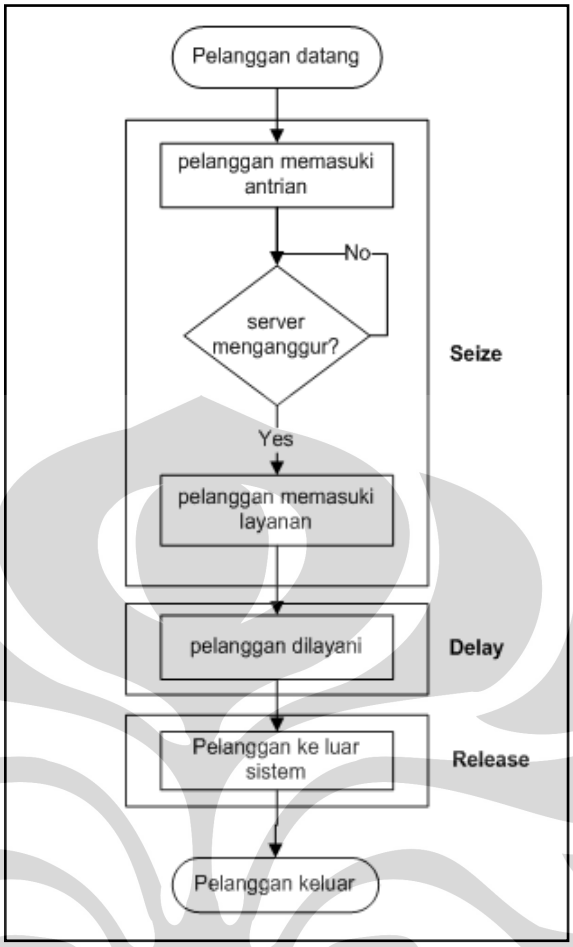
Namun apabila *customer* yang datang menemukan *server* dalam keadaan sibuk, maka *customer* tersebut akan memasuki antrian. *Customer* akan dilayani berdasarkan aturan disiplin antrian *first-in first-out* (FIFO), yaitu *customer* yang datang lebih dahulu akan dilayani lebih dulu.

Simulasi dimulai pada saat sistem dalam kondisi kosong dan *idle*. Simulasi dimulai saat *n* buah *customer* memasuki sistem hingga selesai melakukan apa yang mereka perlukan. Ada beberapa variabel yang dapat digunakan untuk mengukur *performance* dari sistem antrian ini. Variabel pertama adalah *delay* yang diharapkan *d(n)* oleh *customer* dalam antrian. Variabel kedua merupakan variabel yang dilihat dari kacamata sistem yaitu jumlah *customer* yang mengantri *q(n)*. Variabel output yang terakhir adalah utilisasi yang diharapkan dari *server* yang ada *u(n)*. Utilisasi adalah proporsi waktu *server* pada saat dalam keadaan *busy*. Karena simulasi bergantung dari variabel yang acak pada saat observasi untuk waktu antar kedatangan dan waktu pelayanan, maka variabel output *d(n)*, *q(n)*, dan *u(n)* juga akan random. Aliran proses antrian tersebut juga dapat dimodelkan terlebih dahulu dengan menggunakan *flowchart* seperti terlihat pada gambar 3.1.

Dalam diagram alir tersebut diperkenalkan tiga buah *class* yang terdapat dalam DSOL *flow library*, yaitu: *Seize*, *Delay*, dan *Release* yang mirip dengan *flow simulation*-nya aplikasi simulasi Arena. Objek *seize* digunakan ketika entitas memesan resource yang akan melayaninya dan melepaskan entitas tersebut ketika resource tersebut *idle*. Objek *delay* digunakan ketika entitas mengalami keadaan stasioner dan sedang dilayani oleh resources dalam jumlah satuan waktu tertentu yaitu selama *sevice time*. Sedangkan *release* bertujuan untuk melepaskan sejumlah entitas dan menjadi pintu keluar dari suatu lokasi ke luar sistem.

Dalam penelitian ini dibahas permasalahan antrian yang didasarkan pada asumsi berikut:

1. Satu pelayanan dan satu tahap
2. Jumlah kedatangan per unit waktu digambarkan oleh distribusi *Poisson* dengan λ = rata-rata kecepatan kedatangan



Gambar 3.1 Konseptualisasi Pada Simulasi Berorientasi Proses (Sumber: Peter H.M. Jacobs, 2004)

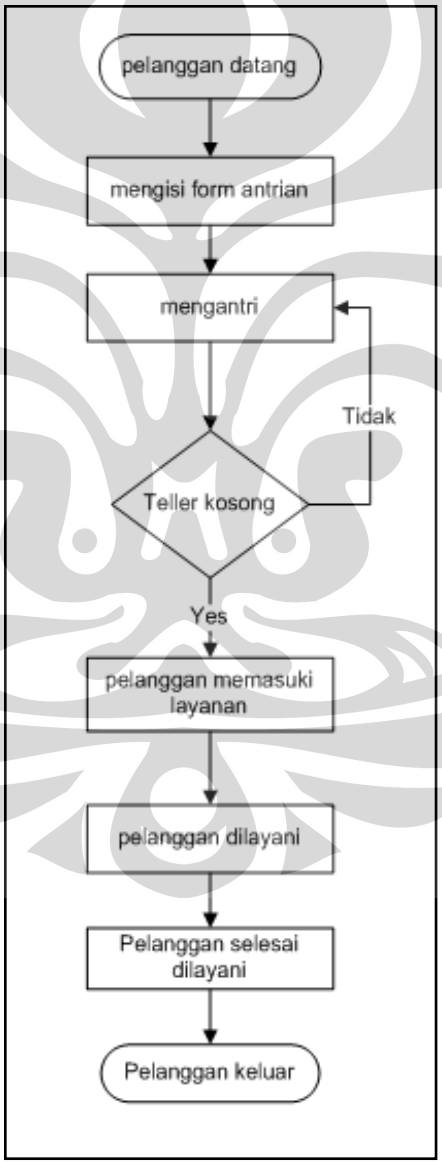
3. Waktu pelayanan eksponensial dengan μ = rata-rata kecepatan pelayanan
4. Disiplin antrian adalah *first come first served* (aturan antrian pertama datang adalah yang pertama dilayani) seluruh kedatangan dalam barisan hingga dilayani
5. Rata-rata kedatangan lebih kecil dari rata-rata waktu pelayanan

3.1.2. Memetakan Konsep Model

Langkah berikutnya adalah memetakan konsep model berdasarkan deskripsi sistem antrian yang telah dijelaskan sebelumnya. Pemetaan konsep model tersebut dilakukan dengan mnenggunakan diagram *unified modeling language* (UML) yang merupakan sebuah bahasa pemodelan grafis yang telah menjadi standar

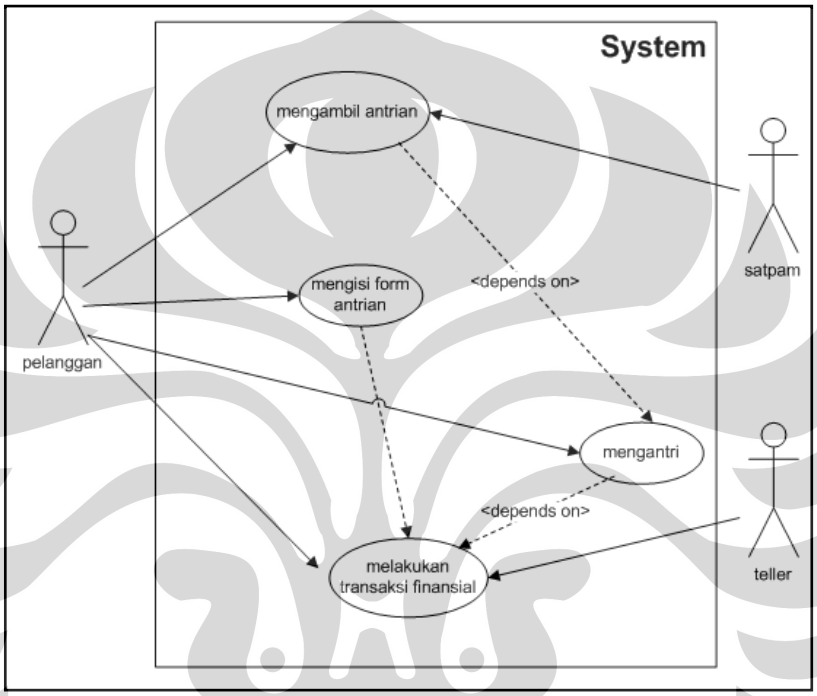
dalam industri untuk memvisualisasi, merancang, dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang sebuah sistem.

Tahapan pertama dalam membuat model dalam bentuk UML adalah membuat daftar proses dari masing-masing level serta mendefinisikan aktivitas dan proses yang mungkin timbul. Untuk mendefinisikan aktivitas yang terdapat dalam *multi server queuing system*, maka digunakan *flow chart* untuk mempermudah dalam memahami proses bisnis itu sendiri. Berikut adalah pemetaan proses bisnis yang terjadi pada sistem antrian kemudian digambarkan dalam bentuk *flow chart*.



Gambar 3.2 *Flowchart* Sistem Antrian

Tahap selanjutnya adalah memetakan aktor-aktor yang terlibat dalam sistem yang dianalisis dengan menggunakan *use case diagram* untuk tiap proses untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem. Yang ditekankan adalah apa yang diperbuat sistem, bukan bagaimana sistem berjalan. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.



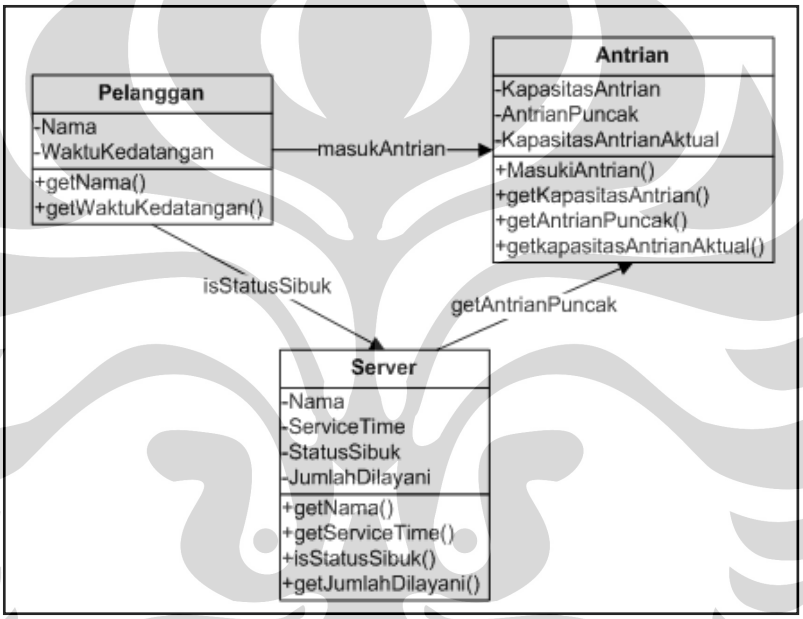
Gambar 3.3 *Use Case Diagram* Sistem Antrian

Use case diagram di atas memperlihatkan aktivitas-aktivitas yang terjadi dalam sistem yang ada, yakni sistem antrian. Garis panah putus-putus dengan keterangan *depends on* menunjukkan relasi antar aktivitas, di mana suatu aktivitas dapat dilakukan apabila aktivitas sebelumnya telah selesai dilakukan. Misalnya, kegiatan mengantri dapat dilakukan apabila aktivitas mengambil nomor antrian telah selesai dilakukan, dan seterusnya.

Dalam *use case diagram* juga diperlihatkan aktor-aktor yang berinteraksi dengan sistem yang ada untuk melakukan aktivitas yang ada. Aktor yang ada dalam model antrian ini adalah *customer*, Satpam, dan *teller*. Hal ini ditunjukkan dengan garis panah langsung dari aktor menuju aktivitas yang bersangkutan. Sehingga

dapat diperoleh informasi dari *use case diagram* di atas bahwa *customer* melakukan aktivitas mengambil nomor antrian, mengisi form, mengantri, dan melakukan transaksi finansial. Satpam bertugas melakukan aktivitas memberikan nomor antrian. *Teller* melakukan aktivitas transaksi finansial.

Langkah selanjutnya adalah membuat *class diagram* untuk memperlihatkan objek-objek yang terlibat di dalam sistem sekaligus menjelaskan keadaan (atribut/properti) suatu sistem, dan menawarkan layanan untuk memanipulasi keadaan tersebut (metode/fungsi).



Gambar 3.4 *Class Diagram* Sistem Antrian

Class diagram menggambarkan struktur dan deskripsi *class, package,* dan objek beserta hubungan satu sama lain, kebijakan, pewarisan, asosiasi, dan lain-lain. *Class* mendefinisikan sebuah tipe dari objek. Dalam model antrian pada *class diagram* di atas terdapat tiga buah *class* utama, yaitu *customer*, *queue*, dan *server*.

Relasi antar masing-masing *class* digambarkan dengan hubungan asosiasi, yaitu hubungan statis antar *class*. Umumnya hubungan ini menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah menunjukkan arah *query* antar *class*.

3.1.3 Identifikasi Variabel

Setelah mengetahui dan memahami konsep dan batasan model yang ada, maka kita dapat mendata variabel-variabel apa saja yang diperlukan untuk membentuk model antrian tersebut. Berikut ini adalah variabel-variabel tersebut beserta jenisnya:

Tabel 3.1 Daftar Variabel Di Dalam Model

No.	Nama Variabel
1	simulator.runLength
2	randomizer.seed
3	generator.startTime
4	generator.InterarrivalTime
5	generator.batchSize
6	generator.maxCreated(n)
7	server.resourceCapacity
8	server.serviceTime

Keterangan:

1. `simulator.runLength`

   Merupakan satuan waktu yang dibutuhkan dalam menjalankan simulasi

2. `randomizer.seed`

   Merupakan setiap replikasi data random

3. `generator.startTime`

   Merupakan waktu awal penciptaan suatu entitas

4. `generator.interarrivalTime`

   Merupakan distribusi waktu kedatangan antar *customer*

5. `generator.batchSize`

   Merupakan satuan unit jumlah entitas tiap kedatangan entitas

6. `generator.maxCreated`

   Merupakan jumlah entitas maksimum dalam proses simulasi keseluruhan

7. `server.resourceCapacity`

   Merupakan kapasitas entitas yang dapat diproses oleh *server* setiap satuan *service time*

8. `server.serviceTime`

   Merupakan distribusi waktu pelayanan *server*

3.2. Formulasi

Langkah selanjutnya adalah menspesifikasi model tersebut dengan memasukkan data numerik dan formulasi yang mendefinisikan perilaku sistem. Di dalam DSOL ada sejumlah *file library* yang dapat langsung digunakan oleh *user* sehingga kita tidak perlu lagi membuat *command* untuk menjalankan model yang ada. Dalam penelitian ini, digunakan beberapa *library* yang terdapat pada DSOL, antara lain:

- java.rmi.RemoteException;
- java.util.Random;
- java.net.URL;
- java.rmi.RemoteException;
- nl.tudelft.simulation.dsol.ModelInterface;
- nl.tudelft.simulation.dsol.SimRuntimeException;
- nl.tudelft.simulation.dsol.formalisms.Resource;
- nl.tudelft.simulation.dsol.formalisms.flow.Delay;
- nl.tudelft.simulation.dsol.formalisms.flow.Generator;
- nl.tudelft.simulation.dsol.formalisms.flow.Release;
- nl.tudelft.simulation.dsol.formalisms.flow.Seize;
- nl.tudelft.simulation.dsol.formalisms.flow.StationInterface;
- nl.tudelft.simulation.dsol.formalisms.flow.statistics.Utilization;
- nl.tudelft.simulation.dsol.simulators.DEVSSimulatorInterface;
- nl.tudelft.simulation.dsol.simulators.SimulatorInterface;
- nl.tudelft.simulation.dsol.statistics.Tally;
- nl.tudelft.simulation.dsol.statistics.charts.BoxAndWhiskerChart;
- nl.tudelft.simulation.jstats.distributions.DistConstant;
- nl.tudelft.simulation.jstats.distributions.DistContinuous;
- nl.tudelft.simulation.jstats.distributions.DistDiscreteConstant;
- nl.tudelft.simulation.jstats.distributions.DistExponential;
- nl.tudelft.simulation.jstats.streams.StreamInterface;
- nl.tudelft.simulation.dsol.experiment.Experiment;
- nl.tudelft.simulation.dsol.gui.DSOLApplication;
- nl.tudelft.simulation.dsol.gui.panels.GUIExperimentParsingThread;

- nl.tudelft.simulation.event.EventInterface;
- nl.tudelft.simulation.event.EventListenerInterface;
- nl.tudelft.simulation.language.io.URLResource;
- nl.tudelft.simulation.xml.dsol.ExperimentParsingThread;

Dari variabel-variabel yang diketahui terdapat dalam sistem maka dapat dikatakan sistem tersebut memiliki nilai yang dapat di-*customize* oleh pengguna sehingga memungkinkan simulasi dapat dilakukan dalam berbagai keadaan dengan suatu sistem tertentu, dalam hal ini adalah *multi server queuing system*. Dari nilai yang dimasukkan kemudian dapat dilakukan replikasi dan diberikan *treatment* tertentu untuk kemudian dianalisis dan dilihat nilai *output* yang diberikan.

Tabel 3.2 Daftar Variabel Di Dalam Model Beserta Nilainya

Deskripsi	Nilai
simulator.runLength	Double.MAXVALUE
randomizer.seed	555
generator.startTime	0
generator.interarrivalTime	EXPO(1.0)
generator.batchSize	1
generator.maxCreated(n)	1000
server.resourceCapacity	1
server.serviceTime	EXPO(0.5)

Keterangan:

1. `simulator.runLength = Double.MAXVALUE`

   Merupakan satuan waktu yang dibutuhkan dalam menjalankan simulasi berbentuk data integer yang besarnya merupakan nilai bertipe data double maksiumum

2. `randomizer.seed = 555`

   Data dapat dirandom sebanyak 555 kali

3. `generator.startTime`

   Merupakan waktu dimulainya simulasi

4. `generator.interarrivalTime`

   Merupakan waktu kedatangan antar *customer* terdistribusi secara *exponensial* dengan jarak 1 satuan waktu

5. `generator.batchSize`

   Entitas tiap kedatangan yang masuk ke dalam sistem sebesar satu unit

6. `generator.maxCreated`

   Jumlah entitas maksimum dalam proses simulasi sebanyak 1000 unit sehingga apabila *input* entitas yang memasuki sistem sudah sebanyak 1000 unit maka simulasi akan berhenti

7. `server.resourceCapacity`

   Kapasitas entitas yang akan diproses oleh *server* sebesar satu unit

8. `server.serviceTime`

   Waktu pelayanan *server* terdistribusi eksponensial dengan jarak 0,5 satuan waktu

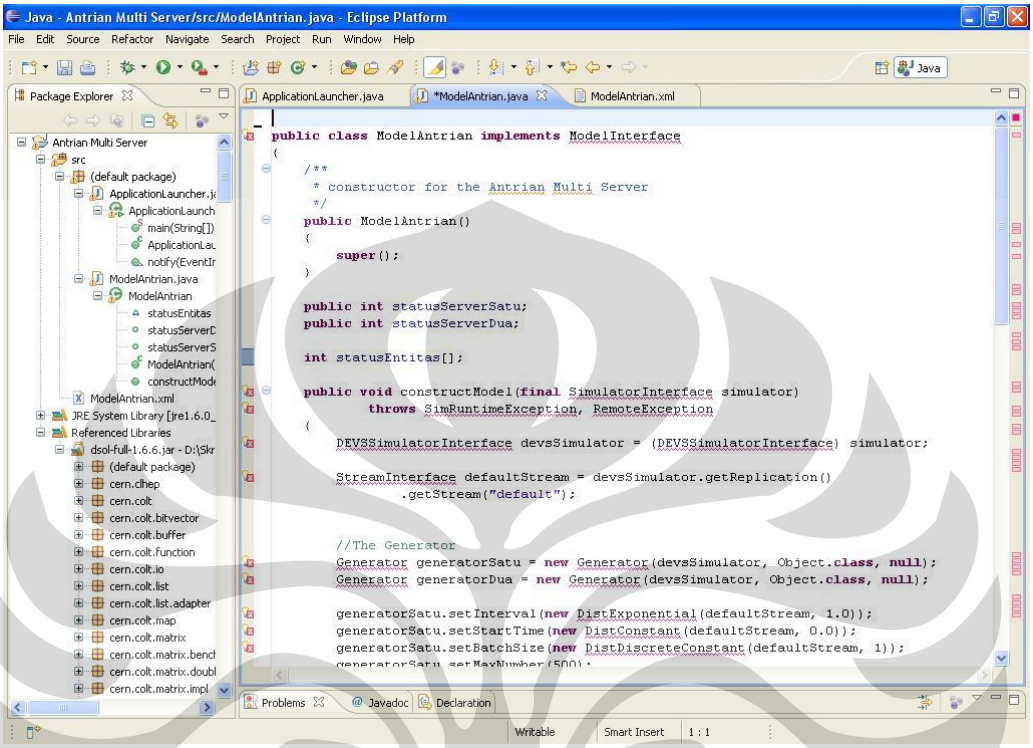
Setelah menentukan variabel model beserta nilainya, maka dibuatlah kode dalam bentuk *file* Java. Untuk mempermudah proses kompilasi dan penggunaan *classpath* maka digunakan IDE (*integrated development environment*), yang pada penelitian ini menggunakan IDE Eclipse.

3.3. Verivikasi Model

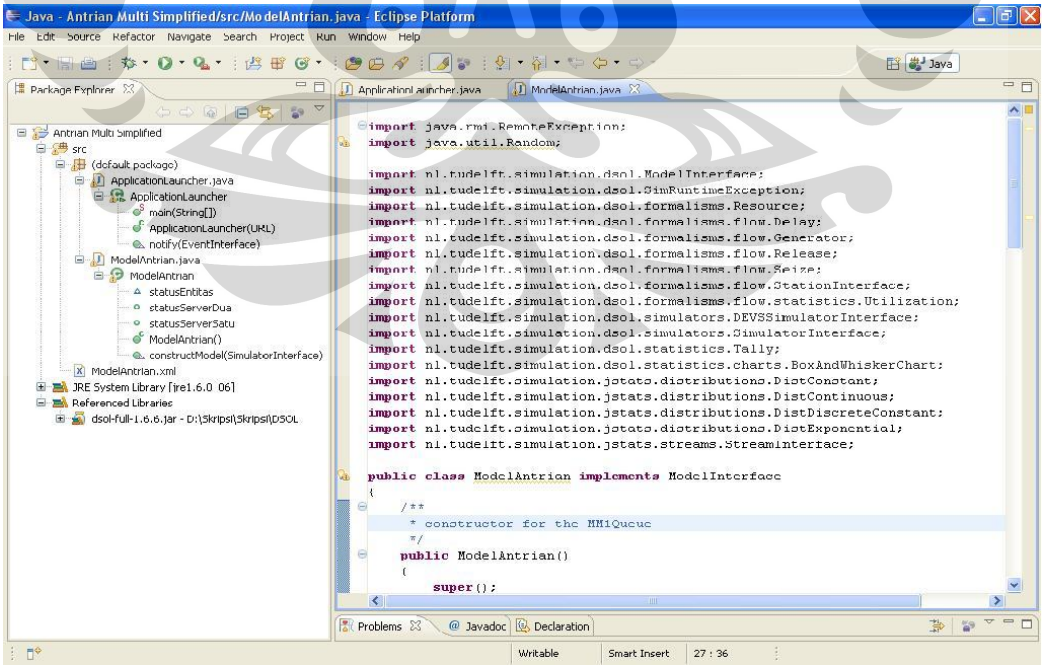
Tujuan dari tahap verifikasi adalah menguji apakah model yang dibuat sudah benar sehingga simulasi model dapat dijalankan tanpa terjadi kesalahan (*error*). Verifikasi dapat dilakukan dengan melihat indikator kesalahan pada formulasi model. Salah satu kemudahan yang didapat dengan membuat model menggunakan IDE Eclipse yaitu adanya indikator *error* pada variabel maupun *link* yang bermasalah berupa tanda merah, seperti contoh pada gambar 3.4. Sehingga kita dapat mendeteksi dan melakukan verifikasi tanpa menjalankan model.

Kesalahan bisa terjadi pada formulasi model, yakni akibat perbedaan unit antara variabel yang berhubungan, maupun pada struktur model karena terdapat variabel maupun *link* yang kurang. Formulasi ini tentunya dibuat sesuai dengan model konseptualnya. Setelah verifikasi terhadap formulasi tidak terdapat *error*, maka selanjutnya adalah membandingkan data model dengan konseptualnya.

Apabila ada kesalahan sedikit saja pada model maka model tidak akan berjalan sebagaimana mestinya. Kesalahan juga dapat dideteksi apakah suatu model dapat segera berjalan di dalam DSOL GUI atau tidak.



Gambar 3.5 Indikator Error pada Formulasi Model



Gambar 3.6 Indikator Error pada Formulasi Model Yang Sudah Diperbaiki

4. ANALISIS

4.1. Analisis Sistem Simulasi Berorientasi Objek

Cara pandang simulasi berorientasi objek memberikan kemudahan dalam mengembangkan model suatu sistem terutama sistem yang memiliki kompleksitas yang tinggi. Dengan pendekatan simulasi berorientasi objek kompleksitas atau masalah akan di-*break down* ke dalam satuan-satuan atau modul yang lebih kecil. Satuan-satuan tersebut berupa objek-objek yang merupakan komponen penyusun sistem sehingga dengan memfokuskan diri pada satuan yang lebih kecil maka masalah tersebut bisa lebih mudah dikelola.

Untuk mengkomunikasikan atau memvisualisasikan objek-objek tersebut maka digunakanlah *Unified Modeling Language* (UML) yang merupakan notasi bahasa grafis yang sudah menjadi standar pengembangan perangkat lunak. UML menghasilkan cetak biru dan sketsa, meliputi bagian konseptual, semisal proses bisnis dan fungsi sistem, serta bagian kongkrit, semisal kelas objek yang dinyatakan dalam bahasa pemrograman dan skema basis data. Penelitian ini mengembangkan arsitektur simulasi yang menerapkan paradigma simulasi berorientasi objek (*object oriented simulation*, OOS) pada pemrograman dan UML pada pemodelan. Dengan menggunakan kerangka kerja tersebut dan sistem model antrian sebagai domain, penelitian ini menghasilkan prototipe simulasi berorientasi objek untuk memodelkan simulasi antrian. Pada tahap perencanaan dan perancangan, simulasi berorientasi objek mendekomposisi sistem menjadi beberapa modul.

4.1.1. Desain Simulasi Berorientasi Objek

Di dalam merancang simulasi berrientasi objek diperlukan pemahaman mengenai konsep objek dan *class*. Suatu objek di dalam UML digambarkan dalam suatu *blueprint* yang bernama *class*. *Class* berfungsi sebagai desain suatu objek dan menjadi abstraksi keseluruhan objek yang dapat dibangun dari *class* tersebut. Abstraksi tersebut mendefinisikan *variable* dan *method* yang akan dimiliki oleh objek yang akan dibuat dari *class* tersebut. Sementara itu objek merupakan bangunan yang dihasilkan dari suatu kelas dan biasa disebut suatu *class instance*.

Analogi class dan objek sama dengan analogi desain bangunan rumah dengan objek rumah sendiri. Desain rumah merupakan abstraksi dari rumah secara keseluruhan dan mendefinisikan apa-apa saja yang akan dimiliki rumah tersebut ketika dibuat *(variable)* dan aksi yang bisa dilakukan rumah tersebut *(method)*. Dari gambar desain rumah tersebut maka dapat dibangun objek rumah yang berbeda-beda dan bisa lebih khusus definisi *variable* dan *method*-nya. Jadi objek dapat bersifat lebih unik dan spesifik disbanding dengan desainnya. Seperti sudah dijelaskan di atas, dalam mendesain model simulasi berorientasi objek menggunakan *Unified Modeling Language* (UML) untuk memvisualisasikan kerangka *software* yang akan bekerja. Dalam penelitian ini, berikut beberapa langkah dalam memodelkan sistem antrian ke dalam UML, yaitu:

1. Membuat daftar proses bisnis dari level tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin timbul. Dalam penelitian ini proses bisnis yang terjadi dalam sistem antrian didefinisikan ke dalam bentuk *flow chart*.
2. Memetakan *use case* untuk tiap proses bisnis untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.
3. Berdasarkan model-model yang sudah ada, setelah itu dibuat *class diagram*. *Class diagram* menggambarkan struktur dan deskripsi *class*, *package*, dan objek beserta hubungan satu sama lain. *Class* mendefinisikan sebuah tipe dari objek. Setiap *package* atau *domain* dipecah menjadi *hierarki class* lengkap dengan atribut dan metodenya.
4. Memulai membangun sistem dengan *software* yang ada, Pada penelitian ini digunakan DSOL sebagai media untuk men-*develop* model yang sudah dibuat.
5. Melakukan uji modul dan uji integrasi serta memperbaiki model beserta kode programnya. Model harus selalu sesuai dengan kode program aktualnya. Dalam hal ini verifikasi model dilakukan terhadap nilai perhitungan secara manual.

4.1.1.1. Enkapsulasi

Enkapsulasi atau pengkapsulan merupakan konsep yang dikenal luas dalam *object orientation paradigm.* Enkapsulasi memungkinkan suatu objek dimodelkan dalam suatu modul-modul karena setiap objek akan memiliki definisi batasan yang membedakannya dari objek lain. Enkapsulasi juga memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam dari sebuah objek dengan cara yang sembarangan. Hanya metode dalam objek tersebut yang diberi ijin untuk mengakses keadaannya saja bisa. Enkapsulasi dapat dicapai dengan mendefinisikan *access modifier* terhadap *class* dan *class members* bersangkutan. Kata kunci atau *syntax* yang sering digunakan adalah *public*, *protected*, dan *private*. Kata kunci tersebut menentukan apakah mereka tersedia untuk semua *class*, *subclass*, atau hanya untuk *class* tertentu saja. Sehingga enkapsulasi dapat menyembunyikan detail secara jelas bagaimana keterangan sebuah *class* bekerja dari suatu objek dan dapat menggunakan kode yang terdapat di dalamnya atau mengirimkan suatu pesan.

Definisi dari sebuah *class* akan menspesifikkan properti dari suatu objek, yaitu karakteristik sesuatu (atribut) dan apa yang sesuatu tersebut dapat lakukan (metode).

4.1.1.2. Komunikasi Antar Objek

Interaksi dan komunikasi antar objek dilakukan dengan saling mengirimkan pesan diantara para objek yang bersangkutan. Hal tersebut bisa dilakukan dengan cara memanggil *method* suatu objek. Pemanggilan tersebut bisa secara langsung dilakukan atau pun dengan menambahkan suatu *argument* ketika komunikasi tersebut menginginkan hasil yang lebih spesifik. Sebuah objek dapat mengirimkan pesan kepada objek lain dan menerima respon dari objek lain pula. Dalam penelitian kali ini, ketika *customer* datang dalam sebuah sistem dan meminta untuk dilayani, maka *customer* akan mengirimkan pesan bahwa ia telah datang dan menunggu untuk dilayani oleh *server*

Contoh komunikasi antar objek data digambarkan sebagai berikut. Di dalam Java ada sebuah objek yang bertugas menangani perhitungan matematika umum, objek

tersebut memiliki nama *Math.* Untuk berkomunikasi dengan objek tersebut maka kita harus memanggil method yang dimilikinya. Misalkan saja kita menginginkan objek *Math* tersebut untuk menghitung akar kuadrat dari 900.0. maka cara pemanggilannya adalah:

```
Math.sqrt( 900.0 )
```

Setelah dipanggil maka objek Math akan mengirimkan hasilnya secara langsung. Untuk melihat hasilnya dapat digunakan perintah berikut ini yang akan memperlihatkan hasil 30.0 di *command line*:

```
System.out.println(Math.sqrt(900.0));
```

4.1.1.3.Pembentukan Objek

Objek dapat dibangun dengan membangunnya dari *class* yang membangunnya. Di dalam Java, sintaks untuk membuat objek adalah dengan ***new.*** Misalkan kita berkeinginan membangun suatu mobil dari desain mobil yang ada, maka perintahnya dapat ditulis sebagai berikut:

```
mobil mobilBagus = new mobil();
```

Perintah tersebut akan langsung menghasilkan sebuah objek mobil yang memiliki nama atau referensi *mobilBagus.*

4.1.2. Perbedaan Simulasi Berorientasi Objek dengan Simulasi Berorientasi Proses

Perancangan simulasi suatu sistem dapat dilakukan setidaknya dengan memakai pendekatan berorientasi proses dan pendekatan berorientasi objek. Pada perancangan simulasi berorientasi objek memang difokuskan pada pembuatan objek itu sendiri kemudian membuat bagaimana objek tersebut dapat bekerja dan berinteraksi dengan objek lain. Sedangkan pada perancangan simulasi berorientasi proses menekankan pada aliran (*flow*) sebuah kegiatan, bagaimana suatu sistem berjalan menurut urutan proses tertentu. Sehingga meskipun *output* yang diinginkan dari suatu simulasi sama tetapi dapat dilakukan dengan pendekatan yang berbeda seperti yang sudah dijelaskan sebelumnya. Adapun *output* umum

dari penggunaan simulasi itu sendiri adalah ingin membangun sebuah pemahaman dari tingkah laku yang ada pada sistem dengan menggambarkan hubungan ketergantungan dan varian yang kompleks.

4.1.2.1. Metodologi

Simulasi secara umum dibagi atas tahap konseptualisasi, formulasi, verifikasi, dan validasi. Baik metode simulasi berorientasi proses maupun simulasi berorientasi objek sama-sama mengalami tahapan dalam simulasi tersebut. Namun perbedaannya terdapat pada proses yang terjadi dalam setiap tahapan itu sendiri. Proses itu sendiri melibatkan *tools* yang spesifik di mana penggunaan *tools* itu sendiri yang akan membedakan pendekatan kedua buah metode simulasi tersebut karena akan menghasilkan model yang berbeda. Dalam mengembangkan model dari proses pembuatan hingga sampai model tersebut dapat berjalan terjadi dalam tahapan konseptualisasi dan formulasi.

Untuk tahapan verifikasi yang bertujuan untuk menentukan apakah model sudah berjalan sesuai dengan yang diinginkan dan tahapan validasi yang bertujuan untuk menentukan apakah model yang dibuat telah menggambarkan sistem yang sesungguhnya, kedua buah metode yang sudah dijelaskan sebelumnya tidak memiliki perbedaan yang substansial secara proses, karena hanya melakukan perbaikan dan penyempurnaan pada model yang ada dengan formula yang sudah digunakan pada tahap formulasi.

1. Simulasi Berorientasi Proses

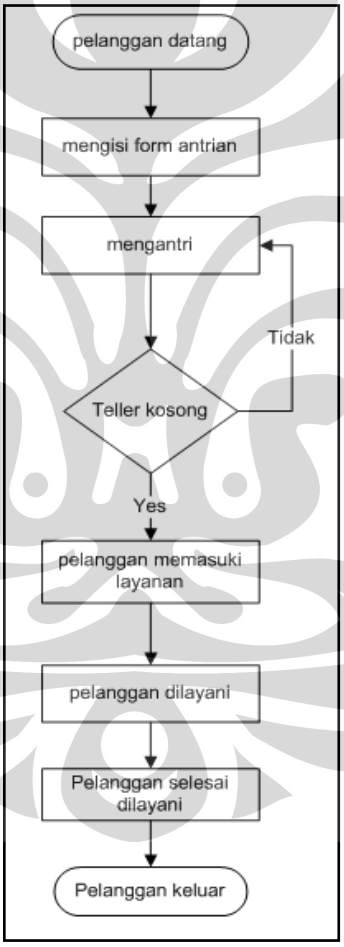
   Sistem berorientasi proses bertujuan untuk merancang dan membangun sistem melalui pendekatan aliran proses mengenai bagaimana sistem tersebut berjalan.

   a. Konseptualisasi

      Yang pertama dilakukan adalah menentukan tujuan, ruang lingkup, dan prasyarat yang dibutuhkan untuk membuat batasan model yang akan dibuat. Semakin luas akan semakin baik, karena akan mempertimbangkan faktor yang berkaitan, namun akan semakin sulit dalam penyusunan modelnya.

Setelah mendefinisikan batasan masalah, maka dibuatlah model konseptual berdasarkan proses-proses yang terjadi dalam sistem tersebut. Dalam simulasi berorientasi proses, model dibuat berdasarkan proses-proses yang terjadi secara berurutan. Hal ini memerlukan pehamaman berfikir secara sistematis agar model yang dibuat benar-benar menggambarkan urutan aktivitas yang terjadi.

Model yang dibuat setelah pemetaan urutan aktivitas dapat berupa *flow chart* atau diagram IDEF0 yang dikenal secara luas. Berikut adalah contoh model berdasarkan proses dari sistem antrian dalam penelitian ini.



Gambar 4.1 Konseptualisasi Pada Simulasi Berorientasi Proses

Dengan menggunakan *flow chart* atau diagram IDEF0 kita dapat melihat penggambaran bagaimana sistem tersebut berjalan sehingga mempermudah untuk mensimulasikan sistem tersebut dengan model yang

ada. Perlu diingat bahwa model yang dibuat harus merepresentasikan sistem yang ada. Sehingga untuk sistem yang kompleks juga memerlukan model yang kompleks pula, sehingga hasil yang disimulasikan memiliki kemiripan dengan hasil dari sistem yang ada.

b. Formulasi

Dalam penjelasan berikut formulasi yang dilakuan akan mengambil contoh formulasi dari *software* ProModel yang sering digunakan untuk mensimulasikan model yang dibuat dari *flow chart* atau diagram IDEF0[1].

Formulasi dilakukan dengan menspesifikasi model yang ada dengan memasukkan data numerik yang mendefinisikan perilaku sistem. Adapun pada tahapan formulasi dilakukan:

- Menambahkan elemen yang terdapat dalam struktur seperti *entity, location, resources, processing*, dan sebagainya
- Menambahkan elemen prosedural seperti operasi, kedatangan, dll.
- Menambahkan permasalahan lainnya jika ada

2. Simulasi Berorientasi Objek

Sistem berorientasi objek bertujuan untuk merancang dan membangun sistem melalui perakitan objek perangkat lunak yang digunakan, daripada menulis *programming language* dari awal. Pada perancangan simulasi berorientasi objek yang ditekankan adalah apa yang diperbuat sistem, bukan bagaimana sistem berjalan.

Empat langkah untuk melakukan desain berorientasi objek:

- Mengidentifikasi *class* dari objek
- Mengidentifikasi hubungan diantara *class* tersebut
- Mengidentifikasi atribut atau *variable* utama serta *method class* tersebut
- Menentukan hubungan penerimaan dan membangun hirarki kelas

[1] Harrel, Charles, Biman K. Ghosh, dan Royce Bowden, *Simulation Using ProModel*, Mc Graw Hill, 2000, hal.81.

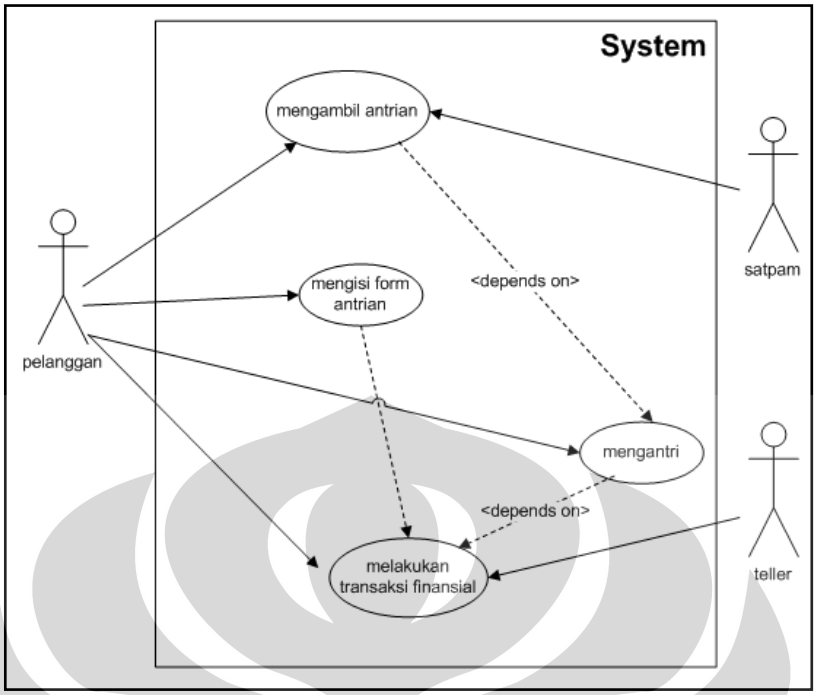
a. Konseptualisasi

Setelah mendefinisikan batasan masalah, maka dibuatlah model konseptual berdasarkan aktivitas yang terjadi dalam sistem tersebut. Dalam simulasi berorientasi objek, model dibuat berdasarkan objek-objek yang saling berinteraksi dalam membentuk sistem.

Sistem berorientasi objek memiliki beberapa elemen utama, yaitu:

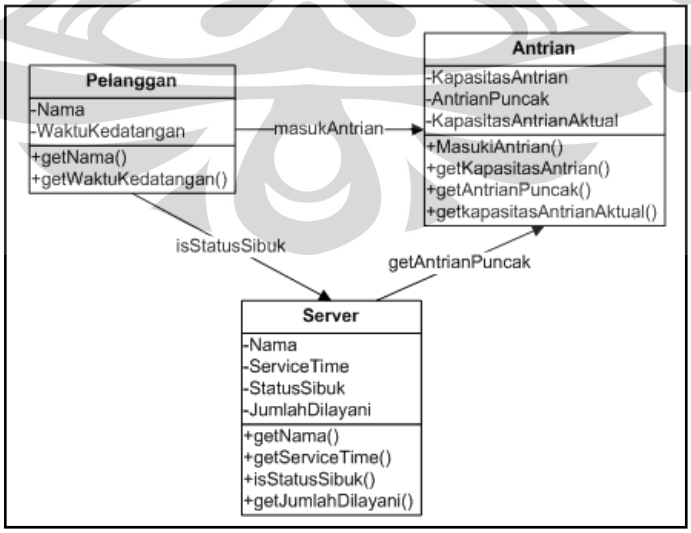
- Objek
  - Semua yang dihadapi dalam lingkungan
  - Mempunyai tingkah laku tertentu
  - *Object-Oriented Programming*
  - Desain sistem dipusatkan pada identifikasi objek lebih dari menspesifikasikan atribut dan kode program untuk memanipulasinya
- *Class*
  - Seperangkat objek yang terbagi atas struktur biasa dan tingkah laku
  - Sebuah jenis; sebuah objek tunggal
  - Contoh: *class* pelanggan dapat dimasukan dua contoh, yaitu objek pelanggan agen dan objek pelanggan eceran

Dalam membuat model perancangan simulasi berorientasi objek yang pertama dilakukan adalah memetakan *use case* untuk tiap proses bisnis untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. Berikut adalah contoh model *use case diagram* berdasarkan proses dari sistem antrian dalam penelitian ini.



Gambar 4.2 Konseptualisasi Pada Simulasi Berorientasi Objek Dengan *Use Case Diagram*

Berdasarkan model-model yang sudah ada, setelah itu dibuat *class diagram*. *Class diagram* menggambarkan struktur dan deskripsi *class*, *package*, dan objek beserta hubungan satu sama lain. *Class* mendefinisikan sebuah tipe dari objek. Setiap *package* atau *domain* dipecah menjadi *hierarki class* lengkap dengan atribut dan metodenya.



Gambar 4.3 Konseptualisasi Pada Simulasi Berorientasi Objek Dengan *Class Diagram*

Relasi antar masing-masing *class* digambarkan dengan hubungan asosiasi, yaitu hubungan statis antar *class*. Umumnya hubungan ini menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah menunjukkan arah *query* antar *class*.

b. Formulasi

Formulasi dilakukan dengan membangun sistem dengan *software* yang ada, yang dalam penelitian ini digunakan DSOL sebagai media untuk men-*develop* model yang sudah dibuat. Adapun formulasi yang dibuat berupa *coding* untuk men-*generate* model berdasarkan *class diagram* yang sudah dibuat. Dalam simulasi dengan menggunakan DSOL ada sejumlah *file external library* yang dapat langsung digunakan oleh *user* sehingga kita tidak perlu lagi membuat *command* untuk menjalankan model yang ada. Dalam penelitian ini, digunakan beberapa *library* yang terdapat pada *dsol-full-1.6.6.jar*. Berikut adalah contoh beberapa formulasi yang dilakukan pada penelitian ini.

Tabel 4.1 Formulasi pada Simulasi Berorientasi Objek

Deskripsi	Nilai
simulator.runLength	Double.MAXVALUE
randomizer.seed	555
generator.startTime	0
generator.interarrivalTime	EXPO(1.0)
generator.batchSize	1
generator.maxCreated(n)	1000
server.resourceCapacity	1
server.serviceTime	EXPO(0.5)

4.1.2.2. Orientasi Rancangan

1. Modularitas

Modularity adalah derajat dari standardisasi dan kebebasan modul, serta variasi dalam penggunaan suatu rancangan model.

- Orientasi proses

  Modul adalah sebuah unit dari kode programa linier yang melakukan sebuah fungsi yang didefinisikan dengan baik.

- Orientasi objek

  Modul adalah sebuah objek yang menggabungkan atribut dan kode program untuk berlaku dalam suatu cara tertentu

2. Desain

- Orientasi proses

  Pada simulasi berorientasi proses menggunakan pendekatan desain *top-bottom* di mana desain dimulai dengan sebuah deskripsi abstrak kemudian menyaring pada langkah berikutnya.

- Orientasi objek

  Pada simulasi berorientasi objek, desain yang dipakai adalah *bottom-up*. Semua level terkecil pada sistem, yang disebut objek dibuat sehingga sistem yang ada dapat dijalankan dengan menginteraksikan objek-objek yang ada tersebut.

4.1.2.3. Kelebihan Simulasi Berorientasi Objek

Simulasi berorientasi objek merupakan teknologi baru yang belum populer pada pembuatan simulasi di tempat di mana penulis melakukan penelitian. Pada keadaan tertentu simulasi dengan menggunakan metode berorientasi objek merupakan suatu pilihan tepat, tapi pada keadaan yang salah pilihan ini dapat berakibat besar. Keuntungan dari menggunakan *object-oriented simulation* adalah[2]:

[2] Chell A. Roberts dan Yasser M. Dessouky, "An Overview of Object-Oriented Simulation", Simulation Councils, 1998, hal.360.

1. Salah satu keuntungan menggunakan simulasi berorientasi objek adalah kemampuan untuk memodelkan sistem dengan menggunakan entitas yang bersifat natural terhadap sistem itu sendiri. Sehingga model yang dibuat juga diharapkan memiliki perilaku yang sama terhadap sistem aslinya.
2. Dengan menggunakan simulasi berorientasi objek memungkinkan pengembangan suatu sistem dapat berjalan lebih cepat, karena untuk melihat *behavior* dari suatu sistem dapat dilihat dari karakteristik masing-masing objek yang ada. Sehingga apabila ingin meng-*improve* kinerja dari suatu sistem dapat disimulasikan objek mana yang paling berpengaruh secara signifikan terhadap peningkatan hasil yang diperoleh.
3. Desain simulasi berorientasi objek dapat meningkatkan kualitas dari sistem itu sendiri karena entitas yang digunakan bersifat natural.
4. Pada desain dengan menggunakan simulasi berorientasi objek memudahkan *user* dalam melakukan modifikasi. Modifikasi dapat dilakukan pada tataran objek hingga interaksi yang terjadi antar objek itu sendiri yang membuat sistem berjalan.
5. Model dari sistem yang dibuat dapat dipakai kembali (*reusability*) untuk desain pada sistem yang berbeda dengan menggunakan bahasa objek yang memiliki kemiripan baik dalam atribut maupun metodenya.

4.1.2.4. Kelemahan Simulasi Berorientasi Objek

Pada desain simulasi berorientasi objek juga memiliki beberapa kelemahan, di antaranya:

1. Produk simulasi berorientasi objek yang ada masih kebanyakan diorientasikan untuk digunakan dalam bahasa Java dan C++. Di samping itu juga belum banyak tersedia komponen untuk pengaksesan simulasi berorientasi objek untuk bahasa pemrograman lainnya. Walaupun simulasi berorientasi objek dapat diimplementasikan dalam Java yang *platform independent*, belum tentu pada *platform* yang kita miliki mendukung implementasi Java. Selain itu Java juga memiliki aneka nuansa dan keanehan-keanehan bila diterapkan pada lingkungan yang berbeda.

2. Cara penyimpanan dan pengaksesan data pada simulasi berorientasi objek sangat spesifik. Sehingga apabila kita menggunakan simulasi berorientasi objek maka kita harus berkomitmen untuk terus menggunakan simulasi berorientasi objek. Setelah mengimplementasikan simulasi berorientasi objek, sangat sulit untuk kembali kepada metode lain.
3. Dalam menggunakan simulasi berorientasi objek dibutuhkan keterampilan khusus, di mana selain mampu berpikir untuk membuat model simulasi yang baik juga dibutuhkan kemampuan pemrograman yang baik dalam menjalankan model dari sistem yang dibuat.

4.2. Analisis Pengerjaan Dengan Menggunakan Dsol

Pada aplikasi model dengan menggunakan DSOL dilakukan beberapa langkah pengerjaan hingga model dapat dijalankan. Tahapan-tahapan yang dilakukan sebelum men-*develop* model sehingga model dapat dijalankan bisa dilihat pada konseptualisasi di Bab 3. Pada bab tersebut dijelaskan pembuatan model *use case diagram* dan *class diagram* yang dipakai sebagai dasar pada penggunaan DSOL sebagai media simulasi sistem antrian.

4.2.1 Pembuatan Simulasi dari Model

Ada beberapa *tools* atau *software* yang dibutuhkan dalam pembuatan model simulasi pada penelitian ini yang sudah harus terpasang di komputer. *Tools* tersebut semuanya gratis dan bersifat *open source* yaitu:

1. Java

   Java merupakan tools utama yang dibutuhkan dan merupakan fondasi pertama agar model dapat berjalan. Tanpa terpasang-nya Java maka sehebat apapun model yang dibuat tidak akan berjalan. Untuk menjalankan sekaligus mengembangkan model dibutuhkan *Java Development Kit* (JDK) sementara jika hanya ingin menjalankan model yang sudah benar saja maka hanya dibutuhkan *Java Runtime Environment* (JRE)
2. Eclipse IDE

   Eclipse IDE merupakan *software* untuk membuat *software* terutama *software* yang dibuat dalam bahasa Java. *Tools* tersebut merupakan alat

sebenarnya dalam mengembangkan sebuah aplikasi yang dalam hal ini model yang dibuat

3. DSOL

   DSOL berfungsi sebagai *library* simulasi. Library tersebut dikembangkan oleh ahli simulasi dari TU Delft sehingga pembuat model simulasi hanya perlu menggunakan *library* tersebut tanpa harus membangun aplikasi simulasi dari awal

Pembuatan model simulasi dengan menggunakan DSOL memiliki beberapa persyaratan kemampuan minimal yang harus dipenuhi. Keahlian yang diperlukan diantaranya pembuat model harus mengerti bahasa pemrograman Java sampai ke tingkat *intermediate*. Pemahaman dasar Java saja belum mencukupi karena penggunaan DSOL untuk mengembangkan suatu model melibatkan penggunaan *library* sendiri di luar Java API sehingga cukup susah bagi *programmer* dasar.

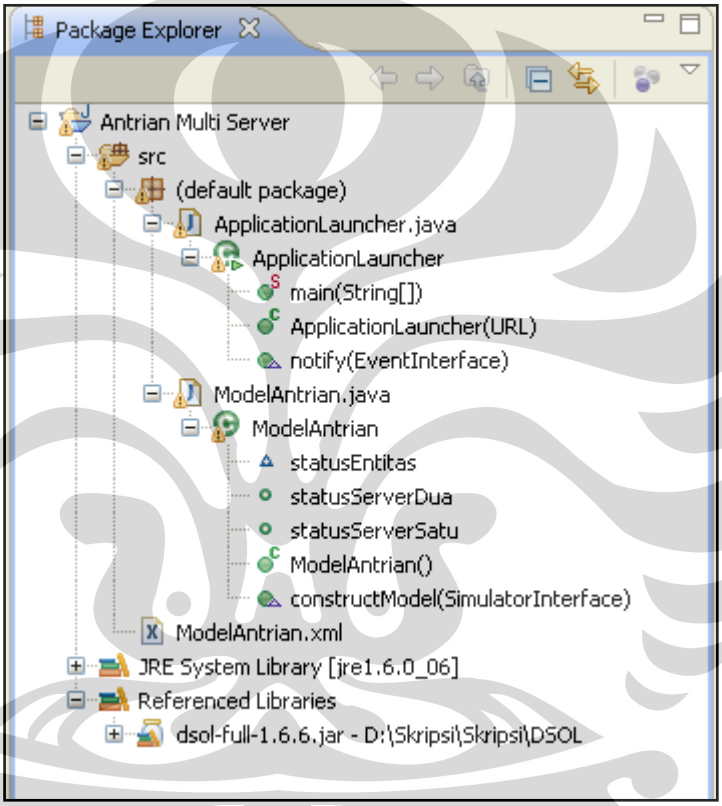
Dalam mengembangkan model dengan menggunakan DSOL, pembuat model simulasi tidak dapat begitu saja menginteraksikan objek-objek yang telah tersedia pada DSOL. Objek-objek yang dibutuhkan untuk simulasi dapat didapat dari *library* DSOL, sedangkan untuk menggunakan dan menginteraksikan objek tersebut harus menggunakan JAVA. Terkadang interaksi antar objek yang terjadi tidak berjalan secara semestinya akibat *coding* JAVA yang dipakai tidak sesuai dengan fungsi yang terdapat dalam *library* DSOL.

Adapun langkah pembuatan model simulasi dapat diterangkan sebagai berikut:

1. Membuka Eclipse IDE dan membuat suatu *Java Project* baru
2. Mendefiniskan *external library project* yang kita buat. Karena proyek yang dibuat membutuhkan library DSOL , maka pada *project properties* harus didefinisikan tempat *DSOL library* berada
3. Membuat beberapa *class* yang dibutuhkan sebagai komponen model yang dibuat. Untuk model antrian pada penelitian ini penyusun membuat dua buah *class* dan ditempatkan dalam dua buah *file* berekstensi *.java,* yaitu *ApplicationLauncher.java* dan *ModelAntrian.java*

4. Membuat XML *file* yang berfungsi mendeskripsikan eksperimen model yang dibuat seperti nama pengembang model, *warm up period*, *runLength*, *startTime*, dan sebagainya
5. Mengembangkan model lebih lanjut sesuai

   Model dibuat dengan menggunakan library simulasi DSOL dengan menuliskan source code di dalam lingkungan Eclipse. Berikut screenshoot untuk hasil dari langkah-langkah tadi



Gambar 4.4 Tampilan Class dan File yang Dibutuhkan Dalam Pembuatan Model

Berikut merupakan *source code* model antrian *multi server* untuk class model utama yaitu *ModelAntrian.java* dengan menggunakan DSOL tanpa *import statement*:

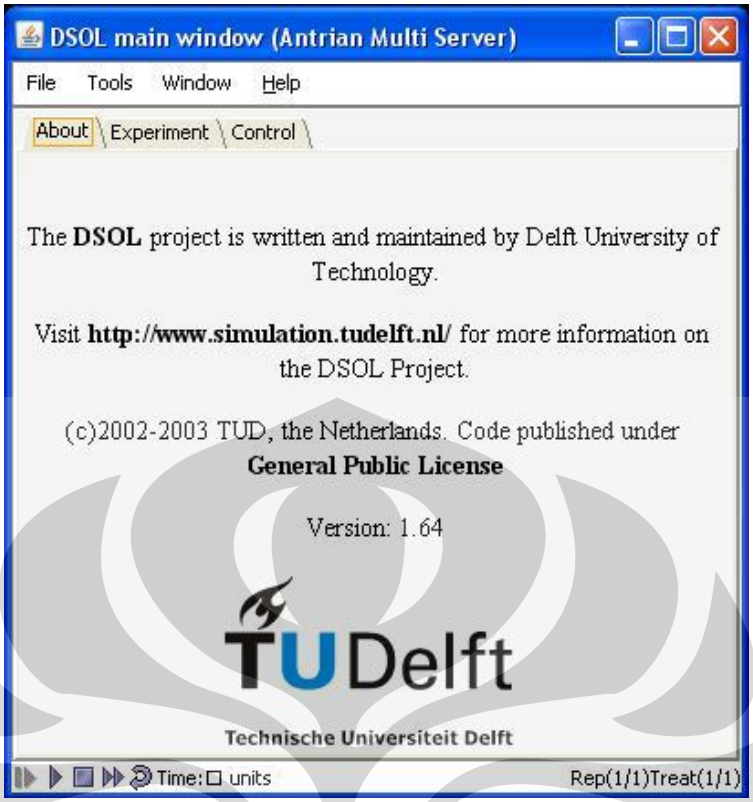
```
public class ModelAntrian implements ModelInterface
{
	/**
	 * constructor for the Antrian Multi Server
	 */
	public ModelAntrian()
```

```
{
      super();
}

public int statusServerSatu;
public int statusServerDua;

int statusEntitas[];

public void constructModel(final SimulatorInterface
simulator)
            throws SimRuntimeException, RemoteException
{
      DEVSSimulatorInterface devsSimulator =
      (DEVSSimulatorInterface) simulator;

      StreamInterface defaultStream =
      devsSimulator.getReplication()
            .getStream("default");


      //The Generator
      Generator generatorSatu = new Generator(devsSimulator,
      Object.class, null);

      Generator generatorDua = new Generator(devsSimulator,
      Object.class, null);

      generatorSatu.setInterval(new
      DistExponential(defaultStream, 1.0));
      generatorSatu.setStartTime(new
      DistConstant(defaultStream, 0.0));
      generatorSatu.setBatchSize(new
      DistDiscreteConstant(defaultStream, 1));
      generatorSatu.setMaxNumber(500);

      generatorDua.setInterval(new
      DistExponential(defaultStream, 1.0));
      generatorDua.setStartTime(new
      DistConstant(defaultStream, 0.0));
      generatorDua.setBatchSize(new
      DistDiscreteConstant(defaultStream, 1));
      generatorDua.setMaxNumber(500);



      //The queue, the resource and the release
      Resource resource = new Resource(devsSimulator, 1.0);
      Resource resourceDua = new Resource(devsSimulator,
      1.0);

      // created a resource
      StationInterface queueSatu = new Seize(devsSimulator,
      resource);
      StationInterface queueDua = new Seize (devsSimulator,
      resourceDua);

      StationInterface releaseSatu = new
      Release(devsSimulator, resource, 1.0);
```

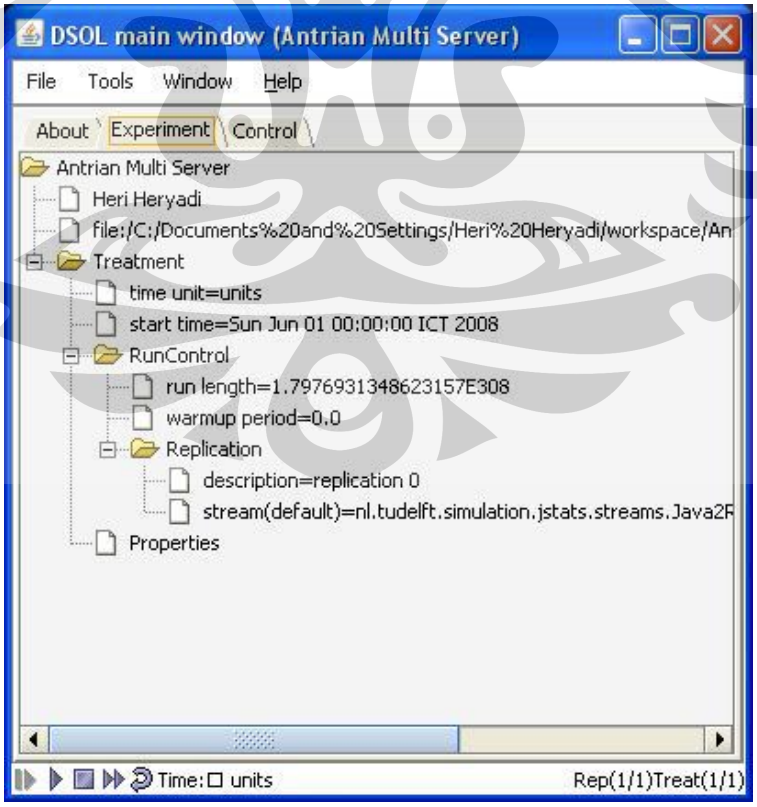
```
        StationInterface releaseDua = new
        Release(devsSimulator, resourceDua, 1.0);

        //The servers
        DistContinuous serviceTimeSatu = new
        DistExponential(defaultStream, 0.5);
        DistContinuous serviceTimeDua = new
        DistExponential(defaultStream, 0.5);
        StationInterface serverSatu = new Delay(devsSimulator,
        serviceTimeSatu);
        StationInterface serverDua = new Delay(devsSimulator,
        serviceTimeDua);

        //The Flow Process
        generatorSatu.setDestination(queueSatu);
        queueSatu.setDestination(serverSatu);
        serverSatu.setDestination(releaseSatu);

        generatorDua.setDestination(queueDua);
        queueDua.setDestination(serverDua);
        serverDua.setDestination(releaseDua);



        //Statistics
        Tally dN = new Tally("d(n)", devsSimulator, queueSatu,
        Seize.DELAY_TIME);
        Tally qN = new Tally("q(n)", devsSimulator, queueDua,
        Seize.QUEUE_LENGTH_EVENT);
        Utilization uN1 = new Utilization("u1(n)",
        devsSimulator, serverSatu);
        Utilization uN2 = new Utilization("u2(n)",
        devsSimulator, serverDua);


        //Charts
        new BoxAndWhiskerChart(devsSimulator, "d(n)
        chart").add(dN);
        new BoxAndWhiskerChart(devsSimulator, "q(n)
        chart").add(qN);
        new BoxAndWhiskerChart(devsSimulator, "u1(n)
        chart").add(uN1);
        new BoxAndWhiskerChart(devsSimulator, "u2(n)
        chart").add(uN2);

    }
}
```

Keterangan:

1. `public class ModelAntrian implements ModelInterface`
   - *Coding* ini berfungsi untuk memperkenalkan ModelAntrian *class* dan menyatakan bahwa *class* ini mengimplementasikan *interface* ModelInterface
   - *Class* ModelAntrian berguna untuk mendefinisikan objek dari suatu model, yaitu sebagai abstraksi atau *blueprint* model yang dibuat



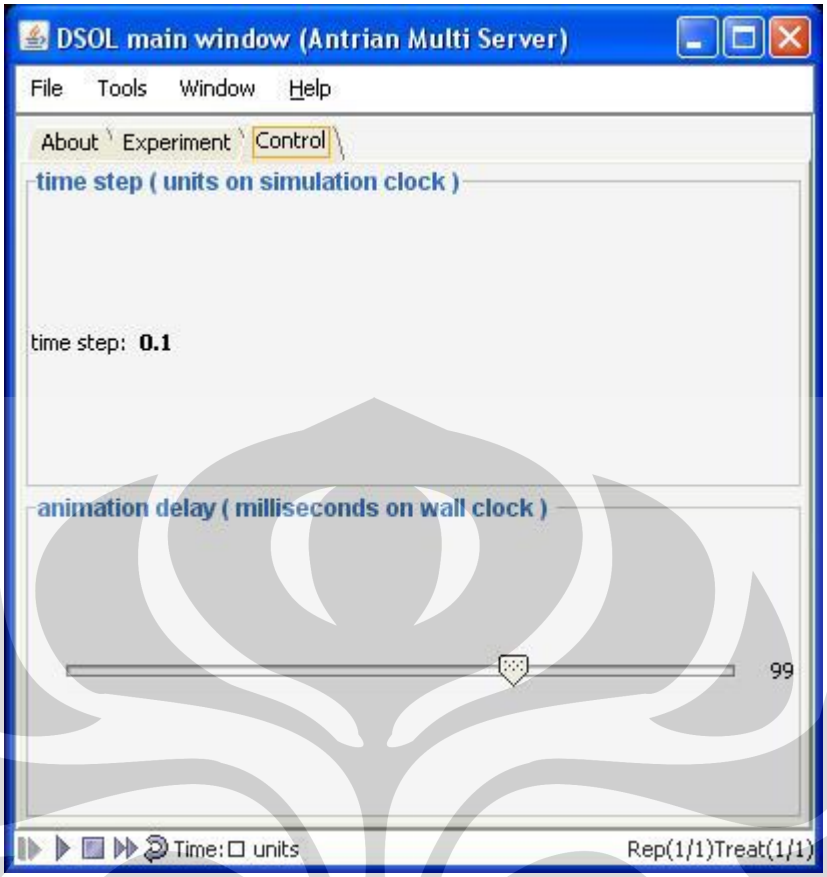
Gambar 4.5 Tampilan awal DSOL



Gambar 4.6 Tampilan Experimentasi Dari XML

2. `public void constructModel(final SimulatorInterface simulator)throws SimRuntimeException, RemoteException {}`

   - Kode ini berfungsi untuk memanggil metode `constructModel()` yang terdapat dalam *class interface ModelInterface*
   - Fungsi `constructModel()` digunakan untuk membangun model dan memasangkan objek *simulator* dari DSOL terhadap model yang dibuat sehingga bisa dijalankan melalui DSOL *Graphical User interface* (GUI)
   - `SimulatorInterface` mendefinisikan *behavior* dari simulator DSOL. Simulator didefinisikan sebagai objek komputasional yang memiliki kemampuan untuk mengeksekusi model. Oleh karena itu simulator merupakan sebuah objek yang dapat melakukan fungsi *stop*, *pause*, *start*, *reset*, dll
   - Metode yang terdapat dalam *class* `SimulatorInterface`:
     - `getReplication()` – untuk mendapatkan replikasi model pada keadaan awal model berjalan
     - `getSimulatorTime()` – untuk mendapatkan waktu aktual pada simulator
     - `initialize(Replication replication)` – untuk menjalankan simulator dengan replikasi yang ditetapkan
     - `isRunning()` – untuk mengetahui status apakah suatu model sedang berjalan atau tidak, bernilai *boolean* yes atau *no*
     - `start()` – berfungsi untuk memulai simulasi
     - `step()` – berfungsi meningkatkan atau memperlambat kecepatan simulasi
     - `stop()` – berfungsi menghentikan simulasi



Gambar 4.7 Tampilan Pengaturan Simulasi

3.
```
StreamInterface
defaultStream = devsSimulator.getReplication()
.getStream("default");
```

   - Kode ini berfungsi untuk menunjukkan bagaimana menggunakan *pseudo random number stream* pada sebuah model. Ini dilakukan dengan cara meminta *stream* data dari suatu replikasi. *Stream* dapat dilihat pada *file* XML *document*

4.
```
Generator generatorSatu = new Generator(devsSimulator,
Object.class, null);

generatorSatu.setInterval(new DistExponential(defaultStream,
1.0));
generatorSatu.setStartTime(new DistConstant(defaultStream,
0.0));
generatorSatu.setBatchSize(new
DistDiscreteConstant(defaultStream, 1));
generatorSatu.setMaxNumber(500);
```

- Kode ini berfungsi untuk membuat objek `Genarator`. Objek `Generator` merupakan sebuah objek yang berfungsi menghasilkan entitas yang akan masuk ke dalam sistem di dalam model. Metode yang terdapat dalam *class* Generator antara lain:
  - `generate()` – berfungsi menghasilkan entitas baru
  - `getBatchSize()` – berfungsi untuk mendapatkan *batch size* dari entitas yang dihasilkan
  - `getInterval()` – berfungsi untuk mendapatkan interval waktu antar kedatangan
  - `getMaxNumber()` – berfungsi mendapatkan jumlah maksimum entitas yang akan dihasilkan
  - `getStartTime()` – berfungsi mendapatkan waktu mulai operasi pada generator
  - `receiveObject()` – berfungsi menerima objek sewaktu entitas datang
  - `setBatchSize()` – berfungsi menentukan *batch size* dari entitas yang dihasilkan pada generator
  - `setInterval()` – berfungsi menentukan interval waktu antar kedatangan
  - `setMaxNumber()` – berfungsi menentukan jumlah maksimal entitas yang dihasilkan
  - `setStartTime()` – berfungsi menentukan waktu awal mulai beroperasi dalam membuat entitas

5. `Resource resource = new Resource(devsSimulator, 1.0);`
   - *Coding* ini berfungsi untuk membuat `resource` dengan kapasitas sebanyak 1 entitas. `Resource` didefinisikan sebagai jumlah jumlah yang dibagi dan dibatasi. Metode yang terdapat dalam *class* `Resource`:
     - `getAvailableCapacity()` – mengambil kapasitas yang tersedia pada `resource`

- `getCapacity()` – mengambil kapasitas asli pada `resource` dan menembalikan pada keadaan maksimum
- `getClaimedCapacity()` – mengambil jumlah kapasitas yang diminta sekarang
- `getQueueLength()` – mengambil jumlah *instance* yang menunggu pada resource
- `releaseCapacity(double amount)` – melepaskan sejumlah kapasitas tertentu dari resource
- `requestCapacity (double amount,ResourceRequestorInterface requestor)` – berfungsi meminta sejumlah kapasitas tertentu dari `resource`
- `setCapacity(double capacity)` – mengatur kapasitas pada `resource`
- `toString()`- berfungsi untuk mengkonversi suatu tipe data ke dalam tipe data *String*

6.
```
// The Queue and release
StationInterface queueSatu = new Seize(devsSimulator,
resource);
StationInterface releaseSatu = new Release(devsSimulator,
resource, 1.0);

//The Server
DistContinuous serviceTimeSatu = new
DistExponential(defaultStream, 0.5);
StationInterface serverSatu = new Delay(devsSimulator,
serviceTimeSatu);
```

- Kode di atas berfungsi untuk membuat objek *`queue, server, dan release`*
- `StationInterface` merupakan *class* yang mengimplementasikan *class* `Station` dan berfungsi untuk membangun suatu *location*
- Metode yang terdapat dalam *class* StationInterface:
  - `getDestination()` – Untuk mendapatkan tujuan selanjutnya dari suatu lokasi
  - `receiveObject(Object object)` – berfungsi menerima objek sewaktu entitas datang

- `setDestination(StationInterface destination)` – mengatur tujuan dari objek

➢ `Seize` merupakan *class* untuk meminta dan mengklaim `resource` dan melepaskan entitas ketika `resource` tersebut diminta oleh objek lain

➢ Metode yang terdapat dalam *class* Seize:

- `getQueue()` – berfungsi untuk mendapatkan antrian
- `receiveObject(Object object)` – menerima objek sewaktu entitas datang
- `receiveRequestedResource(double requestedCapacity, Resource resource)` – berfungsi menerima objek `resource` yang diminta dengan mengirimkan argument berupa kapasitas pelayanan dan jenis `resource`-nya
- `setQueue(List queue)` – mengatur antrian pada `Seize`

➢ `Release` merupakan *class* untuk melepaskan entitas yang diberikan pada `resource` yang diminta

➢ Metode yang terdapat dalam *class* Release:

- `receiveObject(Object object)` – menerima objek sewaktu entitas datang

➢ Objek `delay` merupakan fungsi ketika sejumlah entitas dilayani berdasarkan sejumlah unit waktu, biasanya sesuai *service time*

➢ Metode yang terdapat dalam *class* delay:

- `receiveObject(Object object)` – menerima objek sewaktu entitas datang

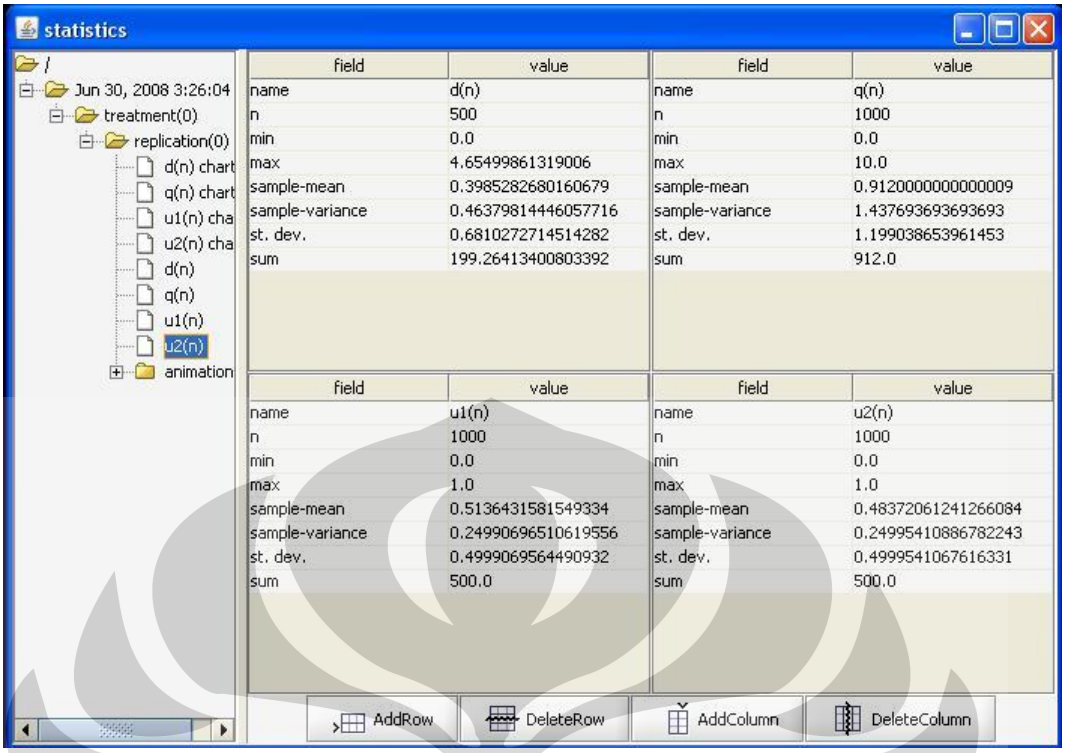
7.
```
generatorSatu.setDestination(queueSatu);
queueSatu.setDestination(serverSatu);
serverSatu.setDestination(releaseSatu);
```

➢ Kode ini berfungsi untuk membuat aliran dalam model dari suatu lokasi ke lokasi yang lainnya yaitu dari luar sistem masuk ke dalam antrian lalu ke server dan selanjutnya dilepas ke luar sistem

8.
```
Tally dN = new Tally("d(n)", devsSimulator, queueSatu,
Seize.DELAY_TIME);
```

```
Tally qN = new Tally("q(n)", devsSimulator, queueDua,
Seize.QUEUE_LENGTH_EVENT);
Utilization uN1 = new Utilization("u1(n)", devsSimulator,
serverSatu);
Utilization uN2 = new Utilization("u2(n)", devsSimulator,
serverDua);
```

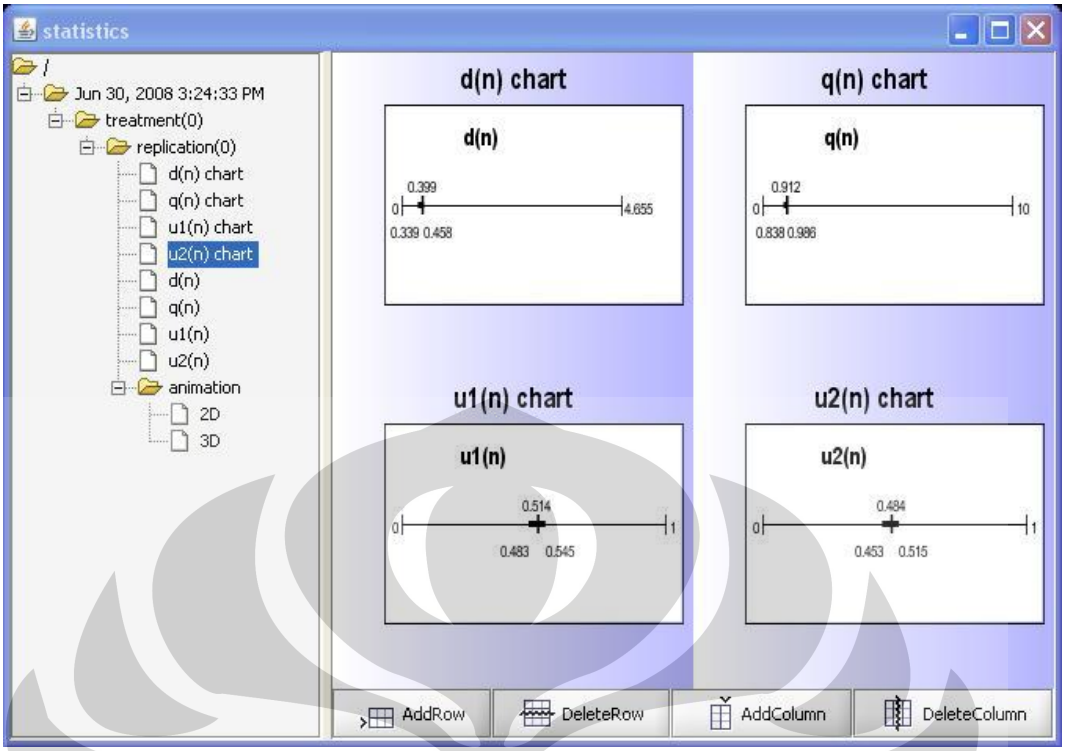
- Kode di atas berfungsi untuk membuat pembilangan entitas yang masuk antrian maupun yang dilayani dan berguna dalam penghitungan data statistik
- `Tally` berfungsi untuk menghitung turus entitas baik yang masuk antrian maupun yang dilayani
- Metode yang terdapat dalam *class* Tally:
  - `endOfReplication()` – digunakan untuk mengakhiri replikasi dari simulasi
  - `notify(EventInterface event)` – digunakan untuk menotifikasi kejadian atau *event*, biasa digunakan dalam *thread programming*
- Objek `utilization` berfungsi untuk menghitung dan merepresentasikan utilisasi dari suatu stasiun atau lokasi, dalam hal ini *server* beserta *resource*-nya
- Metode yang terdapat dalam *class* utilization:
  - `endOfReplication()` – digunakan untuk mengakhiri replikasi dari simulasi
  - `notify(EventInterface event)` – digunakan untuk menotifikasi kejadian atau *event*, biasa digunakan dalam *thread programming*



Gambar 4.8 Tampilan Data Statistik

9.
```
new BoxAndWhiskerChart(devsSimulator, "d(n) chart").add(dN);
new BoxAndWhiskerChart(devsSimulator, "q(n) chart").add(qN);
new BoxAndWhiskerChart(devsSimulator, "u1(n)
chart").add(uN1);
new
BoxAndWhiskerChart(devsSimulator, "u2(n) chart").add(uN2);
```

- Kode ini berfungsi untuk membuat grafik atau chart. Input dari chart tersebut diperoleh dari objek *Tally*



Gambar 4.9 Tampilan Data Grafik

10. `generatorSatu.setInterval(new DistExponential(defaultStream, 1.0));`
    `DistContinuous serviceTimeSatu = new DistExponential(defaultStream, 0.5);`

    - Kode di atas merupakan fungsi yang dapat dimodifikasi oleh pembuat model untuk mendapatkan sistem antrian dengan hasil yang berbeda sesuai input data distribusi yang berbeda-beda. `Interval` merupakan rata-rata kecepatan kedatangan (λ) `serviceTime` adalah waktu pelayanan (μ)
    - Nilai 1.0 pada interval dan 0.5 pada `serviceTime` dapat diubah sesuai dengan keinginan pembuat model
    - Distribusi statistik yang digunakan merupakan bagian *package* pada *nl.tudelft.simulation.jstats.distributions*

4.2.2. Kelemahan Sistem

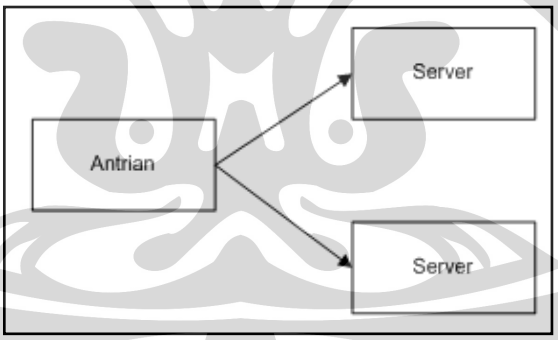
Model simulasi antrian yang dibuat dengan menggunakan DSOL pada penelitian ini masih terdapat beberapa kekurangan, yaitu:

4.2.2.2. Belum adanya visualisasi simulasi

Hasil simulasi belum dapat disajikan tampilan pergerakan sistem atau animasi secara visualisasi dua dimensi karena keterbatasan teknis dalam pemrograman Java yang akan dipakai terutama dalam menggunakan *library* animasi dalam DSOL. DSOL sendiri memiliki dokumentasi yang sangat kurang dalam memberikan keterangan tentang *library* yang dimuat sehingga terdapat kesulitan terutama dalam mengembangkan model yang lebih detail.

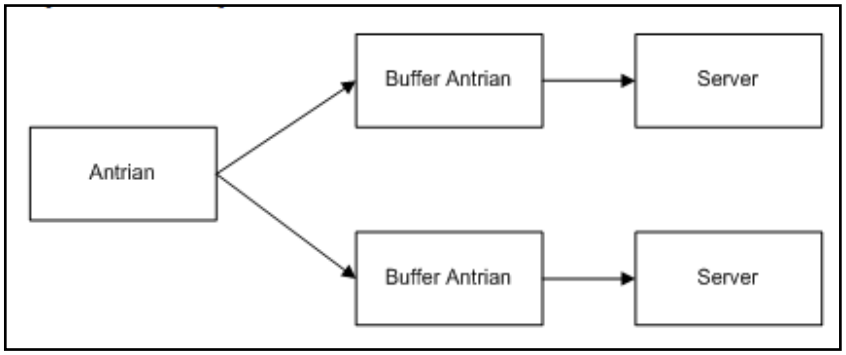
4.2.2.3. Fungsi pada antrian *multi server* masih tidak sesuai yang diinginkan

Secara umum pada sistem antrian *multi server* pelayanan dapat dilakukan secara paralel. Apabila suatu tugas atau pelayanan sudah diselesaikan maka secara otomatis *server* akan "menarik" entitas yang sedang menunggu, jika ada, dari antrian. Tapi jika antrian kosong, *server* menjadi *idle* sampai entitas baru datang. Sehingga apabila terdapat dua *server* atau lebih, maka seharusnya ketika salah satu *server* dalam keadaan *idle*, *server* tersebut akan langsung mengambil entitas yang sedang menunggu secara paralel.



Gambar 4.10 Sistem Antrian Paralel *Multi Server*

Tetapi pada model yang dibuat yang terjadi adalah ketika salah satu *server* dalam keadaan *idle* sedangkan *server* lainnya sedang *busy*, *server* yang *idle* tersebut akan menunggu sampai *server* yang *busy* selesai melayani. Juga pada model yang dibuat, ketika entitas memasuki antrian maka entitas akan memasuki antrian *buffer* untuk menentukan ke server mana entitas akan meminta pelayanan.



Gambar 4.11 Sistem *Multi Server* Dengan *Buffer*

Secara logika kode fungsi Java yang digunakan sudah benar. Namun terdapat kesulitan dalam mengakses metode dari *library* DSOL dalam menghasilkan entitas yang memiliki kemampuan untuk mengambil keputusan terhadap server mana yang dipilih.

5. KESIMPULAN

1. Pembuatan simulasi yang berbasis *object-oriented simulation* dengan menggunakan DSOL telah berhasil dibuat dengan memodelkan system antrian multi server
2. Perbandingan antara *object-oriented simulation* dengan *flow-based simulation* adalah*:*
   a. Metodologi

      Pada tahap konseptualisasi model, metode *object-oriented simulation* menggunakan *Unified Modeling Language*, sedangkan *flow-based simulation* memetakan proses dengan *flow chart* atau IDEF0 *diagram*.
   b. Orientasi rancangan
      i. Modularitas

         Modularitas adalah derajat dari standardisasi dan kebebasan modul, serta variasi dalam penggunaan suatu rancangan model.
         - Orientasi proses

           Berfokus pada fungsi atau metode yang terlibat dalam sistem
         - Orientasi objek

           Berfokus pada objek atau actor yang terlibat

      ii. Desain
         - Orientasi proses

           Desain dimulai dengan sebuah deskripsi abstrak kemudian menyaring pada langkah berikutnya.
         - Orientasi objek

           Desain yang dipakai adalah bottom-up. Semua level terkecil pada sistem, yang disebut objek dibuat sehingga sistem yang ada dapat dijalankan dengan menginteraksikan objek-objek yang ada tersebut.

3. Langkah-langkah penggunaan metodologi *object-oriented simulation* dalam membuat model adalah sebagai berikut:

a. Membuat daftar proses bisnis dari level tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin timbul. Dalam penelitian ini proses bisnis yang terjadi dalam sistem antrian didefinisikan ke dalam bentuk *flow chart*.
b. Memetakan *use case* untuk tiap proses bisnis untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh sistem. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.
c. Berdasarkan model-model yang sudah ada, setelah itu dibuat *class diagram*. *Class diagram* menggambarkan struktur dan deskripsi *class*, *package*, dan objek beserta hubungan satu sama lain. *Class* mendefinisikan sebuah tipe dari objek. Setiap *package* atau *domain* dipecah menjadi *hierarki class* lengkap dengan atribut dan metodenya.
d. Memulai membangun sistem dengan *software* yang ada, Pada penelitian ini digunakan DSOL sebagai media untuk men-*develop* model yang sudah dibuat.
e. Melakukan uji modul dan uji integrasi serta memperbaiki model beserta kode programnya. Model harus selalu sesuai dengan kode program aktualnya. Dalam hal ini verifikasi model dilakukan terhadap nilai perhitungan secara manual.

DAFTAR PUSTAKA

Allimant, F. (2004). *The Java Tutorial: A Practical Guide for Programmers*.

Armstrong, Eric et al, (2005). *The J2EE 1.4 Tutorial*. For Sun Java System Application Server Platform Edition 8.2, Sun Microsystems California, hal.1-252.

Banks, Jerry. (1998). *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practices*. John Willey and Son, Atlanta

Baldwin, Rusty O. (1998). *Real-Time Queueing Theory: A Tutorial Presentation with An Admission Control Application*. Air Force Institute of Technology, hal.1-21.

Deitel, H.M & Deitel, P.J. (2004). *Java How to Program*. New Jersey: Prentice Hall.

Fowler, Martin. (2004). *UML Distilled: A Brief Guide ToThe Standard Object Modeling Language*, Third Edition. Boston: Addison-Wesley.

Harrel, Charles, Ghosh, Biman K., & Bowden, Royce. (2000). *Simulation Using ProModel*. Mc Graw Hill.

Jacobs, Peter H. M. dan Verbraeck, Alexander. (2004). *Mastering D-SOL: A Java Based Suite for Simulation*. Delft University of Technology Netherlands, hal.1-61.

Jacobs, Peter H. M. dan Verbraeck, Alexander. (2004). *Single-Threaded Spesification of Process-Interaction Formalism in Java*. Delft University of Technology Netherlands, hal.1-8.

Jacobs, Peter H. M., Lang Niels A., dan Verbraeck, Alexander. (2002). *D-SOL: Distributed Java Based Discrete Event Simulation Architecture*. Delft University of Technology Netherlands, hal.1-8.

Jacobs, Peter H. M., Verbraeck, Alexander, dan Rengelink, William, (2004). *Emulation With DSOL*. Delft University of Technology Netherlands, hal.1-10.

Jacobs, Peter H. M., (2005). *The DSOL Simulation Suite*. Delft University of Technology Netherlands, hal.1-228.

Joines, Jeffrey A. dan Roberts, Stephen D. (1999). *Simulation in An Object-Oriented World*. North Carolina State University, hal.132-140.

Lang Niels A., Jacobs, Peter H. M, & Verbraeck, Alexander. (2005). *RMIS: Middleware for Transparent Object-Oriented Modeling in Multi-Simulator System*, Delft University of Technology Netherlands, hal.1-10.

Lang, Niels A. (2002). *Distributed, Open Simulation Model Development with DSOL Service*. Erasmus University Rotterdam, hal.1-9.

Roberts, Chell A. D & Dessouky, Yasser M. (1998). *An Overview of Object-Oriented Simulation*. Simulation Councils United States of America, hal.359-368.

Schild, Herbert. (2005). *Java A Beginner's Guide*. McGraw-Hill.

Sierra, Kathy & Bates, Bert. (2003). *Head First Java*. O'Reilly, United States of America.

Utama, Ginanjar. (2002). *Berfikir Objek: Cara Efektif Menguasai Java*. IlmuKomputer.Com, hal.1-182.

<http://www.w3c.org>

LAMPIRAN

Lampiran 4.1. *Source Code* ApplicationLauncher.java

```
import java.net.URL;
import java.rmi.RemoteException;

import nl.tudelft.simulation.dsol.experiment.Experiment;
import nl.tudelft.simulation.dsol.gui.DSOLApplication;
import
nl.tudelft.simulation.dsol.gui.panels.GUIExperimentParsingThread;
import nl.tudelft.simulation.event.EventInterface;
import nl.tudelft.simulation.event.EventListenerInterface;
import nl.tudelft.simulation.language.io.URLResource;
import nl.tudelft.simulation.xml.dsol.ExperimentParsingThread;


public class ApplicationLauncher extends DSOLApplication
implements EventListenerInterface
{
    /**
     * Construct ModelAntrian yang baru
     *
     * @param url url with navigation-bar properties
     */
    public ApplicationLauncher(final URL url)
    {
        super(url);

        /*
         * DSOL-1.6.2 gunakan constructor yang berbeda!
         * super();
         */
        this.addListener(this,
        ExperimentParsingThread.EXPERIMENT_PARSED_EVENT);

        new GUIExperimentParsingThread(this, null, URLResource
                .getResource("/ModelAntrian.xml")).start();

    }

    /**
     * @lihat nl.tudelft.simulation.event.EventListenerInterface#
     */

    public void notify(final EventInterface event) throws
    RemoteException{
        this.setExperiment((Experiment) event.getContent());
    }
```

```
    /**
     * executes dsol control panel
     *
     * @param args the command-line arguments (none required)
     */


    public static void main(final String[] args)
    {
        // Note: in DSOL-1.6.6 we can set the navigation bar
        URL navigation = null;
        if (args.length != 0)
        {
            navigation = URLResource.getResource(args[0]);
        } else if (System.getProperty("dsol.navigation") != null)
        {
            navigation = URLResource.getResource(System
                    .getProperty("dsol.navigation"));
        }
        if (navigation == null)
        {
            navigation =
            URLResource.getResource("/navigation.xml");
        }
        new ApplicationLauncher(navigation);
    }
}
```

Lampiran 4.2. *Source Code* ModelAntrian.java

```
import java.rmi.RemoteException;
import nl.tudelft.simulation.dsol.ModelInterface;
import nl.tudelft.simulation.dsol.SimRuntimeException;
import nl.tudelft.simulation.dsol.formalisms.Resource;
import nl.tudelft.simulation.dsol.formalisms.flow.Delay;
import nl.tudelft.simulation.dsol.formalisms.flow.Generator;
import nl.tudelft.simulation.dsol.formalisms.flow.Release;
import nl.tudelft.simulation.dsol.formalisms.flow.Seize;

import
nl.tudelft.simulation.dsol.formalisms.flow.StationInterface;

import
nl.tudelft.simulation.dsol.formalisms.flow.statistics.Utilization;
import
nl.tudelft.simulation.dsol.simulators.DEVSSimulatorInterface;

import nl.tudelft.simulation.dsol.simulators.SimulatorInterface;
import nl.tudelft.simulation.dsol.statistics.Tally;

import
nl.tudelft.simulation.dsol.statistics.charts.BoxAndWhiskerChart;
import nl.tudelft.simulation.jstats.distributions.DistConstant;
import nl.tudelft.simulation.jstats.distributions.DistContinuous;

import
nl.tudelft.simulation.jstats.distributions.DistDiscreteConstant;
import nl.tudelft.simulation.jstats.distributions.DistExponential;
import nl.tudelft.simulation.jstats.streams.StreamInterface;

public class ModelAntrian implements ModelInterface
{
	/**
	 * constructor for the Antrian Multi Server
	 */
	public ModelAntrian()
	{
		super();
	}

	public int statusServerSatu;
	public int statusServerDua;

	int statusEntitas[];

	public void constructModel(final SimulatorInterface
	simulator)
			throws SimRuntimeException, RemoteException
	{
		DEVSSimulatorInterface devsSimulator =
		(DEVSSimulatorInterface) simulator;
```

```
StreamInterface defaultStream =
devsSimulator.getReplication()
            .getStream("default");


//The Generator
Generator generatorSatu = new Generator(devsSimulator,
Object.class, null);
Generator generatorDua = new Generator(devsSimulator,
Object.class, null);

generatorSatu.setInterval(new
DistExponential(defaultStream, 1.0));
generatorSatu.setStartTime(new
DistConstant(defaultStream, 0.0));
generatorSatu.setBatchSize(new
DistDiscreteConstant(defaultStream, 1));
generatorSatu.setMaxNumber(500);

generatorDua.setInterval(new
DistExponential(defaultStream, 1.0));
generatorDua.setStartTime(new
DistConstant(defaultStream, 0.0));
generatorDua.setBatchSize(new
DistDiscreteConstant(defaultStream, 1));
generatorDua.setMaxNumber(500);



//The queue, the resource and the release
Resource resource = new Resource(devsSimulator, 1.0);
Resource resourceDua = new Resource(devsSimulator,
1.0);

// created a resource
StationInterface queueSatu = new Seize(devsSimulator,
resource);
StationInterface queueDua = new Seize (devsSimulator,
resourceDua);

StationInterface releaseSatu = new
Release(devsSimulator, resource, 1.0);
StationInterface releaseDua = new
Release(devsSimulator, resourceDua, 1.0);

//The servers
DistContinuous serviceTimeSatu = new
DistExponential(defaultStream, 0.5);
DistContinuous serviceTimeDua = new
DistExponential(defaultStream, 0.5);
StationInterface serverSatu = new Delay(devsSimulator,
serviceTimeSatu);
StationInterface serverDua = new Delay(devsSimulator,
serviceTimeDua);

//The Flow
generatorSatu.setDestination(queueSatu);
queueSatu.setDestination(serverSatu);
```

```
        serverSatu.setDestination(releaseSatu);

        generatorDua.setDestination(queueDua);
        queueDua.setDestination(serverDua);
        serverDua.setDestination(releaseDua);



        //Statistics
        Tally dN = new Tally("d(n)", devsSimulator, queueSatu,
        Seize.DELAY_TIME);
        Tally qN = new Tally("q(n)", devsSimulator, queueDua,
        Seize.QUEUE_LENGTH_EVENT);
        Utilization uN1 = new Utilization("u1(n)",
        devsSimulator, serverSatu);
        Utilization uN2 = new Utilization("u2(n)",
        devsSimulator, serverDua);


        //Charts
        new BoxAndWhiskerChart(devsSimulator, "d(n)
        chart").add(dN);
        new BoxAndWhiskerChart(devsSimulator, "q(n)
        chart").add(qN);
        new BoxAndWhiskerChart(devsSimulator, "u1(n)
        chart").add(uN1);
        new BoxAndWhiskerChart(devsSimulator, "u2(n)
        chart").add(uN2);

    }
}
```

Lampiran 4.3. *Source Code* ModelAntrian.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dsol:experiment
      xmlns:dsol="http://www.simulation.tudelft.nl"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <name>Antrian Multi Server</name>
  <analyst>Heri Heryadi</analyst>
    <model>
        <model-class>ModelAntrian</model-class>
    </model>
  <treatment>
    <startTime>2008-06-01T00:00:00</startTime>
    <timeUnit>UNIT</timeUnit>
    <runControl>
         <warmupPeriod  unit="UNIT">0</warmupPeriod>
         <runLength unit="UNIT">INF</runLength>
         <replication description="replication 0">
           <stream name="default" seed="555"/>
         </replication>
   </runControl>
 </treatment>
</dsol:experiment>
```