PENGEMBANGAN *AGENT SERVER* SEBAGAI ANTARMUKA ANTARA *AGENT* DENGAN *SERVER DATABASE* PADA SISTEM MANAJEMEN KELAS BERBASIS *MULTI-AGENT* DENGAN MENGGUNAKAN JADE

SKRIPSI

OLEH:

ARMAN
04 04 03 0164

DEPARTEMEN TEKNIK ELEKTRO

FAKULTAS TEKNIK UNIVERSITAS INDONESIA

JUNI 2008

PENGEMBANGAN *AGENT SERVER* SEBAGAI ANTARMUKA ANTARA *AGENT* DENGAN *SERVER DATABASE* PADA SISTEM MANAJEMEN KELAS BERBASIS *MULTI-AGENT* DENGAN MENGGUNAKAN JADE

SKRIPSI

OLEH:

ARMAN
04 04 03 0164

SKRIPSI INI DIAJUKAN UNTUK MELENGKAPI SEBAGIAN PERSYARATAN UNTUK MENJADI SARJANA TEKNIK

DEPARTEMEN TEKNIK ELEKTRO
FAKULTAS TEKNIK UNIVERSITAS INDONESIA
JUNI 2008

PERNYATAAN KEASLIAN SKRIPSI

Saya menyatakan dengan sesungguhnya bahwa skripsi dengan judul:

PENGEMBANGAN *AGENT SERVER* SEBAGAI ANTARMUKA ANTARA *AGENT* DENGAN *SERVER DATABASE* PADA SISTEM MANAJEMEN KELAS BERBASIS *MULTI-AGENT* DENGAN MENGGUNAKAN JADE

yang dibuat untuk melengkapi sebagian persyaratan menjadi sarjana teknik pada Program Studi Teknik Komputer Departemen Teknik Elektro Fakultas Teknik Universitas Indonesia, sejauh yang saya ketahui bukan merupakan tiruan atau duplikasi dari skripsi yang sudah dipublikasikan dan atau pernah dipakai untuk mendapatkan gelar kesarjanaan di lingkungan Universitas Indonesia maupun di Perguruan Tinggi atau Instansi manapun, kecuali bagian yang sumber informasinya dicantumkan sebagaimana mestinya.

Depok, 25 Juni 2008

Arman
NPM 04 04 03 0164

PENGESAHAN

Skripsi dengan judul:

PENGEMBANGAN *AGENT SERVER* SEBAGAI ANTARMUKA ANTARA *AGENT* DENGAN *SERVER DATABASE* PADA SISTEM MANAJEMEN KELAS BERBASIS *MULTI-AGENT* DENGAN MENGGUNAKAN JADE

dibuat untuk melengkapi sebagian persyaratan menjadi Sarjana Teknik pada Program Studi Teknik Komputer Departemen Teknik Elektro Fakultas Teknik Universitas Indonesia. Skripsi ini telah diujikan pada sidang ujian skripsi pada tanggal 9 Juli 2008 dan dinyatakan memenuhi syarat/sah sebagai skripsi pada Departemen Teknik Elektro Fakultas Teknik Universitas Indonesia.

Depok, 25 Juni 2008

Dosen Pembimbing,

F Astha Ekadiyanto ST, M.Sc.

NIP 132 166 489

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada :

F. Astha Ekadiyanto ST, M.Sc.

selaku dosen pembimbing yang telah bersedia meluangkan waktu untuk memberi pengarahan, diskusi dan bimbingan serta persetujuan sehingga skripsi ini dapat selesai dengan baik.

Disamping itu, penulis tidak lupa pula mengucapkan terima kasih kepada:

Prof. Dr. -Ing. Dr. h.c.(UKM) Axel Hunger

yang telah memberikan ijin kepada penulis untuk mengerjakan skripsi ini di Multimedia Lab Mercator Office, Universitas Indonesia.

Arman
NPM 04 04 03 0164
Departemen Teknik Elektro

Dosen Pembimbing
F. Astha Ekadiyanto ST, M.Sc.

PENGEMBANGAN *AGENT SERVER* SEBAGAI ANTARMUKA ANTARA *AGENT* DENGAN *SERVER DATABASE* PADA SISTEM MANAJEMEN KELAS BERBASIS *MULTI-AGENT* DENGAN MENGGUNAKAN JADE

ABSTRAK

Pemrograman berorientasi *agent* (AOP) dalam pengembangan perangkat lunak termasuk sebuah paradigma yang masih relatif baru. AOP memodelkan aplikasi sebagai sekumpulan komponen yang disebut *agent*. Sebuah *agent* perangkat lunak memiliki sifat yang serupa dengan agen manusia. *Agent* juga memiliki kemampuan untuk mengerjakan tugas rumit yang diberikan kepadanya secara otomatis, dan mampu berkomunikasi dan bekerja sama dengan *agent* lain untuk menyelesaikan tugasnya. Salah satu *middleware* yang digunakan untuk pengembangan aplikasi *multi-agent* adalah JADE yang berjalan pada bahasa pemrograman *Java*.

Untuk melakukan eksplorasi terhadap pemrograman berorientasi *agent*, maka dalam penulisan skripsi ini akan dibahas tahapan-tahapan yang dilakukan pada pengembangan Sistem Manajemen Kelas berbasis *multi-agent*. Bagian Sistem Manajemen Kelas yang dibahas pada skripsi ini adalah penggunaan Agent Server sebagai antarmuka yang menghubungkan *Agent* Kelas dengan *server database*.

Optimasi terhadap Agent Server juga telah dilakukan terutama dengan penggunaan *class* BehaviourPool dan *connection pooling* yang berdasarkan hasil pengujian mampu mengurangi waktu pemrosesan sebesar 16% pada beban di atas 10 permintaan/detik.

Kata Kunci: *Agent*, JADE, Java, Sistem Manajemen Kelas

Arman
NPM 04 04 03 0164
Electrical Engineering Department

Counsellor
F. Astha Ekadiyanto ST, M.Sc.

DEVELOPMENT OF SERVER AGENT AS AN INTERFACE BETWEEN AGENT AND DATABASE SERVER IN MULTI-AGENT BASED CLASS MANAGEMENT SYSTEM WITH JADE

ABSTRACT

Agent Oriented Programming (AOP) in software development is a relatively new paradigm. AOP models an application as a collection of components called agents. A software agent has similar behaviour as human agent. An agent also has the ability to carry a complex task assigned to them automatically, and communicate and cooperate with other agents to complete their tasks. One of several middlewares that is used to develop a multi-agent application is JADE which is run on Java programming language.

To explore agent oriented programming, this thesis will analyse several phases in developing agent-based Class Management System. Part of Class Management System that will be analysed in this thesis is the use of Server Agent as an interface that connects Class Agent with database server.

Optimization on Agent Server was done by using `BehaviourPool` class and connection pooling which based on testing result can reduce processing time by 16% at request rate above 10 requests/second.

Keyword: *Agent*, JADE, Java, Class Management System

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR SINGKATAN

AMS	*Agent Management System*
ACL	*Agent Communication Language*
API	*Application Programming Interface*
CFP	*Call for Proposal*
DBMS	*Database Management System*
DF	*Directory Facilitator*
FIPA	*Foundation for Intelligent Physical Agent*
GPL	*GNU General Public License*
GUI	*Graphical user Interface*
JADE	*Java Agent Development Framework*
JDBC	*Java DataBase Connectivity*
MAS	*Multi Agent System*
RDBMS	*Relational Database Management System*
RMA	*Remote Monitoring Agent*
SQL	*Structured Query Language*

BAB I
PENDAHULUAN

1.1 LATAR BELAKANG

Perkembangan di bidang ilmu pengetahuan dan teknologi telah menghasilkan berbagai kemudahan bagi kehidupan manusia. Namun seiring dengan perkembangan tersebut masalah yang harus dihadapi manusia juga menjadi semakin rumit. Salah satu pendekatan yang digunakan untuk memecahkan permasalahan yang rumit adalah pendekatan berbasis *agent*.

Pendekatan berbasis *agent* dapat mempermudah perancangan sebuah sistem perangkat lunak yang rumit dengan memecah permasalahan menjadi beberapa bagian dan menugaskan penyelesaian bagian-bagian permasalahan tersebut kepada beberapa *agent* sesuai dengan keahlian masing-masing *agent*. Sebuah *agent* perangkat lunak memiliki sifat yang menyerupai agen manusia pada umumnya. *Agent* perangkat lunak memiliki kecerdasannya sendiri, mampu bertukar informasi dan bekerja sama dengan *agent* lain, dan selalu berusaha menyelesaikan tugas yang diberikan kepadanya.

Salah satu *middleware* yang digunakan untuk mengembangkan perangkat lunak berbasis *agent* adalah JADE (*Java Agent DEvelopment Framework*). JADE sepenuhnya dibuat dengan menggunakan bahasa pemrograman *Java*, sehingga memiliki semua fitur *library* yang ditawarkan pada bahasa pemrograman *Java*, dan dapat berjalan pada berbagai *platform* tanpa harus mengubah kode sumbernya.

Manajemen kegiatan perkuliahan di Fakultas Teknik, Universitas Indonesia sampai saat ini masih dilakukan secara manual, mulai dari pencarian jadwal sampai pengisian absen. Manajemen kegiatan perkuliahan secara manual memiliki banyak masalah, mulai dari ketidaksamaan jadwal yang terdapat pada departemen dan ruang kelas, sampai daftar pengisian absen yang sering hilang.

Skripsi ini mencoba melakukan eksplorasi terhadap pendekatan berbasis *agent* dengan mengembangkan sebuah aplikasi berbasis agent untuk mengatur kegiatan perkuliahan secara otomatis dengan menggunakan JADE yang diberi nama Sistem Manajemen Kelas. Selain itu pengujian juga dilakukan terhadap

aplikasi Sistem Manajemen Kelas untuk mengetahui sejauh mana kemampuan dari aplikasi berbasis *agent* yang dibuat dengan menggunakan JADE.

1.2 TUJUAN

Penulisan skripsi ini bertujuan untuk

1. Membahas proses pengembangan aplikasi Sistem Manajemen Kelas yang dilakukan penulis mulai dari tahap analisa sampai tahap pengujian.
2. Mengetahui kinerja dari aplikasi *multi-agent* yang dibuat dengan menggunakan JADE.

1.3 PEMBATASAN MASALAH

Pada skripsi ini akan dibahas proses pengembangan aplikasi Sistem Manajemen Kelas. Proses pengembangan yang dibahas meliputi tahapan-tahapan perencanaan, analisa, perancangan, implementasi, dan pengujian. Tahap perencanaan dan analisa akan membahas Sistem Manajemen Kelas secara keseluruhan, sedangkan tahap perancangan, implementasi dan pengujian hanya akan membahas mengenai pengembangan *database*, *Agent Server*, dan GUI (*Graphical User Interface*). Skripsi ini tidak membahas tahap perancangan, implementasi, dan pengujian dari *agent* pada ruang kelas dan departemen, karena bagian tersebut dikerjakan oleh pemrogram lain yang ikut mengembangkan aplikasi Sistem Manajemen Kelas.

1.4 SISTEMATIKA PENULISAN

Skripsi ini ditulis dengan sistematika sebagai berikut:

- **BAB I PENDAHULUAN**

  Bab ini berisi Latar Belakang, Tujuan Penulisan, Batasan Masalah, dan Sistematika Penulisan dari skripsi ini.

- **BAB II *JAVA*, *AGENT*, JADE, DAN MYSQL**

  Bab ini menjelaskan konsep dasar dari *Java*, *Agent*, JADE, dan MySQL yang digunakan sebagai dasar pada penulisan skripsi ini.

- **BAB III PERENCANAAN, ANALISA, DAN PERANCANGAN APLIKASI SISTEM MANAJEMEN KELAS**

  Bab ini membahas tentang tahapan-tahapan yang dilakukan pada proses perencanaan, analisa, dan perancangan aplikasi Sistem manajemen Kelas beserta diagram-diagram yang dihasilkan untuk mempermudah proses implementasi.
- **BAB IV IMPLEMENTASI *DATABASE*, *AGENT SERVER* DAN GUI PADA SISTEM MANAJEMEN KELAS**

  Bab ini menjelaskan hasil implementasi database, Agent Server, dan GUI dari Sistem Manajemen Kelas beserta penjelasan mengenai cara kerja dan teknik yang digunakan.
- **BAB V PENGUJIAN GUI, DAN *AGENT SERVER* PADA SISTEM MANAJEMEN KELAS**

  Bab ini menjelaskan proses yang dilakukan pada tahap pengujian beserta hasil dan analisa pengujian.
- **BAB VI KESIMPULAN**

  Bab ini berisi kesimpulan dari penulisan skripsi ini.

BAB II
JAVA, AGENT, JADE, DAN MYSQL

2.1 JAVA

Java merupakan sebuah bahasa pemrograman tingkat tinggi yang berorientasi objek yang dikembangkan oleh *Sun Microsystems*. Kemampuannya untuk dijalankan pada berbagai jenis arsitektur komputer dan ketersediannya sebagai perangkat lunak gratis membuat bahasa pemrograman *Java* lebih banyak dikenal dan digunakan daripada bahasa pemrograman berorientasi objek lainnya.

2.1.1 Sejarah Bahasa Pemrograman *Java*

Pada tahun 1991, James Gosling dan Patrick Naughton membentuk *Green Project* yang bertujuan untuk menemukan cara agar peralatan elektronik dapat saling berkomunikasi satu dengan lainnya. Mereka menyadari bahwa peralatan elektronik konsumen memiliki CPU yang berbeda-beda dan memori yang terbatas. Untuk mengatasi masalah tersebut, James Gosling mulai mengembangkan bahasa pemrograman baru yang diberi nama *Oak* pada bulan April 1991.

Bahasa pemrograman *Oak* dibuat berdasarkan bahasa pemrograman C++. Agar *Oak* dapat digunakan pada peralatan elektronik konsumen, maka James Gosling memangkas fitur-fitur bahasa pemrograman C++ menjadi seminimal mungkin. Selain itu Oak juga memiliki kemampuan untuk berjalan di berbagai jenis CPU.

Pada bulan Agustus 1991, *Oak* berhasil menjalankan program pertamanya, dan pada Oktober 1992, *Green Project* berhasil menciptakan *7 (Gambar 2.1) yang berfungsi untuk mengatur berbagai peralatan rumah tangga secara nirkabel.

Karena industri *video-on-demand* yang menjadi sasaran pemasaran *7 masih merupakan industri yang baru berkembang, maka pasar tanggapan terhadap *7 tidak terlalu baik. Kegagalan tersebut menyebabkan *Green Project* beralih ke pengembangan aplikasi *on-line* dan CD-ROM berbasis *Oak*. Seiring dengan peralihan ini, nama bahasa pemrograman *Oak* dan *GreenProject* mengalami perubahan menjadi *Java*, dan *FirstPerson*.

Gambar 2.1. Perangkat *7 dengan *user interface* eksperimental

Akhirnya pada Maret 1995, *FirstPerson* meluncurkan *Java 1.0a* kepada publik dengan slogan WORA (*Write Once, Run Anywhere*). Setelah peluncuran pertama ini, bahasa pemrograman *Java* semakin dikenal dan digunakan secara luas. Banyak *web browser* segera mengimplementasikan kemampuan untuk menjalakan *Java applet* yang aman. Dengan datangnya *Java 2*, ditambahkan beberapa konfigurasi untuk menjalankan *Java* di *platform* yang berbeda. Sebagai contoh J2EE diperuntukkan bagi aplikasi di tingkat perusahaan, dan J2ME dikhususkan bagi aplikasi pada perangkat bergerak.

2.1.2 Fitur-fitur Bahasa Pemrograman *Java*

Persyaratan dari bahasa pemrograman *Java* berasal dari lingkungan komputasi dimana perangkat lunak harus dijalankan. Perkembangan *internet* dan *World-Wide Web* yang pesat telah membawa kita kepada cara baru dalam memandang pengembangan dan pendistribusian perangkat lunak. Oleh sebab itu, teknologi *Java* harus memungkinkan pengembangan aplikasi yang aman, berkinerja tinggi, dan tangguh yang dapat bekerja pada beberapa platform dalam jaringan yang heterogen dan terditribusi [2].

Persyaratan-persyaratan di atas terdiri dari beberapa kata kunci. Kata-kata kunci tersebut akan dijelaskan dalam sub-sub bab berikut ini [2] [3].

2.1.2.1 Sederhana, Berorientasi Objek, dan Dikenal

Sederhana berarti bahasa pemrograman *Java* dapat dilakukan tanpa membutuhkan pelatihan pemrogram yang ekstensif. Konsep dasar bahasa pemrograman *Java* adalah dapat dimengerti dengan cepat, sehingga seorang pemrogram dapat menjadi produktif sejak awal.

Sistem *Java* menyediakan *platform* pengembangan berorientasi objek yang bersih dan efisien agar perangkat lunak dapat berfungsi pada lingkungan jaringan yang semakin rumit.

Pengembangan bahasa pemrograman *Java* yang menyerupai C++ membuat bahasa pemrograman *Java* menjadi bahasa yang dikenal. Dengan digunakannya fitur-fitur dari C++, pemrogram dapat berpindah ke *Java platform* dengan cepat dan segera menjadi produktif.

2.1.2.2 Tangguh dan Aman

Bahasa pemrograman *Java* dirancang untuk membuat perangkat lunak yang dapat diandalkan. Bahasa pemrograman ini menyediakan pemeriksaan pada proses *compile*, dan pemeriksaan tingkat kedua di saat menjalankan program.

Teknologi *Java* dirancang untuk beroperasi di lingkungan yang terdistribusi dimana keamanan menjadi faktor yang sangat penting. Dengan adanya fitur keamanan, teknologi *Java* memungkinkan pemrogram membangun aplikasi yang tidak dapat diserang dari luar.

2.1.2.3 Tidak Tergantung Arsitektur dan Mudah Dibawa

Teknologi *Java* dirancang untuk mendukung aplikasi yang akan dijalankan dalam lingkungan jaringan yang heterogen. Pada lingkungan semacam itu, aplikasi harus memiliki kemampuan untuk dieksekusi pada berbagai jenis arsitektur perangkat keras. Di dalam berbagai jenis platform perangkat keras ini, aplikasi harus dapat dieksekusi di atas berbagai sistem operasi dan berkomunikasi dengan berbagai antarmuka bahasa pemrograman.

Ketidak-tergantungan terhadap arsitektur merupakan salah satu bagian dari sistem yang sepenuhnya mudah dibawa. Program yang dibuat dengan bahasa pemrograman *Java* akan memiliki arti yang sama di setiap *platform*. Dalam teknologi bahasa pemrograman *Java*, *platform* yang tidak tergantung arsitektur dan mudah dibawa dikenal dengan nama *Java Virtual Machine*.

2.1.2.4 Berkinerja Tinggi

Kinerja selalu menjadi pertimbangan dalam pemrograman. *Java platform* mencapai kinerja yang tinggi dengan menggunakan skema dimana *interpreter* dapat bekerja dengan kecepatan penuh tanpa perlu memeriksa *run-time environment*. Sebuah *garbage collector* bekerja sebagai *background thread* dengan prioritas rendah untuk memastikan tersedianya memori pada saat dibutuhkan.

2.1.2.5 Interpreted, Threaded, dan Dinamis

Java interpreter dapat mengeksekusi *bytecodes* secara langsung pada semua jenis peralatan dimana *interpreter* dan sistem *runtime* dijalankan. Selain itu teknologi Java juga memiliki kemampuan *multithreading* yang memungkinkan pemrogram untuk membangun aplikasi dengan banyak aktifitas *thread* yang berjalan bersamaan.

Jika *Java Compiler* bersifat ketat dalam pemeriksaan kesalahan, sistem bahasa dan *runtime Java* bersifat dinamis pada tahap penghubungan. *Class* hanya dihubungkan seperlunya saja. Modul kode baru dapat dihubungkan sesuai permintaan dari berbagai sumber, termasuk dari sumber antar jaringan.

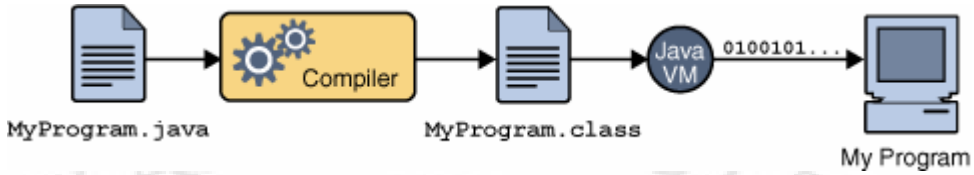
2.1.3 Java Platform

Sebuah *platform* adalah lingkungan perangkat keras atau lunak dimana program berjalan [3]. Sebagian besar *platform* dapat dijelaskan sebagai kombinasi dari sistem operasi dan perangkat keras di bawahnya. *Java platform* berbeda dari sebagian besar *platform* lainnya karena hanya terdiri dari perangkat lunak yang berjalan di atas *platform* perangkat keras lain. *Java platform* terdiri dari 2 bagian:

- *Java Virtual Machine*
- *Java Application Programming Interface* (API)

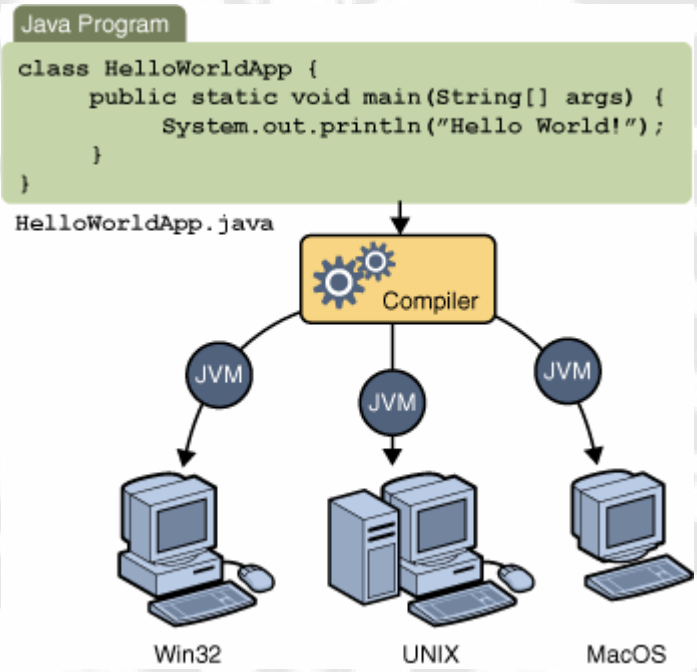
Pada bahasa pemrograman *Java*, semua kode sumber pertama-tama ditulis dalam file teks dangan ekstensi `.java`. Kode sumber tersebut kemudian diterjemahkan ke dalam *file* `.class` oleh `javac` *compiler*. Sebuah *file* `.class` tidak mengandung kode yang dapat dimengerti oleh prosesor, melainkan

bytecodes yang merupakan bahasa dari *Java Virtual Machine* (*Java* VM). Urutan proses pengembangan perangkat lunak pada Java diilustrasikan pada Gambar 2.2.



Gambar 2.2. Urutan proses pengembangan perangkat lunak dengan menggunakan *Java*

Karena *Java* VM tersedia pada banyak sistem operasi yang berbeda, maka file `.class` yang sama dapat dijalankan pada berbagai sistem operasi yang berbeda. Kemampuan ini diilustrasikan pada Gambar 2.3.

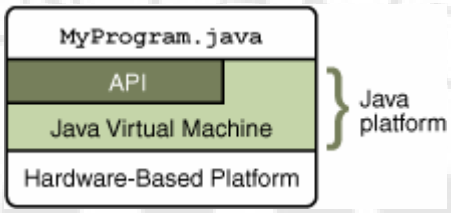


Gambar 2.3. Kemampuan *Java* untuk menjalankan aplikasi yang sama di beberapa *platform* yang berbeda

API merupakan sekumpulan perangkat lunak yang siap digunakan yang menyediakan banyak kegunaan. API dikelompokkan lagi menjadi kumpulan dari *class* dan antarmuka yang berhubungan yang disebut *package*.

Sebagai lingkungan yang tidak tergantung pada *platform*, *Java* dapat menjadi sedikit lebih lambat daripada bahasa pemrograman lain. Namun

perkembangan pada *compiler* dan *virtual machine* semakin membawa kinerja *Java* mendekati bahasa pemrograman lainnya dengan tidak mengurangi kemudahannya untuk dipindahkan. Susunan dari *Java platform* ditunjukkan pada Gambar 2.4.



Gambar 2.4. API dan *Java Virtual Machine* memisahkan program dari perangkat keras dibawahnya

2.2 PENGERTIAN *AGENT*

Dalam bidang ilmu komputer, istilah *agent* atau *agent* perangkat lunak adalah sebuah perangkat lunak yang bertindak untuk seorang pengguna atau program lainnya dalam hubungan perwakilan [4]. Tindakan sebagai perwakilan dari pangguna atau program lainnya berarti *agent* memiliki kuasa untuk menentukan apakah tindakannya tersebut benar. Sebuah *agent* dapat menentukan apakah dirinya akan bekerja sama dengan *agent* lain, atau berkompetisi untuk mencapai tujuannya sendiri.

Dari konsep di atas, sebuah *agent* memiliki sifat-sifat sebagai berikut [5]:

- otomatis:

  Sebuah *agent* harus dapat bekerja tanpa campur tangan secara langsung dari manusia, dan memiliki kemampuan untuk mengatur keadaan internalnya.
- sosial:

  Sebuah *agent* dapat bekerja sama dengan manusia atau *agent* lainnya dalam mencapai tujuannya,
- reaktif:

  Agent harus dapat mempelajari lingkungannya dan merespon secara berkala terhadap perubahan di lingkungannya,

- proaktif:

  Agent tidak hanya bertindak untuk merespon keadaan lingkungannya, tetapi juga dapat beringkah laku dalam mencapai suatu tujuan dengan mengambil inisiatif,
- bergerak:

  Agent dapat memiliki kemampuan untuk berpindah antara simpul yang berbeda pada jaringan komputer,
- rasional:

  Agent selalu bertindak untuk mencapai tujuannya dan tidak pernah mencegah pencapaian tujuannya,
- dapat belajar:

  Agent dapat menyesuaikan dirinya dengan lingkungannya dan dengan keinginan dari penggunanya.

2.3 JADE

JADE (*Java Agent DEvelopment Framework*) merupakan sebuah kerangka kerja perangkat lunak yang diimplementasikan sepenuhnya dalam bahasa *Java*. JADE memudahan implementasi dari *Multi-Agent System* (MAS) melalui *middle-ware* yang bekerja sesuai spesifikasi FIPA (*Foundation for Intelligent Physical Agent*).

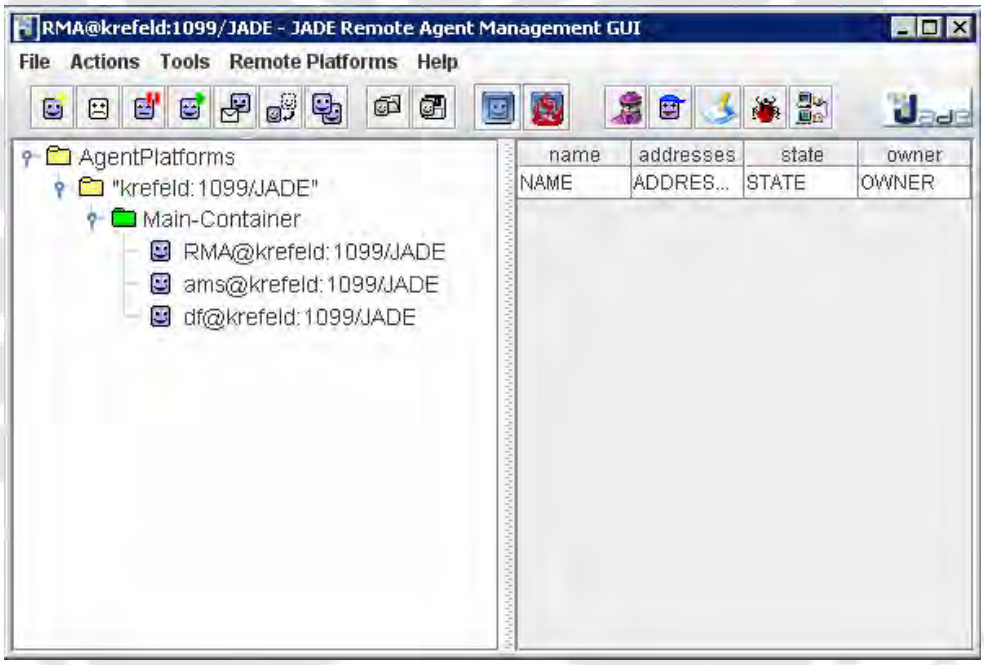
Karena sepenuhnya diimplementasikan dalam bahasa pemrograman *Java*, maka JADE juga mendapatkan seluruh keuntungan dari bahasa pemrograman tersebut, termasuk ketidak-tergantungan pada arsitektur *platform*. *Agent platform* pada JADE dapat didistribusikan di beberapa mesin yang berbeda, dan tidak perlu menggunakan sistem operasi yang sama.

Secara umum, JADE terdiri dari beberapa bagian, yaitu:

- sebuah *runtime environment* sebagai tempat hidupnya agent JADE yang harus diaktifkan terlebih dahulu sebelum satu atau lebih agent dapat dijalankan pada sebuah *host*,
- sekumpulan *class* (*library*) yang dapat digunakan oleh para pemrogram dalam mengembangkan *agent*, dan

- seperangkat peralatan dengan GUI (*Graphical User Interface*) untuk mengatur dan memantau aktifitas dari *agent* yang sedang berjalan.

Salah satu peralatan penting yang disediakan JADE adalah RMA (*Remote Monitoring Agent*) dan Introspector Agent. RMA berfungsi untuk mengawasi keadaan semua *agent* pada *platform* lokal maupun *remote* dan menyediakan beberapa pilihan untuk mengubah keadaan dan memindahkan *agent*. Sedangkan *Introspector Agent* berfungsi untuk menampilkan keadaan *agent* beserta seluruh *behaviour* yang aktif, dan pesan yang dikirim dan diterima. Tampilan RMA dan Introspector Agent ditunjukkan pada Gambar 2.5 dan 2.6.



Gambar 2.5. Tampilan *Remote Monitoring Agent*

Gambar 2.6. Tampilan *Introspector Agent*

2.3.1 *Container* dan *Platform*

Setiap bagian dari JADE *runtime environment* disebut *container*, karena dapat menampung beberapa *agent*. Sedangkan sekumpulan *container* yang aktif disebut *platform* [5]. Sebuah *main container* harus selalu aktif di dalam sebuah *platform*. Semua *container* lainnya pada *platform* yang sama akan mendaftar pada *main container* segera setelah semua *container* tersebut dijalankan. Oleh karena itu, main *container* harus menjadi *container* pertama yang dijalankan pada sebuah *platform* JADE.

Jika terdapat *main container* lain yang dijalankan di suatu tempat, maka *main container* tersebut akan membentuk pada *platform* yang berbeda. *Agent* pada JADE diidentifikasi oleh sebuah nama yang unik, dan dengan mengetahui nama agent lainnya maka agent tersebut dapat saling berkomunikasi meskipun terletak di *container* maupun *platform* yang berbeda. Ilustrasi mengenai container dan platform ditunjukkan pada Gambar 2.7.

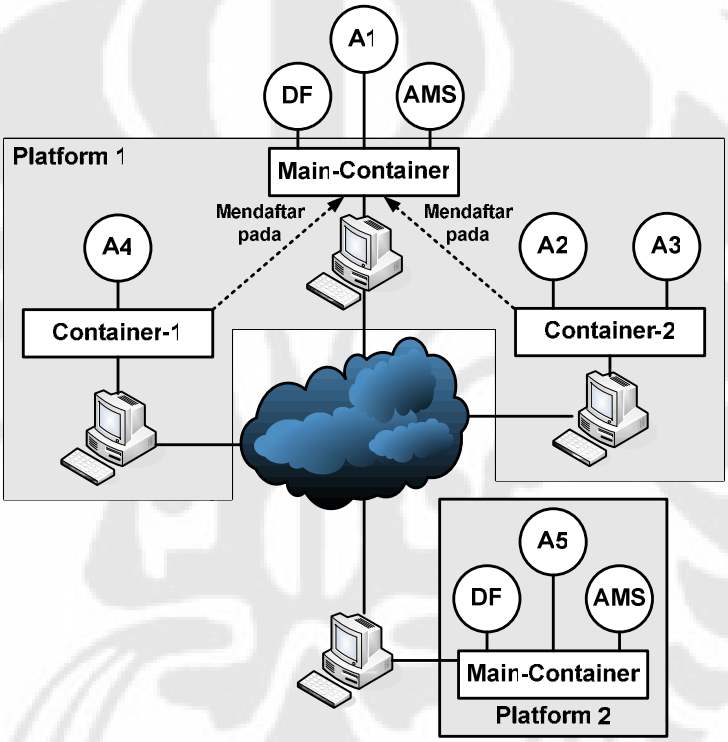
2.3.2 AMS dan DF

Perbedaan lain yang dimiliki *main container* dibandingkan dengan *container* lainnya adalah *main container* menampung 2 *agent* khusus yang

dijalankan secara otomatis saat *main container* dijalankan. Kedua *agent* tersebut adalah AMS (*Agent Management System*) dan DF (*Directory Facilitator*).

AMS menyediakan layanan penamaan untuk menjamin keunikan nama dari setiap *agent*, dan merepresentasikan pihak yang berwenang di dalam sebuah *platform*.

DF menyediakan layanan *Yellow Pages* dimana setiap *agent* dapat menemukan *agent* lainnya yang menyediakan layanan yang dibutuhkannya.



Gambar 2.7. *Container* dan *Platform*

2.3.3 *Agent Class*

Membuat sebuah *agent* pada JADE dapat dilakukan dengan mendefinisikan sebuah *class* dengan `extends jade.core.Agent`, dan mengimplementasikan `setup()` method seperti contoh di bawah ini.

```
import jade.core.Agent;
public class ContohAgent extends Agent {
      protected void setup() {
            // Mulai inisialisasi agent pada bagian ini
      }
}
```

Method `setup()` dimaksudkan untuk menginisialisasi *agent*. Sedangkan tugas *agent* yang sebenarnya terdapat pada *behaviour*.

2.3.4 Menjalankan *agent* JADE melalui *terminal*

Cara menjalankan *agent* pada platform JADE melalui *terminal* tidak jauh berbeda dengan cara menjalankan program *Java* pada umumnya.

Kode sumber dari *agent* dapat di-*compile* dengan perintah:

```
javac -classpath <JADE-CLASSES> ContohAgent.java
```

dimana `ContohAgent.java` adalah kode sumber dari *agent* yang akan dijalankan.

Untuk menjalankan *agent* yang telah di-*compile*, maka JADE *runtime* harus dijalankan terlebih dahulu, dan nama dari *agent* harus dipilih dengan perintah:

```
java -classpath <JADE-CLASSES>;. jade.Boot agent1:ContohAgent
```

Perintah di atas menjalankan JADE *runtime* dan *agent* `ContohAgent` dengan nama agent1.

Perintah di atas dapat digabung dengan beberapa perintah tambahan seperti:

- `-container`

  Digunakan untuk membuat *container* biasa yang akan bergabung dengan *platform* yang sudah ada.

- `-host <nama/alamat host>`

  Digunakan untuk menentukan nama *host* dari *main container* tujuan.

- `-port <nomor port>`

  Digunakan untuk menentukan nomor *port* yang digunakan oleh *main container* tujuan.

- `-local-host`

  Digunakan untuk menentukan *host* dimana *container* ini akan dijalankan. Nilai awalnya adalah *localhost*.

- `-local-port`

  Digunakan untuk menentukan nomor *port* dimana *container* ini dapat dihubungi. Nilai awalnya adalah 1099.

Daftar lengkap mengenai perintah yang dapat digunakan terdapat pada [6] beserta penjelasan masing-masing perintah.

2.3.5 *Agent Behaviour*

Sebuah *behaviour* merepresentasikan tugas yang dapat dilakukan oleh sebuah *agent* yang diimplementasikan sebagai *object* dari sebuah *class* dengan `extends jade.core.behaviours.Behaviour`. Method `addBehaviour()` digunakan pada *agent class* untuk menambahkan sebuah *behaviour* ke daftar tugas yang harus dilakukan oleh sebuah *agent*. *Behaviour* dapat ditambahkan setiap saat, baik saat *agent* baru dijalankan, maupun dari dalam *behaviour* lain.

Setiap *class* yang menggunakan *behaviour* harus mengimplementasikan *method* `action()` yang mendefinisikan kegiatan yang harus dilakukan ketika *behaviour* tersebut dijalankan. *Method* `done()` mengembalikan sebuah nilai `boolean` untuk menunjukkan apakah *behaviour* sudah selesai dijalankan dan dapat dikeluarkan dari daftar *behaviour* yang harus dilakukan oleh *agent*.

Behaviour pada JADE dibagi menjadi 3 jenis utama, yaitu: *one-shot behaviour, cyclic behaviour, dan generic behaviour* [5].

2.3.5.1 One-shot Behaviour

Behaviour ini dirancang untuk selesai dalam 1 fase eksekusi, dimana *method* `action()` hanya dieksekusi sekali saja. Pada behaviour ini *method* `done()` telah diimplementasikan dengan mengembalikan nilai `true`.

2.3.5.2 Cyclic Behaviour

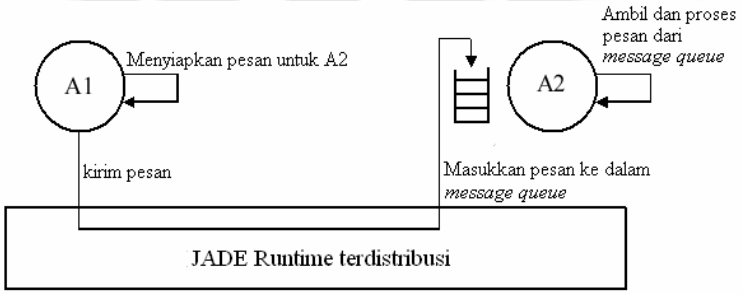
Behaviour ini dirancang agar tidak pernah selesai dengan *method* `action()` yang terus menjalankan operasi yang sama secara berulang-ulang. Pada *behaviour* ini *method* `done()` akan selalu mengembalikan nilai `false`.

2.3.5.3 Generic Behaviour

Behaviour ini menggunakan pemicu keadaan dan menjalankan operasi yang berbeda tergantung dari nilai keadaannya. Kondisi yang harus dipenuhi diletakkan pada *method* `done()` sehingga *behaviour* ini selesai jika kondisi yang disyaratkan sudah terpenuhi pada method `done()` mengambalikan nilai `true`.

2.3.6 Komunikasi *Agent*

Kemampuan berkomunikasi merupakan salah satu fitur paling penting yang dimiliki oleh *agent*. Paradigma komunikasi yang digunakan didasarkan pada *asynchronous message passing* (Gambar 2.8.), dimana setiap *agent* memiliki kotak surat (*message queue*) sebagai tempat JADE *runtime* meletakkan surat yang dikirim oleh *agent* lain. Setiap *message queue* menerima pesan, *agent* yang bersangkutan akan diberitahu. Tetapi kapan dan bagaimana agent mengambil surat dari *message queue* ditentukan sepenuhnya oleh pemrogram.



Gambar 2.8. *Asynchronous message passing* pada JADE

Format pesan yang digunakan pada JADE mengikuti ketentuan yang didefinisikan pada struktur pesan FIPA-ACL (*Agent Communication Language*). Parameter-parameter yang terdapat pada setiap pesan ACL ditunjukkan pada Tabel 2.1.

Tabel 2.1. Parameter pada pesan ACL

Parameter	Deskripsi
Performative	Jenis tindakan komunikatif dari pesan
sender	Identitas dari pengirim pesan
receiver	Identitas dari penerima pesan
reply-to	Agent yang menjadi tujuan diteruskannya pesan dalam urutan percakapan
content	Isi dari pesan
language	Bahasa yang digunakan untuk mengekspresikan parameter pada isi
encoding	Pengkodean spesifik dari isi pesan
ontology	Referensi dari *ontology* yang memberi arti dai simbol di isi pesan
protocol	Protokol interaksi yang digunakan untuk membentuk pecakapan
conversation-id	Identitas unik dari sebuah percakapan
reply-with	Ekspresi yang digunakan oleh agent yang merespon untuk mengidentifikasi pesan
in-reply-to	Referensi dari kegiatan sebelumnya yang menjadi penyebab pesan balasan
reply-by	Waktu/tanggal yang menunjukkan kapan pesan balasan harus diterima

2.3.6.1 *Mengirim Pesan*

Mengirim pesan kepada *agent* lain dilakukan dengan mengisi *parameter* yang terdapat pada objek `ACLMessage`, dan memanggil *method* `send()` dari *class* `Agent`. Contoh kode yang digunakan untuk mengirimkan pesan adalah sebagai berikut:

```
ACLMessage pesan = new ACLMessage();
pesan.setPerformative(ACLMessage.INFORM);
pesan.addReceiver(new AID("AgentTujuan", AID.ISLOCALNAME));
pesan.setContent("Isi Pesan");
send(pesan);
```

2.3.6.2 *Menerima Pesan*

Sebuah *agent* dapat mengambil pesan yang tedapat dalam *message queue* dengan menggunakan *method* `receive()`. *Method* ini mengembalikan pesan pertama yang terdapat di dalam *message queue*, atau `null` jika tidak terdapat pesan.

```
ACLMessage terima = receive();
If (terima != null) {
      // Proses pesan yang diterima
}
```

2.3.6.3 *Memilih Pesan dari Message Queue*

Sebuah *agent* dapat memilih pesan yang akan diambil dari *message queue* dengan menggunakan *template* bersama dengan *method* `receive()`. Ketika sebuah *template* ditentukan, maka *method* `receive()` akan mengembalikan pesan pertama yang sesuai dengan *template*, dan membiarkan pesan yang lain yang tidak sesuai. *Template* untuk menerima pesan diimplementasikan sebagai objek dari *class* `jade.lang.acl.MessageTemplate` yang menyediakan sekumpulan *method* untuk membentuk *template* secara sederhana dan fleksibel.

Contoh di bawah ini mengaplikasikan *template* untuk menerima pesan dengan *performative* CFP, dan membiarkan pesan-pesan lainnya:

```
Private MessageTemplate mt =
      MessageTemplate.MatchPerformative(ACLMessage.CFP));
```

```
Public void action() {
      ACLMessage terima = myAgent.receive(mt);
      if (terima != null) {
            // Pesan CFP diterima dan diproses
      }
      else {
            block();
      }
}
```

2.3.7 Layanan Yellow Pages

Layanan *yellow pages* pada JADE memungkinkan *agent* untuk menemukan *agent* lain yang tersedia pada suatu saat secara dinamis.

2.3.7.1 Agent DF

Layanan *yellow pages* memungkinkan *agent* untuk mengumumkan deskripsi dari satu atau lebih layanan yang disediakannya agar dapat ditemukan dan digunakan oleh *agent* lain.

Setiap *agent* dapat mengumumkan layanannya dan mencari layanan yang disediakan oleh *agent* lain. Pendaftaran, pembatalan, perubahan, dan pencarian dapat dilakukan setiap saat selama *agent* hidup.

Layanan *yellow pages* pada JADE disediakan oleh sebuah *agent* khusus, yaitu DF (*Directory Facilitator*). Setiap *platform* JADE harus memiliki sebuah *agent* DF awal dengan nama 'df@<nama-platform>'. *Agent* DF lainnya dapat dijalankan jika dibutuhkan, dan beberapa *agent* DF dapat disatukan untuk menyediakan katalog *yellow pages* tunggal.

2.3.7.2 Berinteraksi dengan DF

Karena DF merupakan *agent*, maka DF dapat berkomunikasi dengan *agent* lainnya dengan saling bertukar pesan ACL. Untuk mempermudah proses interaksi ini, JADE menyediakan `jade.domain.DFService` *class* yang memungkinkan pengumuman dan pencarian layanan melalui berbagai jenis pemanggilan *method*.

2.3.7.3 Mengumumkan layanan

Sebuah *agent* yang ingin mengumumkan layannya harus menyediakan AID, daftar layanannya, dan, jika diperlukan, juga daftar bahasa dan *ontology* yang harus digunakan oleh *agent* lain untuk berinteraksi dengannya.

Untuk mengumukan layanan, *agent* harus membuat deskripsi yang sesuai dan memanggil *static method* `register()` dari `DFService` *class*. Contoh pengumuman layanan ditunjukkan pada kode di bawah ini:

```
protected void setup() {
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("Jenis Layanan");
sd.setName("Nama Layanan");
dfd.addService(sd);
try {
      DFService.register(this, dfd);
}
catch (FIPAException fe) {
      fe.printStackTrace();
}
```

Saat *agent* diberhentikan, layanan yang terdaftar pada DF harus diberhentikan. Contoh kode untuk pemberhentian pengumuman layanan adalah sebagai berikut:

```
protected void takedown() {
try {
      DFService.deregister(this);
}
catch (FIPAException fe) {
      fe.printStackTrace();
}
```

2.3.7.4 Mencari layanan

Sebuah *agent* yang ingin mencari layanan harus menyediakan deskripsi layanan kepada DF berupa *template*. Hasil dari pencarian adalah daftar semua deskripsi yang sesuai dengan *template*.

Static *method* `search()` dari `DFService` *class* dapat digunakan seperti pada contoh berikut:

```
private Vector daftarAgent;
protected void setup() {
addBehaviour(new TickerBehaviour(this, 60000) {
      protected void onTick( {
            DFAgentDescription template =
                  new DFAgentDescription();
            ServiceDescription sd = new ServiceDescription();
            sd.setType("Jenis Layanan");
            template.addService(sd);
      try {
            DFAgentDescription[] hasil = DFService.search(myAgent,
template);
            daftarAgent.clear();
            for (int I = 0; I < hasil.length; ++i) {
                  daftarAgent.addElement(hasil[i].getName());
            }
      }
            catch (FIPAException fe) {
                  fe.printStacktrace();
            }
      }
});
}
```

Pada contoh di atas, mekanisme pencarian layanan dilakukan pada *class* `TickerBehaviour`, sehingga pencarian dilakukan secara berulang-ulang pada selang waktu yang telah ditentukan.

2.4 MYSQL

MySQL merupakan sebuah *relational database management system* (RDBMS) yang mendukung *multi-thread*, *multi-user*, dan SQL (*Structured Query Language*) *database server* [7].

MySQL dimiliki dan disponsori oleh perusahaan MySQL AB yang merupakan bagian dari *Sun Microsystems*. Kode sumber dari proyek MySQL

tersedia di bawah lisensi GPL (*GNU General Public License*), dan berbagai perjanjian *proprietary* lainnya.

2.4.1 Konsep Relational Model

Konsep *relational model* pertama kali diusulkan oleh Dr. Edgar F. Codd pada tahun 1970. Model ini kemudian dijelaskan dalam 12 peraturan yang dikenal dengan nama *Codd's Twelve Rules*.

Sebuah *database management system* (DBMS) yang ideal akan memenuhi semua peraturan tersebut. Tetapi, pada saat melakukan perancangan database, peraturan yang dianggap paling penting adalah peraturan pertama dan kedua saja [8]. Di bawah ini dijelaskan ringkasan dari kedua peraturan tersebut.

1) Peraturan 1

   Peraturan ini menyatakan bahwa data terletak di dalam tabel-tabel. Sebuah tabel mengelompokan informasi mengenai subyek tertentu. Baris mendeskripsikan informasi mengenai sebuah hal, sedangkan kolom mendeskripsikan karakteristik dari setiap hal. Perpotongan antara baris dan kolom berisi sebuah nilai dari atribut spesifik sebuah hal tunggal.

2) Peraturan 2

   Data tidak diambil atau direferensikan oleh lokasi fisik. Data harus diperoleh dengan mereferensikan tabel, *key* yang unik, dan satu atau beberapa nama kolom.

2.4.2 Instalasi MySQL *Community Server*

Untuk menjalankan MySQL *Community Server* sebagai *service* pada *Windows* dilakukan dengan membuat sebuah *file* konfigurasi dengan nama `my.ini`, atau menggunakan *file* konfigurasi awal yang sudah disediakan oleh MySQL. Untuk memudahkan proses instalasi, masukkan lokasi *folder* `bin` pada direktori `mysql` ke dalam `PATH` pada *environment variable Windows*.

Setelah *file* konfigurasi ditentukan, masukkan perintah:

```
mysqld --install <nama-service> --defaults-file=<file-konfigurasi>
```

Setelah *server* MySQL ter-*install* sebagai *service*, maka *server* MySQL akan dijalankan secara otomatis setiap kali sistem operasi dijalankan. *Server* MySQL sebagai *service* dapat dihentikan dengan menggunakan perintah:

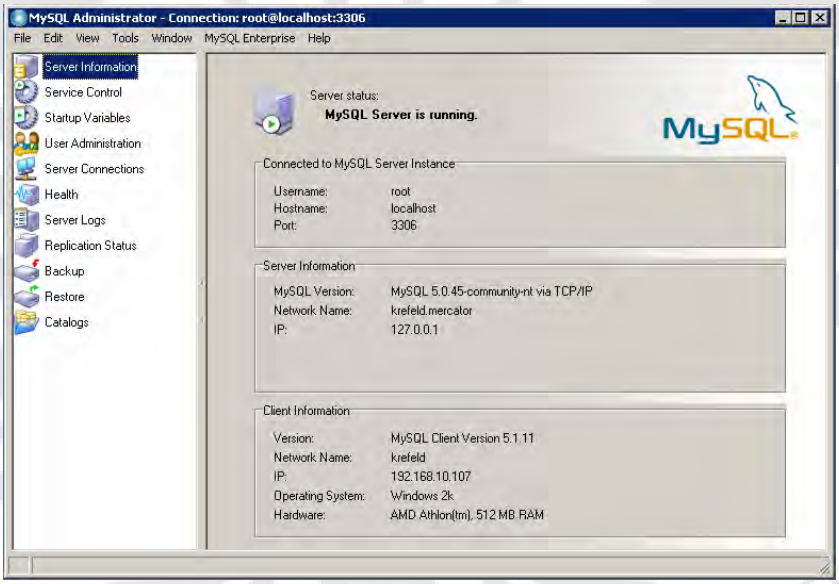
```
mysqld --remove.
```

Untuk mengakses *server* MySQL dengan menggunakan *command prompt*, dapat digunakan perintah:

```
mysql -u <user> -p <password>.
```

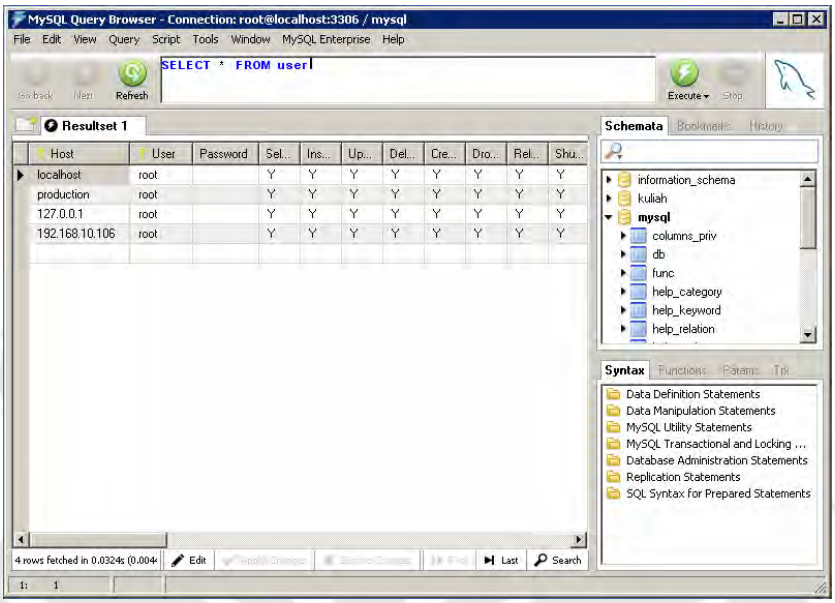
2.4.3 MySQL Administrator

MySQL *Administrator* merupakan sebuah program tambahan yang disediakan oleh MySQL untuk mempermudah pengawasan dan pengaturan lingkungan MySQL dengan menampilkan data secara grafis. Tampilan dari MySQL Administrator ditunjukkan pada Gambar 2.9.



Gambar 2.9. Tampilan *MySQL Administrator*

MySQL Administrator juga menyertakan MySQL *Query Browser* yang berguna untuk memasukkan, membaca, dan mengubah hasil *query* MySQL secara grafis. Tampilan MySQL Query Browser ditunjukkan pada Gambar 2.10.



Gambar 2.10. Tampilan *MySQL Query Browser*

2.4.4 JDBC

JDBC (*Java Database Connectivity*) merupakan API untuk bahasa pemrograman *Java* yang mendefinisikan cara megakses *database*. JDBC menyediakan beberapa *method* untuk mengambil dan mengubah data di dalam *database*.

2.4.4.1 Menghubungkan Java dengan MySQL

Untuk menghubungkan *Java* dengan MySQL, dibutuhkan *driver Connector-J* yang disediakan oleh MySQL. Sampai tulisan ini dibuat, *Connector-J* sudah tersedia sampai versi 5.2 secara gratis.

Mendefinisikan driver *Connector-J* dapat dilakukan dengan menggunakan perintah:

```
Class.forName("com.mysql.jdbc.driver").newInstance();
```

Hubungan antara *Java* dengan MySQL direpresentasikan sebagai objek dari `Connection` *class* pada paket `java.sql`. Objek *Connection* dapat diinisialisasi dengan menggunakan perintah:

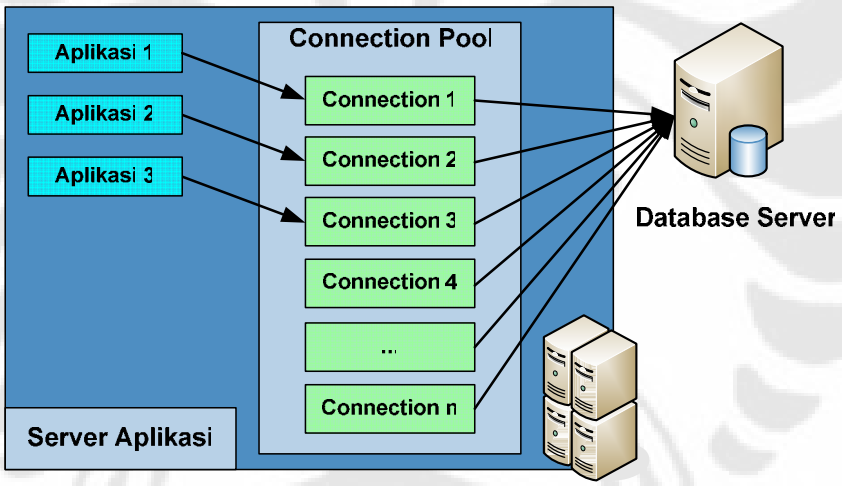
```
Connection conn =
      DriverManager.getConnection(url,username,password);
```

dengan format url sebagai berikut: `jdbc:mysql//<host>:<port>/<schema>`.

2.4.4.2 *Connection pooling*

Proses inisialisasi dan autentikasi hubungan pada MySQL memakan waktu yang cukup lama, sehingga dapat menurunkan kinerja pada sistem yang banyak melakukan akses terhadap *database* MySQL.

Untuk mengatasi masalah tersebut dapat digunakan *connection pooling*. *Connection pooling* merupakan teknik pemrograman yang menggunakan sebuah penampungan yang berisi objek-objek dari `Connection` *class* yang telah diibuat sebelumnya. Setiap program yang ingin mengakses *database* dapat mengambil objek `Connection` dari penampungan. Dengan cara ini waktu yang diperlukan untuk proses inisialisasi dan autentikasi saat pembentukan objek `Connection` dapat dihilangkan. Prinsip kerja connection pooling diiliustrasikan pada Gambar 2.11.



Gambar 2.11. *Connection pooling*

BAB III
PERENCANAAN, ANALISA, DAN PERANCANGAN APLIKASI SISTEM MANAJEMEN KELAS

3.1 SKENARIO APLIKASI SISTEM MANAJEMEN KELAS

Aplikasi Sistem Manajemen Kelas berbasis agent merupakan sebuah aplikasi yang ditujukan untuk mengotomatisasi proses pencatatan kegiatan perkuliahan yang berlangsung di Fakultas Teknik, Universitas Indonesia. Kegiatan yang diotomatisasi oleh Sistem Manajemen Kelas antara lain: pencarian kelas untuk kuliah pengganti, pencatatan absen mahasiswa, pencatatan absen dosen, dan pencatatan informasi kuliah pengganti.

Pada skenario Sistem Manajemen Kelas, terdapat satu buah komputer di setiap ruang kelas. Pada saat sebuah ruang kuliah dijadwalkan untuk digunakan sebagai tempat kegiatan peruliahan, maka komputer di ruang kelas tersebut akan menampilkan informasi yang berhubungan dengan mata kuliah yang sedang berlangsung. Bersamaan dengan dimulainya kegiatan perkuliahan, maka komputer di ruang kelas akan menghubungi komputer di departemen yang sesuai dengan kuliah yang berlangsung. Komputer di departemen kemudian akan mencatat dan menampilkan informasi mengenai kuliah yang sedang berlangsung tersebut.

Setiap peserta kegiatan perkuliahan dapat mengisi daftar hadir dengan memasukkan NPM atau NIP pada komputer dengan menggunakan *keyboard*. Jika NPM atau NIP tersebut terdaftar sebagai peserta kegiatan perkuliahan, maka NPM atau NIP tersebut akan dicatat. Setiap NPM atau NIP yang tidak tercatat di dalam daftar peserta tidak akan diproses.

Dosen atau petugas di departemen juga dapat mengirimkan pesan mangenai perubahan atau informasi tambahan lainnya melalui komputer yang terdapat di masing-masing departemen. Pesan yang dikirimkan akan ditampilkan pada layar monitor dari komputer di ruang kelas yang dituju, sehingga dapat dilihat oleh mahasiswa yang berkepentingan.

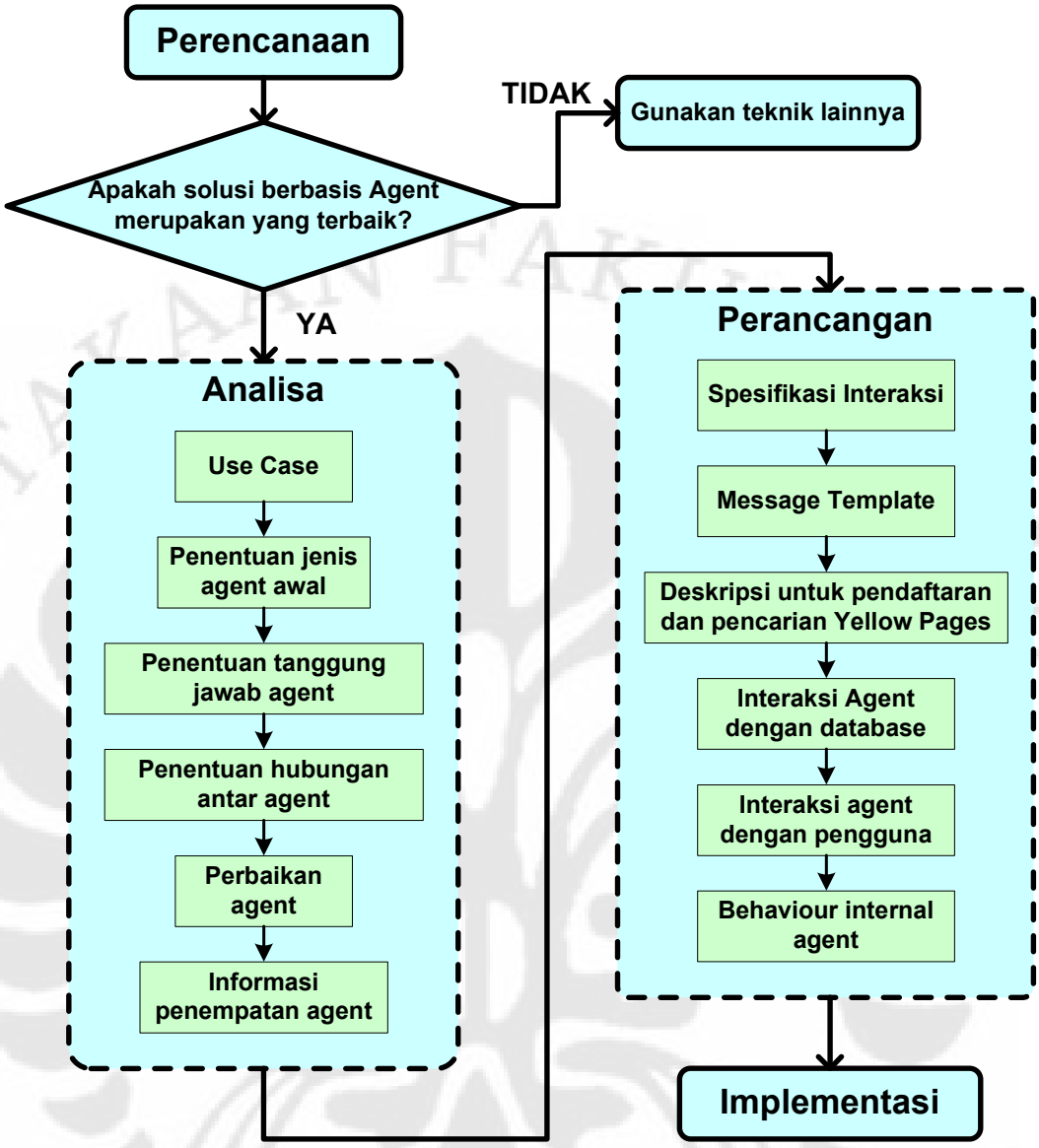
Komputer di masing-masing departemen juga memiliki hak untuk membatalkan serta mengubah jadwal dan lokasi kuliah yang merupakan bagian dari departemen tersebut. Pembatalan dan perubahan jadwal kuliah dilakukan dengan terlebih dahulu memasukkan informasi mengenai kuliah yang akan diubah atau dibatalkan. Jika informasi yang dimasukkan sesuai, maka pembatalan atau perubahan jadwal kuliah akan dilakukan.

Saat kegiatan perkuliahan sedang berlangsung, mahasiswa tetap dapat mengsi daftar absen selama belum melewati batas waktu pengisian daftar absen. Batas waktu pengisian absen ditetapkan oleh sistem selama 30 menit, tetapi dosen dapat mengubah nilai batas waktu tersebut.

Untuk mengetahui keadaan kuliah yang sedang berlangsung, dosen atau petugas di departemen dapat melakukan polling terhadap komputer yang berada di masing-masing kelas. Komputer yang terdaftar pada departemen tersebut kemudian akan membalas dengan informasi kuliah yang sedang berlangsung, jumlah siswa yang sudah hadir, dan kehadiran dosen di kelas.

3.2 METODOLOGI YANG DIGUNAKAN

Tahap perancangan aplikasi Sistem Manajemen Kelas dilakukan dengan mengacu pada sebuah *whitepaper* oleh Magid Nikraz *et.al.* [9], yang membahas sebuah metodologi yang digunakan dalam menganalisa dan merancang sebuah sistem *multi-agent*. Berdasarkan *whitepaper* tersebut, terdapat 6 langkah yang harus dilakukan pada tahap analisa, dan 10 langkah pada tahap perancangan. Tahap perencanaan, implementasi, dan pengujian tidak dijelaskan pada *whitepaper* ini. Gambaran umum dari semua tahap tersebut ditunjukkan pada Gambar 3.1.



Gambar 3.1 Gambaran umum dari metodologi untuk menganalisa dan merancang sistem *multi-agent*

Dari semua tahapan yang diperlihatakan pada Gambar 3.1, tidak seluruhnya dijelaskan pada skripsi ini karena adanya pembagian tugas, sehingga sebagian tahap perancangan akan dikerjakan oleh pemrogram lain dalam satu tim. Tetapi seluruh tahapan pada proses analisa akan dijelaskan untuk memberi gambaran umum yang utuh mengenai Sistem Manajemen Kelas.

3.3 TAHAP PERENCANAAN

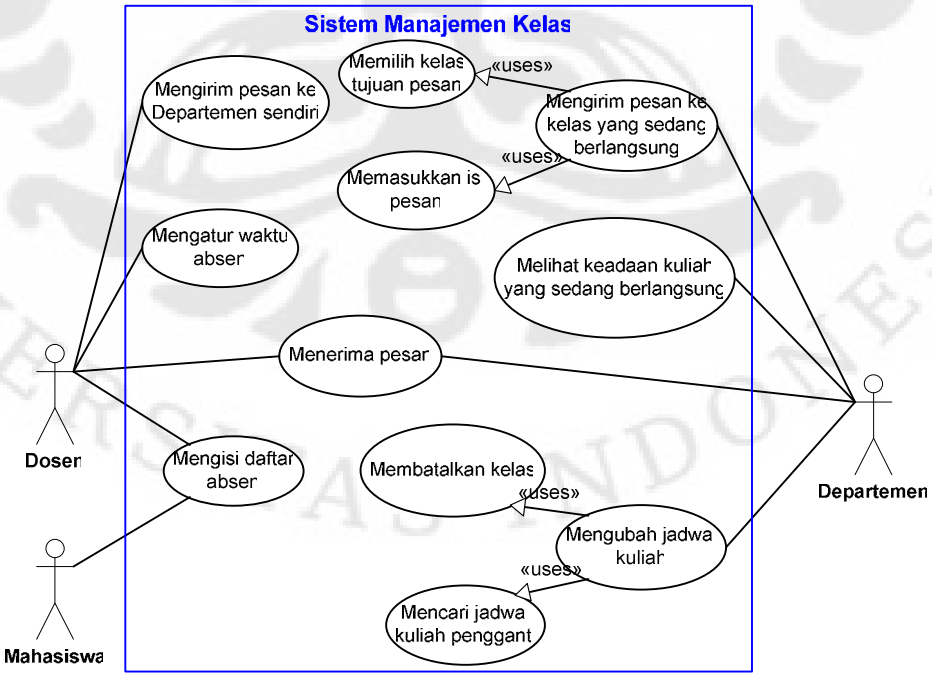
Tujuan dari pengembangan aplikasi Sistem Manajemen Kelas adalah untuk mengeksplorasi kemampuan dan fitur-fitur yang dimiliki sistem *multi-agent*. Oleh sebab itu, pertanyaan yang terdapat pada tahap perencanaan, yaitu: "Apakah solusi berbasis *agent* merupakan solusi yang terbaik?" tidak digunakan sebagai dasar pemilihan *agent* sebagai dasar pengembangan aplikasi Sistem Manajemen Kelas.

3.4 TAHAP ANALISA

Tahap analisa bertujuan untuk menentukan batasan tanggung jawab yang jelas untuk setiap *agent* yang digunakan, dan mendefinisikan hubungan antara satu agent dengan *agent* lainnya.

3.4.1 *Use Case*

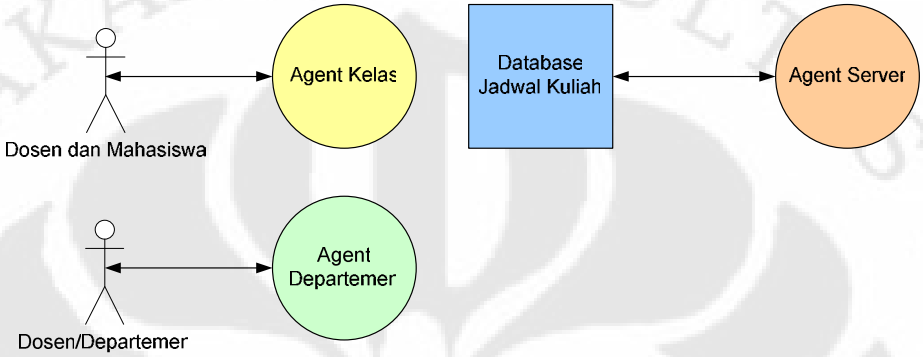
Berdasarkan deskripsi pada skenario aplikasi Sistem Manajemen Kelas, dapat diketahui fungsi-fungsi yang dimiliki oleh Sistem Manajemen Kelas. Untuk memperjelas interaksi yang dapat dilakukan pengguna dengan Sistem Manajemen Kelas, digunakan diagram *use case* yang ditunjukkan pada Gambar 3.2.



Gambar 3.2. Diagram *use case* untuk Sistem Manajemen Kelas

3.4.2 Penentuan Jenis *Agent* Awal

Penetuan jenis *agent* awal dilakukan dengan menugaskan satu jenis agent untuk setiap pengguna dan sumber daya lainnya, seperti *server database*. Hasil penentuan jenis *agent* digambarkan dengan menggunakan diagram *agent* yang ditunjukkan pada Gambar 3.3.



Gambar 3.3. Diagram *agent* untuk Sistem Manajemen Kelas

Agent Server pada Gambar 3.3 berfungsi untuk sumber daya di luar *platform agent*, yaitu *server database*. Untuk melaksanakan tugas tersebut, pendekatan *transducer* digunakan. Pada pendekatan ini, *agent* berfungsi sebagai antarmuka bagi *agent* lain yang ingin berhubungan dengan *server database*, sehingga *agent* lain cukup menggunakan pesan ACL.

3.4.3 Penentuan Tanggung Jawab Masing-masing *Agent*

Pada tahap ini ditentukan tanggung jawab utama dari masing-masing *agent* yang ditentukan pada tahap sebelumnya. Proses ini menghasilkan tabel tanggung jawab yang ditunjukan pada Tabel 3.1.

Tabel 3.1. Tabel tanggung jawab *agent* setelah tahap ke-3

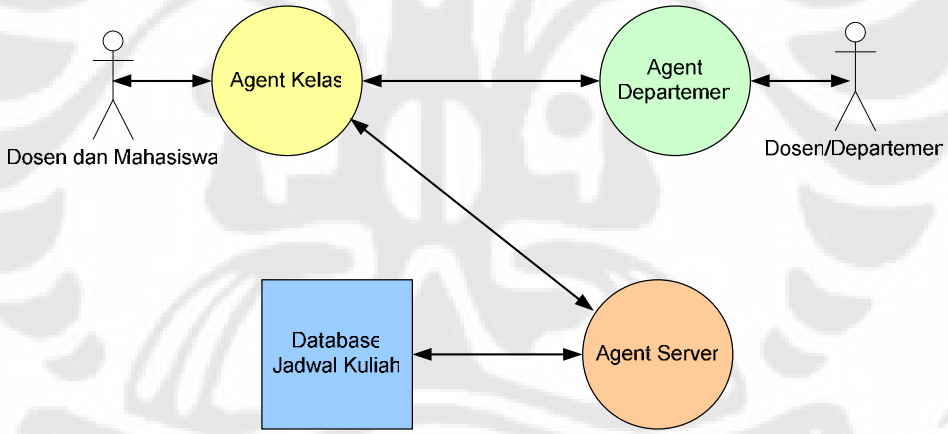
Jenis *Agent*	Tanggung Jawab
Agent Kelas	• Menampilkan informasi kelas yang sedang berlangsung • Mencatat informasi absen mahasiswa dan dosen • Memberikan pilihan untuk mengubah batas waktu pengisian absen kepada dosen

Tabel 3.1. Tabel tanggung jawab *agent* setelah tahap ke-3 (lanjutan)

Jenis *Agent*	Tanggung Jawab
Agent Departemen	• Memproses permintaan pembatalan jadwal kuliah • Memproses permintaan perubahan dan penambahan jadwal kuliah

3.4.4 Penentuan Hubungan Antar *Agent*

Pada tahap ini ditentukan hubungan yang terjadi antar *agent*. Salah satu hubungan antar *agent* pada Sistem Manajemen Kelas adalah hubungan *Agent* Kelas dengan *Agent Server* untuk mengambil dan mencatat data perkuliahan ke dalam *server database*. Hubungan lainnya adalah antara *Agent* Kelas dengan *Agent* Departemen untuk mengirimkan informasi mengenai kuliah yang sedang berlangsung. Diagram *agent* yang telah diperbaharui dengan hubungan antar *agent* ditunjukkan pada Gambar 3.4.



Gambar 3.4. Diagram *agent* untuk Sistem Manajemen Kelas setelah tahap ke-4

Dari penentuan hubungan antar agent, tabel tanggung jawab *agent* dapat diperbaharui dengan tanggung jawab yang berhubungan dengan agent lainnya. Tabel tanggung jawab yang telah diperbaharui ditunjukkan pada Tabel 3.2.

Tabel 3.2. Tabel tanggung jawab *agent* setelah tahap ke-4

Jenis *Agent*	Tanggung Jawab
Agent Kelas	• Menampilkan informasi kelas yang sedang berlangsung • Mencatat informasi absen mahasiswa dan dosen • Memberikan pilihan untuk mengubah batas waktu pengisian absen kepada dosen • Mengirim pesan ke *Agent* Departemen • Menerima pesan dari *Agent* Departemen • Meminta jadwal kuliah harian kepada *Agent Server* • Menjawab permintaan pembatalan kuliah dari *Agent* Departemen • Menjawab permintaan perubahan dan penambahan jadwal kuliah dari *Agent* Departemen • Mengirim informasi pembatalan kuliah ke *Agent Server* • Mengirim informasi perubahan jadwal kuliah ke *Agent Server* • Mendaftar kepada *Agent* Departemen yang sesuai saat ada kuliah yang aktif • Menjawab *polling* dari *Agent* Departemen
Agent Departemen	• Memproses permintaan pembatalan jadwal kuliah • Memproses permintaan perubahan jadwal kuliah • Mengirim pesan ke kelas yang terdaftar • Menerima pesan dari *Agent* Kelas • Melakukan *polling* terhadap *Agent* Kelas yang terdaftar
Agent Server	• Melayani permintaan daftar jadwal kuliah harian dari *Agent* Kelas • Mencatat penambahan daftar absen dari *Agent* Kelas ke dalam *database* • Mencatat pembatalan jadwal kuliah dari *Agent* Kelas ke dalam *database* • Mencatat perubahan dan penambahan jadwal kuliah dari *Agent* Kelas ke dalam *database*

3.4.5 Perbaikan *Agent*

Pada tahap ini, semua *agent* yang ditentukan pada tahap ke-2 diperbaiki dengan mempertimbangkan hal-hal berikut [8]:

- dukungan: informasi tambahan yang dibutuhkan *agent* untuk melakukan tanggung jawabnya,
- penemuan: cara *agent* yang saling berhubungan menemukan pasangannya,
- pengawasan: kebutuhan sistem untuk mengawasi keadaan *agent* yang aktif.

Hal-hal yang menjadi pertimbangan di atas akan dijelaskan pada Sub Bab 3.4.5.1, 3.4.5.2, dan 3.4.5.3.

3.4.5.1 Dukungan

Semua informasi yang digunakan oleh *agent* pada Sistem Manajemen Kelas tersedia pada *server database*, sehingga setiap *agent* tidak memerlukan informasi tambahan untuk melakukan tugasnya.

3.4.5.2 Penemuan

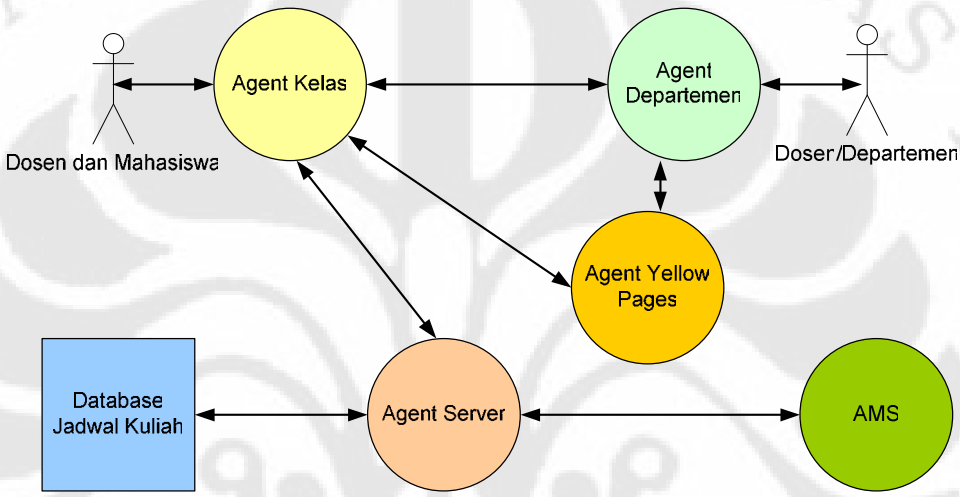
Agar setiap agent pada Sistem Manajemen Kelas dapat menemukan *agent* lainnya, maka dapat digunakan 2 cara, yaitu:

- menetapkan konvensi penamaan, dan
- melakukan pendaftaran dan pencarian pada *Yellow Pages.*

Agent Server dan *Agent* Departemen merupakan penyedia layanan, sehingga kedua *agent* tersebut harus mendaftarkan layanannya pada *Agent Yellow Pages*. Sedangkan *Agent* Kelas lebih berfungsi sebagai pengguna layanan, sehingga tidak perlu mendaftar kepada *Agent Yellow Pages*. Penamaan *Agent* Kelas cukup dilakukan dengan menggunakan konvensi, yaitu sesuai dengan nama kelas tempat *agent* tersebut berada.

3.4.5.3 Pengawasan

Pada Sistem Manajemen Kelas, pengawasan perlu dilakukan terhadap *Agent* Kelas yang aktif. Kewajiban untuk mengawasi diserahkan kepada *Agent Server*, dan untuk melakukan pengawasan *Agent Server* perlu berhubungan dengan AMS untuk mendapatkan informasi mengenai semua agent yang aktif. Diagram agent dan tabel kewajiban yang dihasilkan pada tahap ini ditunjukkan pada Gambar 3.5 dan Tabel 3.3.



Gambar 3.5. Diagram *agent* untuk Sistem Manajemen Kelas setelah tahap ke-5

Tabel 3.3. Tabel kewajiban setelah tahap ke-5

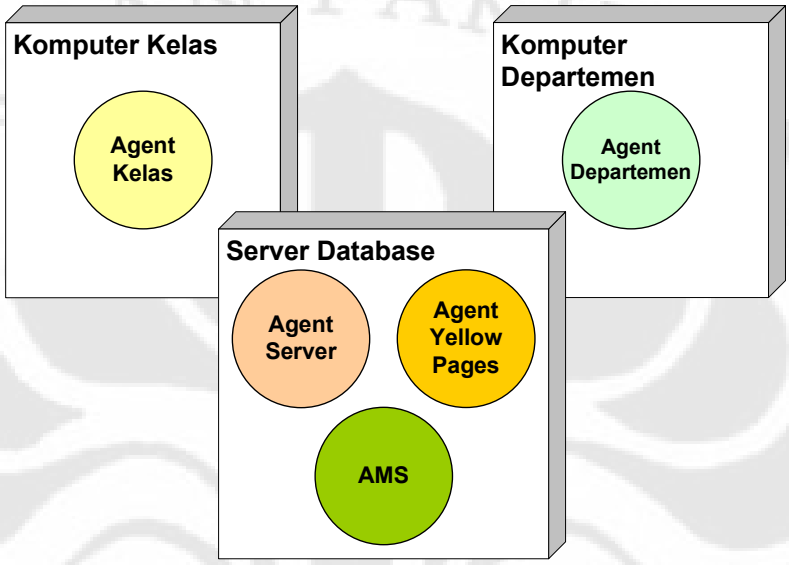
Jenis *Agent*	Tanggung Jawab
Agent Kelas	1. Menampilkan informasi kelas yang sedang berlangsung 2. Mencatat informasi absen mahasiswa dan dosen 3. Memberikan pilihan untuk mengubah batas waktu pengisian absen kepada dosen 4. Mengirim pesan ke *Agent* Departemen 5. Menerima pesan dari *Agent* Departemen 6. Meminta jadwal kuliah harian kepada *Agent Server* 7. Menjawab permintaan pembatalan kuliah dari *Agent* Departemen 8. Menjawab permintaan perubahan dan penambahan jadwal kuliah dari *Agent* Departemen

Tabel 3.3. Tabel kewajiban setelah tahap ke-5 (lanjutan)

Jenis *Agent*	Tanggung Jawab
	9. Mengirim informasi pembatalan kuliah ke *Agent Server* 10. Mengirim informasi perubahan dan penambahan jadwal kuliah ke *Agent Server* 11. Mendaftar kepada *Agent* Departemen yang sesuai saat ada kuliah yang aktif 12. Menjawab *polling* dari *Agent* Departemen 13. Mencari *Agent* Server melalui *Yellow Pages* 14. Mencari *Agent* Departemen yang sesuai melalui *Yellow Pages*
Agent Departemen	1. Memproses permintaan pembatalan jadwal kuliah 2. Memproses permintaan perubahan dan penambahan jadwal kuliah 3. Mengirim pesan ke kelas yang terdaftar 4. Menerima pesan dari *Agent* Kelas 5. Melakukan *polling* terhadap *Agent* Kelas yang terdaftar 6. Mendaftar pada *Agent Yellow Pages*
Agent Server	1. Melayani permintaan daftar jadwal kuliah harian dari *Agent* Kelas 2. Mencatat penambahan daftar absen dari *Agent* Kelas ke dalam *database* 3. Mencatat pembatalan jadwal kuliah dari *Agent* Kelas ke dalam *database* 4. Mencatat perubahan dan penambahan jadwal kuliah dari *Agent* Kelas ke dalam *database* dan mengirim informasi kuliah ke *Agent* Kelas jika perubahan jadwal berlaku untuk hari yang sama. 5. Mendaftar pada *Agent Yellow Pages* 6. Mencari *Agent* Kelas yang aktif melalui AMS

3.4.6 Informasi Penempatan *Agent*

Pada tahap ini ditentukan *host* atau komputer sebagai lokasi penempatan masing-masing *agent* dengan menggunakan diagram penempatan *agent*. Diagram ini bertujuan untuk ditunjukkan pada Gambar 3.6.



Gambar 3.6. Diagram penempatan *agent* untuk Sistem Manajemen Kelas

3.5 TAHAP PERANCANGAN *AGENT SERVER*

Tahap perancangan bertujuan untuk menentukan solusi dari spesifikasi yang ditentukan pada tahap analisa. Beberapa tahapan perancangan tidak dijelaskan karena tidak termasuk dalam batasan pengerjaan skripsi. Perancangan yang dijelaskan pada skripsi ini terbatas pada perancangan *Agent Server*.

3.5.1 Spesifikasi Interaksi *Agent Server*

Pada tahap ini, semua tanggung jawab *Agent Server* yang berhubungan dengan komunikasi antar *agent* diperhitungkan, dan ditentukan spesifikasinya. Hasil penentuan spesifikasi interaksi ditulikan pada tabel interaksi *Agent Server* (Tabel 3.4).

Tabel 3.4. Tabel interaksi *Agent Server*

Interaksi	No	Peran	Pasangan	Pemicu
Mengirim daftar jadwal harian	1	R	*Agent* Kelas	Permintaan jadwal kuliah diterima
Mengirim daftar jadwal selama satu semester	1	R	*Agent* Kelas	Permintaan jadwal kuliah diterima
Mengirim informasi kuliah yang baru ditambahkan atau mengalami perubahan jadwal	4	I	*Agent* Kelas	Terjadi perubahan atau penambahan jadwal kuliah untuk hari yang sama

3.5.2 Message Template *Agent Server*

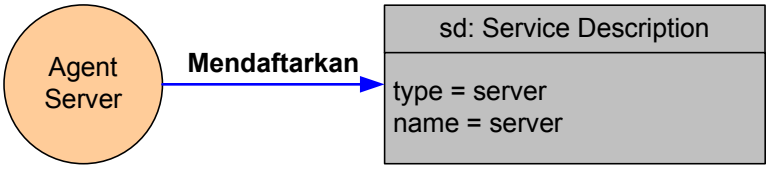
Semua spesifikasi interaksi pada subbab sebelumnya diimplementasikan sebagai behaviour. Pada tahap ini ditentukan objek `MessageTemplate` yang sesuai untuk menerima setiap pesan pada tabel interaksi. Spesifikasi `MessageTemplate` tersebut kemudian ditambahkan pada tabel interaksi agent server (Tabel 3.5).

Tabel 3.5. Tabel interaksi *Agent Server* dengan *message template*

Interaksi	No	Peran	Pasangan	Pemicu	Message Template
Mengirim daftar jadwal harian	1	R	*Agent* Kelas	Permintaan jadwal kuliah diterima	Perf = PROPOSE Conv-id Ontology Language
Mengirim daftar jadwal selama satu semester	1	R	*Agent* Kelas	Permintaan jadwal kuliah diterima	Perf = INFORM Conv-id Ontology Language
Mengirim informasi kuliah yang baru ditambahkan atau mengalami perubahan jadwal	4	I	*Agent* Kelas	Terjadi perubahan atau penambahan jadwal kuliah untuk hari yang sama	Perf = CONFIRM Conv-id Ontology Language

3.5.3 Deskripsi untuk Pendaftaran dan Pencarian pada *Yellow Pages*

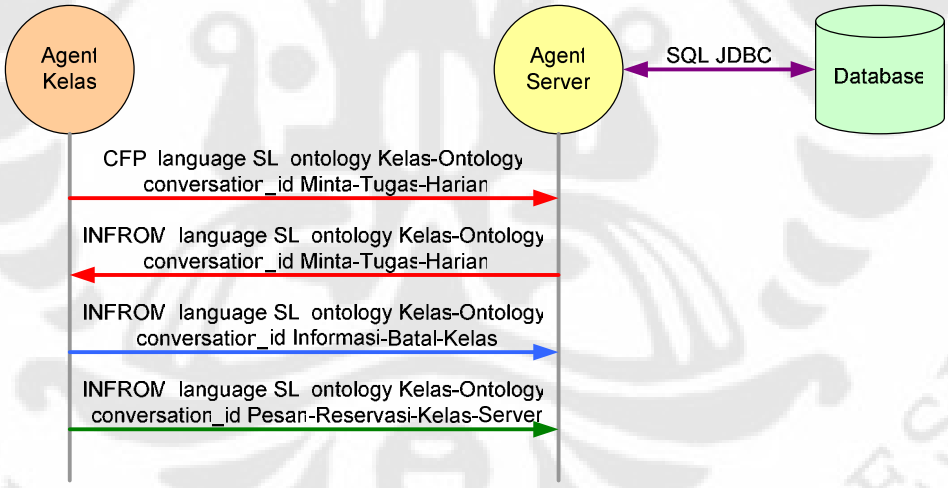
Pada subbab 3.4.5 Agent Server harus mendaftarkan layanannya kepada *Agent Yellow Pages*. Spesifikasi layanan yang akan didaftarkan dimasukkan ke dalam objek `ServiceDescription`. Gambaran `ServiceDesciption` yang digunakan *Agent Server* ditunjukkan pada Gambar 3.7.



Gambar 3.7. Pendaftaran layanan *Agent Server* ke *Agent Yellow Pages* pada Sistem Manajemen Kelas

3.5.4 Interaksi *Agent* dengan *Database*

Interaksi antara *Agent* Kelas dengan database dilakukan dengan menggunakan *Agent* Server sebagai *transducer*. Karena jenis komunikasi antara *Agent* Kelas dengan *database* terbatas pada beberapa jenis *query* dan *update* yang jenisnya selalu tetap, maka *Agent* Kelas cukup mengirimkan pesan ACL dengan *template* yang sesuai kepada *Agent Server*. Untuk membedakan jenis interaksi dengan *database* digunakan *conversation-id* yang berbeda. Gambaran interaksi *Agent* Kelas dengan *database* untuk 3 jenis interaksi yang berbeda digambarkan pada Gambar 3.8.



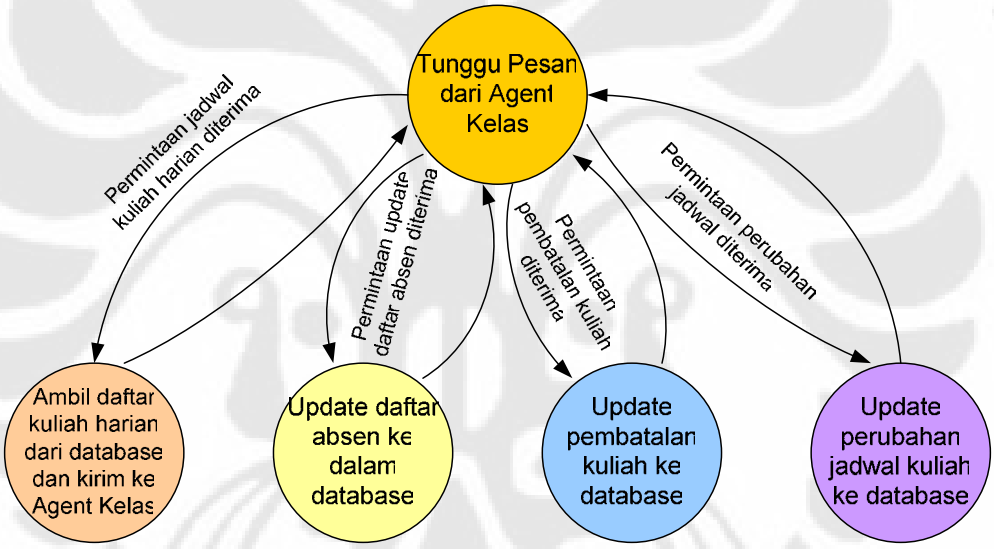
Gambar 3.8. Diagram interaksi *Agent* Kelas dengan *database* melalui dengan *Agent Server* sebagai perantara

3.5.5 Interaksi *Agent* dengan Pengguna

Pada Sistem Manajemen Kelas antara pengguna dengan *agent* terjadi secara lokal, sehingga antarmuka yang digunakan cukup berupa GUI lokal. Agar GUI dapat bertukar data dengan *agent*, maka referensi dari objek *agent* yang bersangkutan perlu disertakan dalam *constructor* saat objek GUI diciptakan.

3.5.6 *Behaviour* Internal *Agent Server*

Pada tahap ini, tanggung jawab dari *Agent Server* dipetakan menjadi behaviour-behaviour yang dijalankan *Agent Server*. Karena sebagian besar tugas dari *Agent Server* adalah melayani 4 jenis permintaan dari *Agent Kelas*, maka dipelukan 4 buah `CyclicBehaviour` yang berfungsi untuk menunggu pesan yang sesuai dengan *message template* masing-masing, dan 4 `OneShotBehaviour` yang akan dipanggil untuk memproses permintaan tersebut. Diagram peralihan keadaan yang menunjukkan proses menunggu permintaan dari *Agent* Kelas ditunjukkan pada Gambar 3.9.



Gambar 3.9. Diagram peralihan keadaan *Agent* Server untuk tanggung jawab melayani permintaan *Agent* Kelas

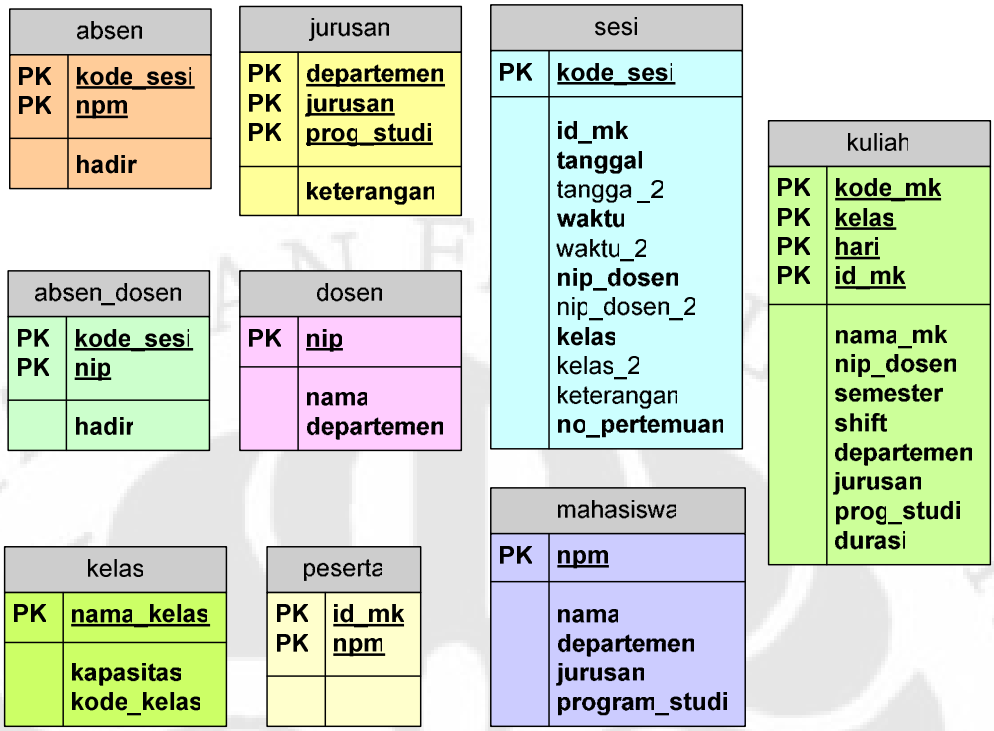
BAB IV
IMPLEMENTASI *DATABASE*, *AGENT SERVER*, DAN GUI PADA SISTEM MANAJEMEN KELAS

4.1 IMPLEMENTASI *SERVER DATABASE*

Implementasi *server database* pada Sistem Manajemen Kelas menggunakan *MySQL Community Server 5.0.45*. Selain itu, digunakan juga beberapa perangkat lunak lainnya untuk membantu proses perancangan struktur data pada *database*, yaitu *MySQL Administrator* dan *MySQL Query Browser*. *MySQL Administrator* digunakan untuk merancang struktur database melalui GUI yang telah disediakan, sedangkan *MySQL Query Browser* digunakan untuk menguji SQL *statement* yang akan digunakan pada *Agent Server*.

Database pada Sistem Manajemen Kelas digunakan untuk menampung semua informasi yang berhubungan dengan kegiatan perkuliahan, termasuk informasi mengenai peserta kuliah dan ruang kelas yang tersedia. Karena banyaknya informasi yang disimpan pada *database*, maka perlu dirancang *database* dengan struktur tabel yang efisien untuk menghindari terjadinya duplikasi data yang dapat menghabiskan memori tempat penyimpanan *database*.

Proses perancangan struktur tabel dimulai dengan menentukan semua informasi yang dibutuhkan pada pelaksanaan kegiatan perkuliahan. Dari daftar semua informasi yang diperlukan kemudian dilakukan pengelompokkan data menjadi tabel-tabel. Proses pengelompokkan data menghasilkan struktur tabel yang ditunjukkan pada Gambar 4.1.

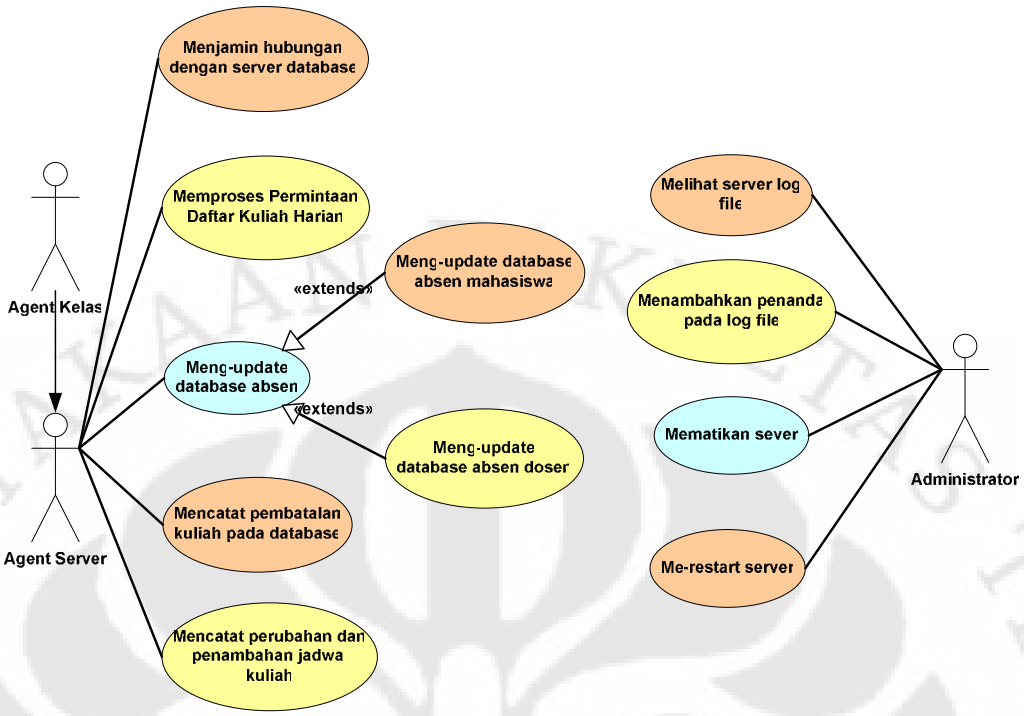


Gambar 4.1. Struktur *database* untuk Sistem Manajemen Kelas

Pada struktur tabel di atas digunakan *primary key* di masing-masing tabel untuk mencegah dimasukkannya informasi yang memiliki arti sama ke dalam *database*.

4.2 IMPLEMENTASI *AGENT SERVER*

Proses implementasi *Agent Server* dimulai dengan menentukan kembali fungsi-fungsi utama dari *Agent Server* sebagai bagian dari Sistem Manajemen Kelas. Fungsi-fungsi *Agent Server* ditunjukkan dengan menggunakan diagram *use case* pada Gambar 4.2.



Gambar 4.2. Diagram *use case* untuk *Agent Server*

Pada diagram *use case* di atas terdapat beberapa fungsi yang tidak terdapat pada bagian perancangan, seperti menjamin hubungan dengan *database* dan fungsi-fungsi untuk *administrator*. Fungsi-fungsi tambahan tersebut bukan merupakan tanggung jawab utama dari *Agent Server*, tetapi dimaksudkan untuk mempermudah pengelolaan *Agent Server*.

4.2.1 Behaviour untuk Tanggung Jawab Utama *Agent Server*

Dengan mengacu pada tabel tanggung jawab *Agent Server* pada bab sebelumnya, dapat ditentukan beberapa *behaviour* yang harus dimiliki oleh *Agent* Server beserta jenisnya. Daftar *behaviour* yang dimiliki *Agent Server* untuk memenuhi tanggung jawab utamanya ditunjukkan pada Tabel 4.1.

Tabel 4.1. Daftar *behaviour* untuk tanggung jawab utama *Agent Server*

Tanggung jawab	Behaviour Class	Jenis Behaviour

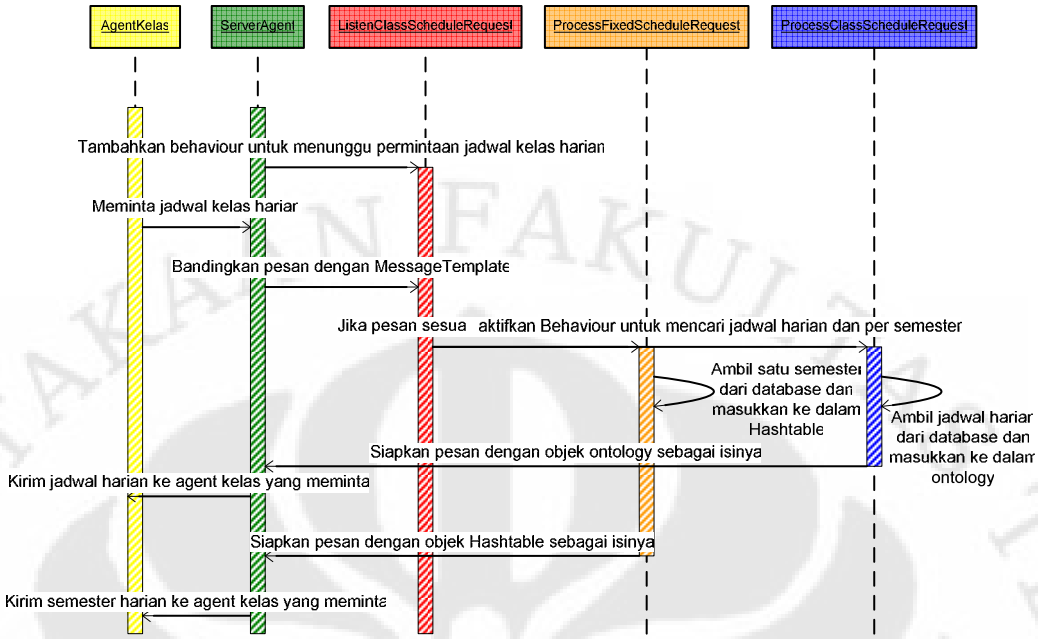
Melayani permintaan daftar jadwal kuliah harian dari Agent Kelas	`ListenClassScheduleRequest`
	`ProcessClassScheduleRequest`
	`ProcessFixedScheduleRequest`
		`CyclicBehaviour`
		`OneShotBehaviour`
		`OneShotBehaviour`

Tabel 4.1. Daftar *behaviour* untuk tanggung jawab utama *Agent Server* (lanjutan)

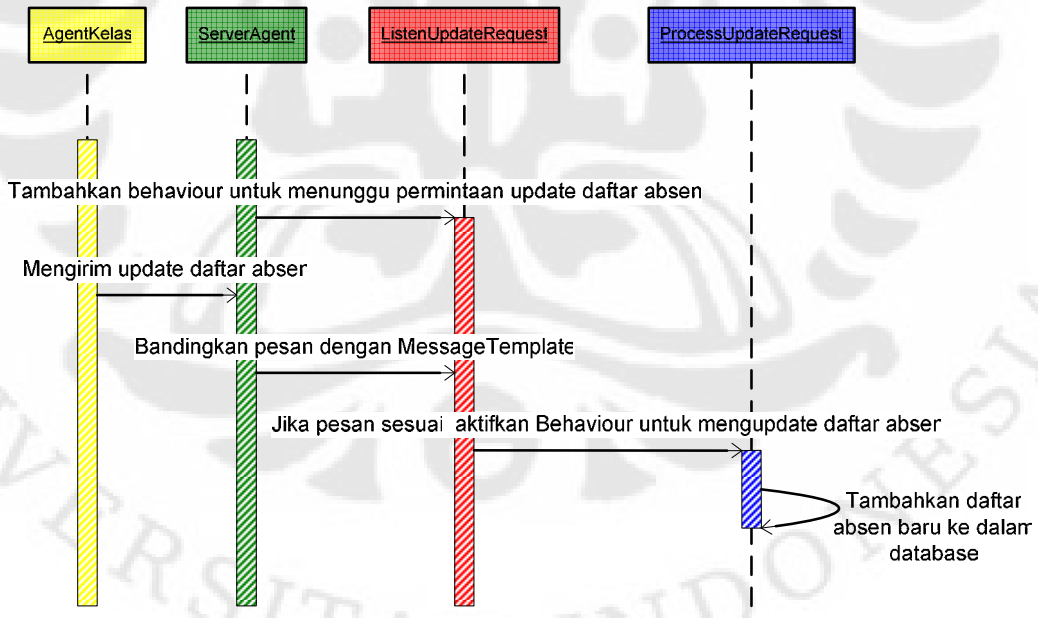
Tanggung jawab	Behaviour Class	Jenis Behaviour
Mencatat penambahan daftar absen dari Agent Kelas ke dalam database	`ListenUpdateRequest` `ProcessUpdateRequest`	`CyclicBehaviour` `OneShotBehaviour`
Mencatat pembatalan jadwal kuliah dari Agent Kelas ke dalam database	`ListenClassCancellation` `ProcessClassCancellation`	`CyclicBehaviour` `OneShotBehaviour`
Mencatat perubahan dan penambahan jadwal kuliah dari Agent Kelas ke dalam database	`ListenChangeScheduleRequest` `ProcessChangeScheduleRequest`	`CyclicBehaviour` `OneShotBehaviour`
Mencari Agent Kelas yang aktif melalui AMS	`AMSSubscriber`	`CyclicBehaviour`

Behaviour untuk mencari *Agent Kelas* yang aktif akan dijalankan segera setelah *Agent Server* dijalankan, sedangkan semua `CyclicBehaviour` lainnya akan dijalankan setelah *Agent Server* berhasil menghubungi *server database*.

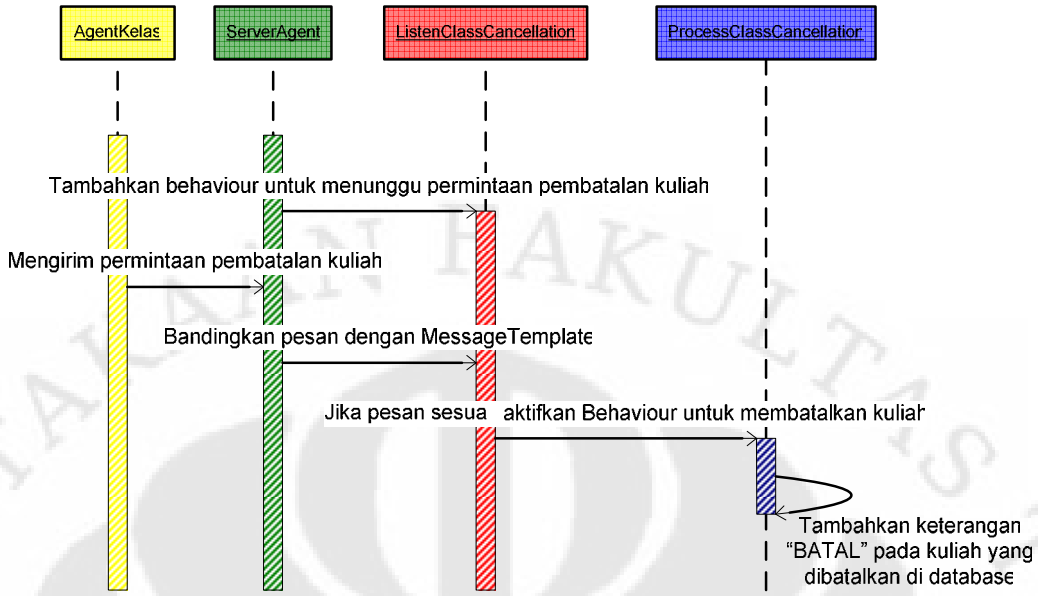
Semua `CyclicBehaviour` selain `AMSSubscriber` digunakan untuk menerima pesan permintaan dari *Agent Kelas*. Jika pesan yang sesuai dengan *message template* diterima oleh salah satu dari `CyclicBehaviour`, maka `CyclicBehaviour` akan menjalankan sebuah `OneShotBehaviour` yang sesuai dengan tugasnya untuk memproses pesan yang diterima. Diagram *sequence* untuk empat tanggung jawab pertama pada Tabel 4.1 ditunjukkan pada Gambar 4.3, 4.4, 4.5, dan 4.6.



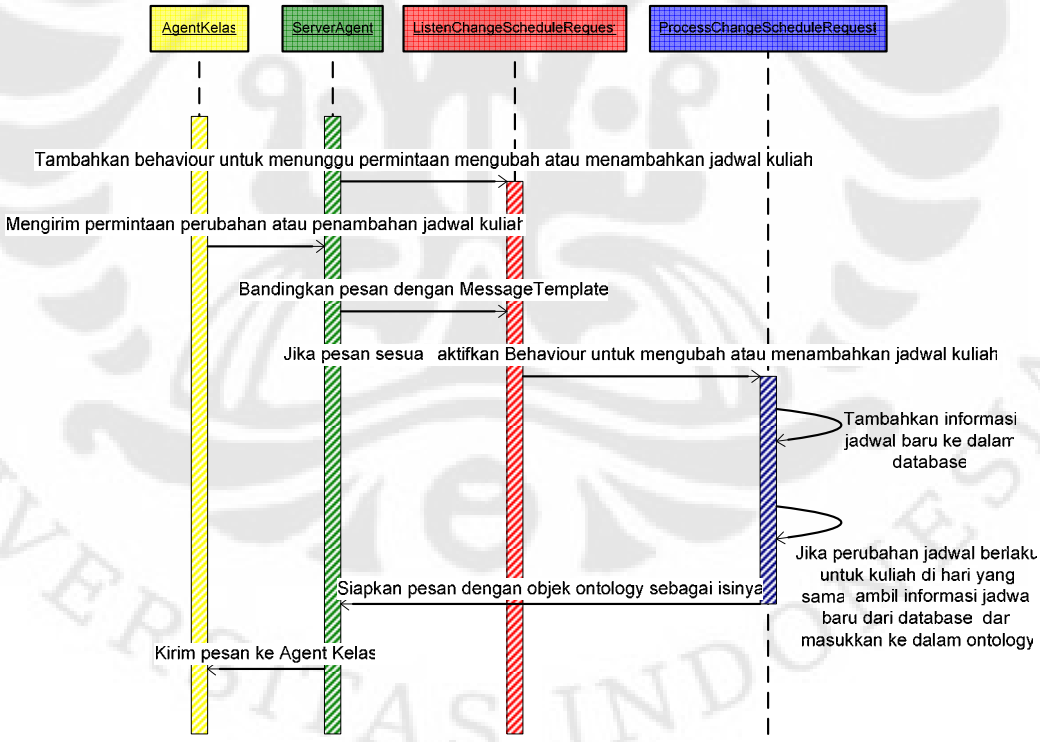
Gambar 4.3. Diagram *sequence* untuk *behaviour* melayani permintaan jadwal kuliah harian pada *Agent Server*



Gambar 4.4. Diagram *sequence* untuk *behaviour* mencatat penambahan daftar absen pada *Agent Server*



Gambar 4.5. Diagram *sequence* untuk *behaviour* mencatat pembatalan kelas



Gambar 4.6. Diagram sequence untuk mencatat perubahan dan penambahan jadwal kuliah pada *Agent Server*

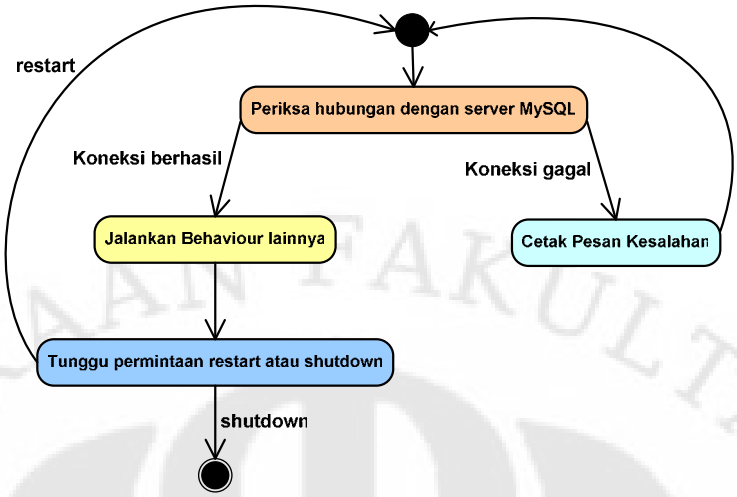
4.2.2 Behaviour untuk Menjamin Hubungan dengan Database

Selain memiliki tanggung jawab untuk melayani permintaan pengambilan informasi dari *database*, *Agent Server* juga memiliki tanggung jawab untuk menjamin adanya hubungan dengan *server database*, dan memberi peringatan jika terjadi kegagalan. Untuk mencapai tujuan tersebut, ditambahkan sebuah `FSMBehaviour` yang bertujuan untuk membangun ulang hubungan dengan *server database* jika terjadi masalah pada hubungan yang sedang berlangsung.

`FSMBehaviour` sendiri hanya berfungsi untuk mengatur perpindahan keadaan dari satu *behaviour* ke *behaviour* lainnya, sedangkan untuk menjalankan tugas menjaga hubungan dengan *server database* diperlukan beberapa *child behaviour*, masing-masing dengan tugas yang berbeda. Daftar *Child behaviour* yang digunakan oleh `FSMBehaviour` ditunjukkan pada Tabel 4.2, sedangkan diagram keadaan untuk perpindahan antar *child behaviour* ditunjukkan pada Gambar 4.7.

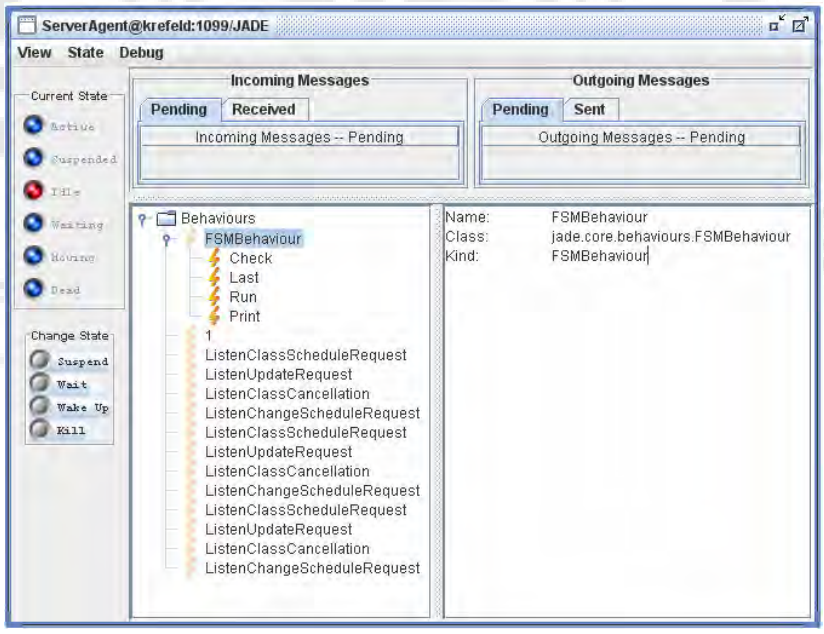
Tabel 4.2. Daftar c*hild behaviour* dari `FSMBehaviour` pada *Server Agent*

Behaviour Class	Jenis Behaviour	Fungsi
`CheckSQLConnection`	`Behaviour`	Menguji hubungan dengan *server database*
`PrintErrorMessage`	`OneShotBehaviour`	Mencetak pesan kesalahan, dan menunggu 10 detik sebelum berpindah ke *behaviour* selanjutnya
`StartBehaviours`	`OneShotBehaviour`	Menjalankan *behaviour* lainnya untuk melakukan tugas utama *Agent Server*
`LastState`	`Behaviour`	Menunggu permintaan untuk *reset* dan *shutdown Agent Server* dari pengguna



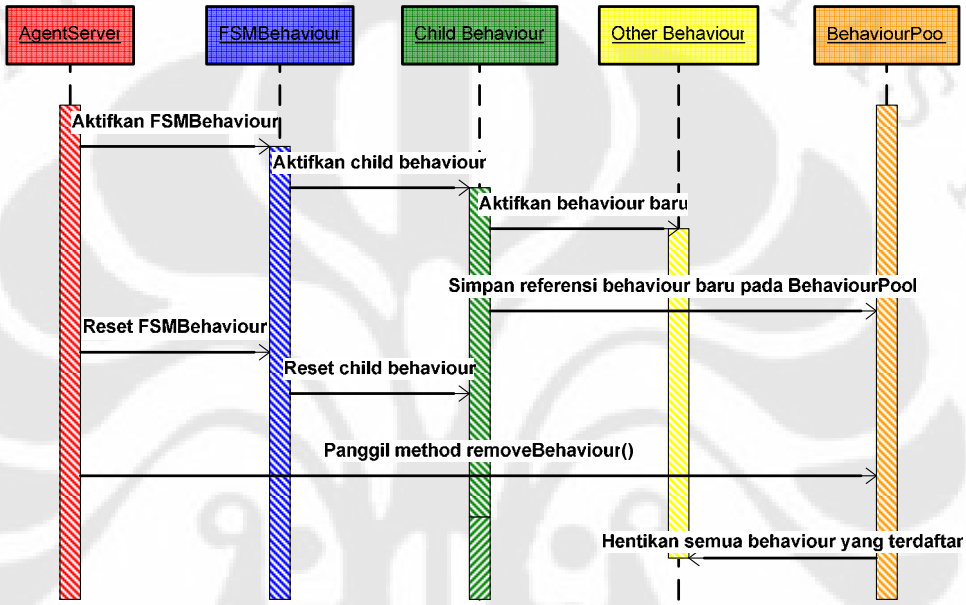
Gambar 4.7. Diagram keadaan untuk `FSMBehaviour` pada *Agent Server*

Pemanggilan fungsi *reset* pada *Agent Server* mengembalikan semua *child behaviour* pada `FSMBehaviour` ke keadaan awal, tetapi tidak menghilangkan *behaviour-behaviour* lain yang dijalankan oleh *child behaviour* sendiri, sedangkan *child behaviour* yang di-*reset* akan kembali menjalankan *behaviour* dengan fungsi yang sama. Adanya dua *behaviour* dengan fungsi yang sama tidak mengganggu cara kerja dari *Agent Server*, tetapi dapat menghabiskan memori komputer. Pembuktian duplikasi behaviour dengan menggunakan *Introspector Agent* ditunjukkan pada Gambar 4.8.



Gambar 4.8. Terjadinya duplikasi behaviour setelah *Agent Server* di-*restart*

Untuk mengatasi masalah tersebut, digunakan sebuah *class* `BehaviourPool` yang berisi beberapa *static method* untuk menyimpan referensi dari semua *behaviour* yang dijalankan, dan menghapus *behaviour* tersebut dari daftar pekerjaan *agent* bila diminta. Diagram *sequence* penggunaan *class* `BehaviourPool` ditunjukkan pada Gambar 4.9.



Gambar 4.9. Diagram *sequence* untuk penggunaan *class* `BehaviourPool`

4.2.3 Fungsi-fungsi Tambahan untuk Administrator

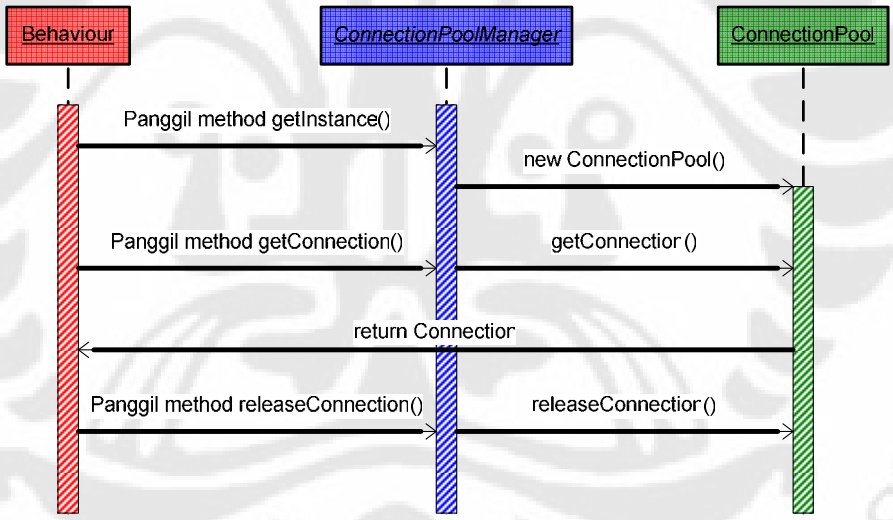
Fungsi-fungsi tambahan administrator sebenarnya tidak diimplementasikan pada *Agent Server* secara langsung, melainkan terdapat pada GUI dari *Agent Server*. Oleh sebab itu pembahasan fungsi-fungsi ini akan dilakukan pada Sub Bab 4.3.

4.2.4 Connection Pooling

Semua tanggung jawab utama dari *Agent Server* berhubungan dengan *server database*. Semua permintaan *Agent* Kelas untuk melakukan *query* dan *update* terhadap *server database* mengharuskan *Agent Server* untuk memiliki objek `Connection`. Pembentukan objek `Connection` baru dapat memakan waktu

yang lama untuk sebuah sistem dengan permintaan akses terhadap *server database* yang tinggi. Oleh sebab itu perlu digunakan mekanisme *connection pooling* untuk menyimpan objek `Connection` yang telah dibuat sebelumnya agar dapat digunakan berulang-ulang tanpa harus membuat objek `Connection` yang baru.

Connection pooling dilakukan dengan menggunakan sebuah class `ConnectionPooling` dengan *method* untuk membuat dan menyimpan, mengambil, dan mengembalikan objek `Connection` dari dan ke dalam *pool*. Namun, proses yang manggunakan objek `Connection` terdapat pada lebih dari satu *class*, sehingga perlu digunakan sebuah *class* dengan *static method* sebagai penghubung antara objek dari *class* `ConnectionPool` dengan class lainnya yang membutuhkan objek `Connection`. Diagram sequence penggunaan connection pooling ditunjukkan pada Gambar 4.10.



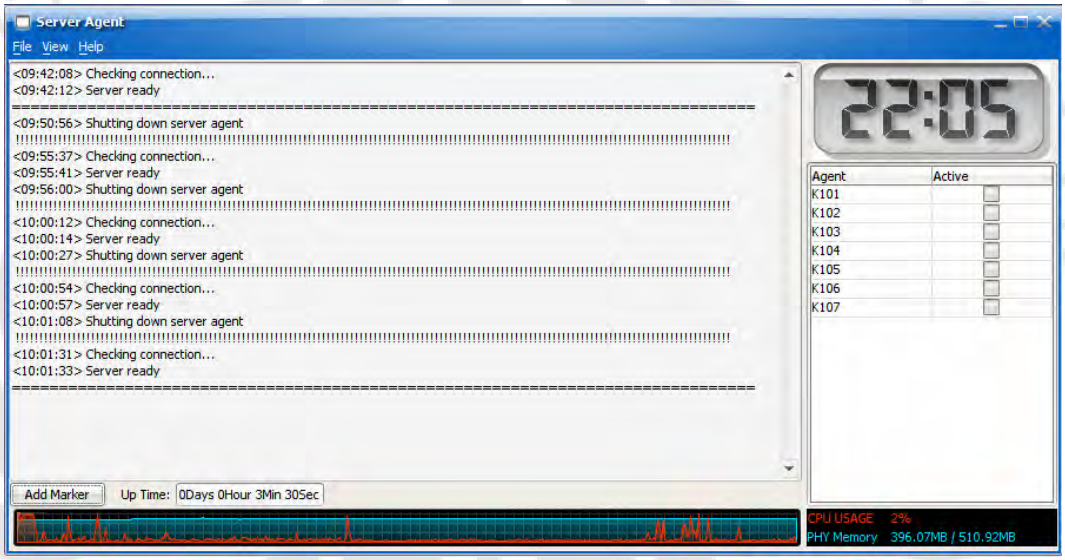
Gambar 4.10. Diagram *sequence* untuk proses pengambilan objek `Connection` dari *pool* pada *Agent Server*

4.3 IMPLEMENTASI GUI

Ketiga *agent* pada Sistem Manajemen Kelas membutuhkan interaksi dengan pengguna. Untuk menjadikan agent *user-friendly*, maka dibutuhkan GUI untuk mempermudah pengguna dalam berinteraksi dengan *agent*.

4.3.1 GUI *Agent* Server

GUI untuk *Agent* Server digunakan untuk memantau keadaan dan keagiatan yang sudah dilakukan oleh *Agent Server* dan mencatatnya ke dalam sebuah *log file* yang diberi nama sesuai dengan tanggal pembuatannya. Melalui GUI, pengguna juga dapat melakukan *reset* dan *shutdown* terhadap *Agent Server*. Tampilan dan daftar komponen GUI dari *Agent* Server ditunjukkan pada Gambar 4.11 dan Tabel 4.3.



Gambar 4.11. Tampilan GUI *Agent Server*

Tabel 4.3. Daftar komponen pada GUI *Agent Server*

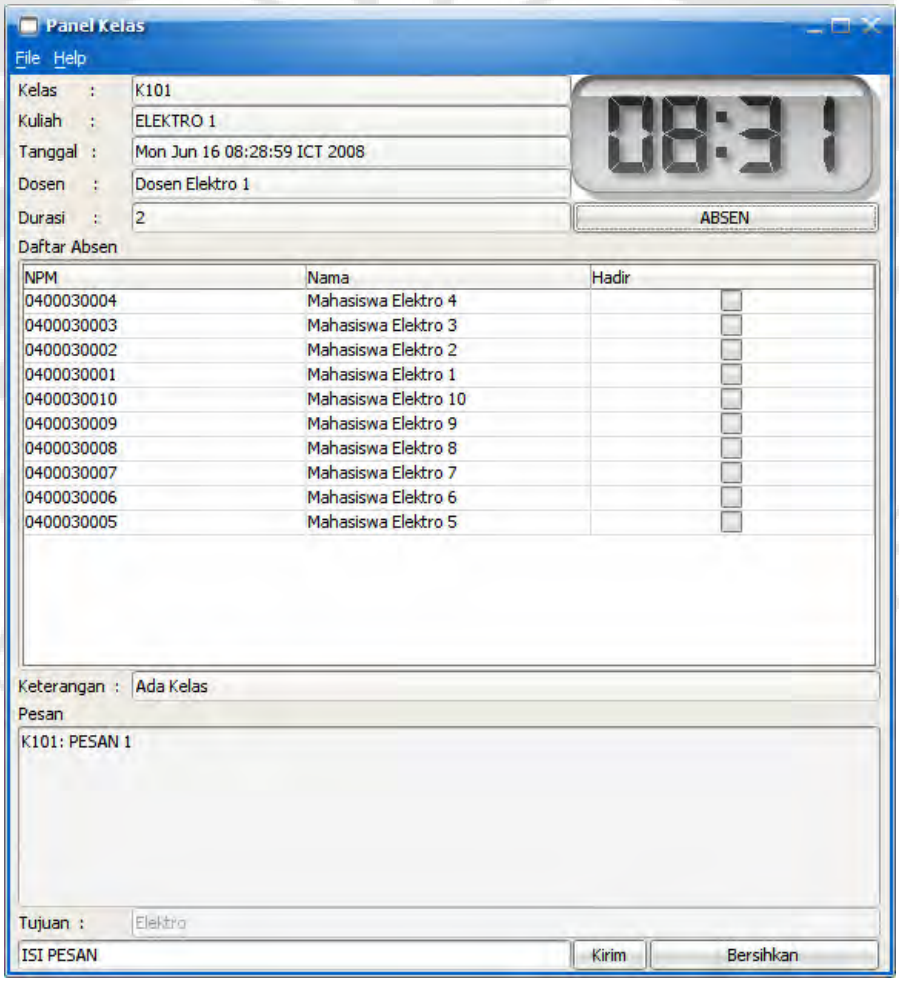
Nama Komponen	Class	Fungsi
Log viewer	`AppendingTextPane`	Menampilkan semua kegiatan yang dilakukan *Agent* Server
Tombol *add marker*	`JButton`	Menambahkan penanda pada *log viewer*
Up time	`JTextArea`	Menampilkan lama *Agent* Server dijalankan
Tabel *Agent* Kelas	`JTable`	Menampilkan daftar *Agent* Kelas yang aktif
Submenu reset	`JMenu`	Melakukan reset terhadap *Agent* Kelas

Tabel 4.3. Daftar komponen pada GUI *Agent Server* (lanjutan)

Nama Komponen	Class	Fungsi
Submenu shutdown	`JMenu`	Mematikan *Agent* Kelas
System Monitor	`SystemMonitor`	Menampilkan grafik pemakaian CPU dan memori fisik
Jam	`LcdClockPanel`	Menampilkan waktu saat ini

4.3.2 GUI *Agent* Kelas

GUI pada *Agent* Kelas berfungsi untuk menampilkan informasi mengenai kegiatan perkuliahan yang sedang berlangsung. Melalui GUI, mahasiswa dan dosen juga dapat mengisi daftar absen, melihat informasi kehadiran peserta kuliah, dan mengirim pesan kepada *Agent* Departemen. Tampilan dan daftar komponen pada GUI *Agent* Kelas ditunjukkan pada Gambar 4.12dan Tabel 4.4.



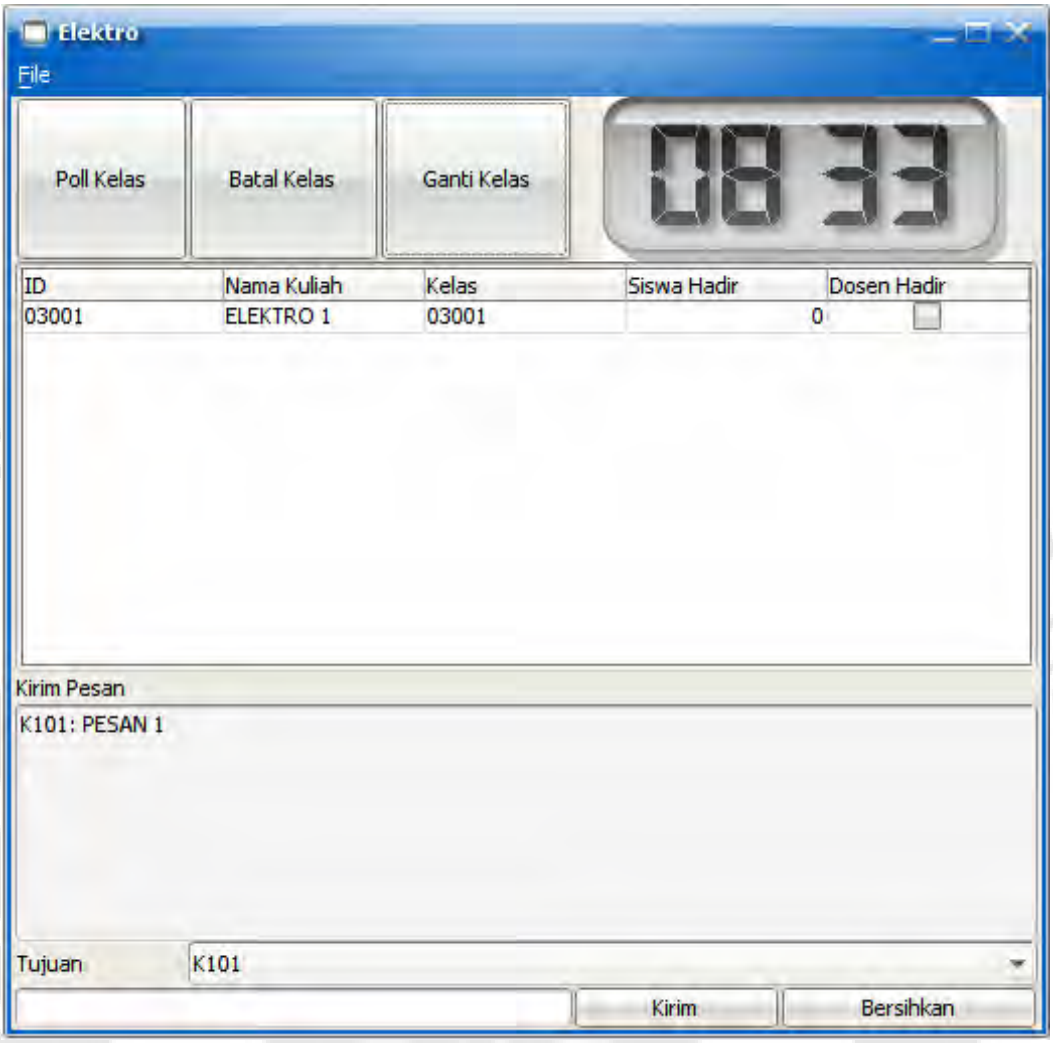
Gambar 4.12. Tampilan GUI *Agent* Kelas

Tabel 4.4. Daftar komponen pada GUI *Agent* Kelas

Nama Komponen	Class	Fungsi
Kelas, kuliah, tanggal, dosen, durasi	`JTextArea`	Menampilkan informasi kuliah yang sedang berlangsung
Tombol absen	`JButton`	Menampilkan jendela untuk mengisi absen
Daftar absen	`JTable`	Menampilkan nama, npm, dan kehadiran peserta kuliah
Keterangan	`JTextArea`	Menampilkan keterangan untuk kuliah yang sedang berlangsung
Daftar pesan	`AppendingTextPane`	Menampilkan pesan yang diterima dan dikirim *Agent* Kelas
Tujuan	`JTextArea`	Menampilkan *Agent* Departemen yang menjadi tujuan pesan
Isi pesan	`JTextArea`	Tempat untuk mengisi pesan yang untuk dikirim ke *Agent* Departemen
Tombol kirim	`JButton`	Mengirim pesan ke *Agent* Departemen
Tombol bersihkan	`JButton`	Membersihkan daftar pesan
Submenu *exit*	`JMenu`	Mematikan *Agent* Kelas
Jam	`LcdClockPanel`	Menampilkan waktu saat ini

4.3.3 GUI *Agent* Departemen

GUI pada *Agent* Departemen berfungsi untuk menampilkan informasi kegiatan kuliah dari *Agent* Kelas yang terdaftar. Melalui GUI ini, dosen atau petugas departemen dapat melakukan pembatalan, perubahan, dan penambahan jadwal kuliah. Tampilan dan komponen yang digunakan pada *Agent* Departemen ditunjukkan pada Gambar 4.13 dan Tabel 4.5.



Gambar 4.13. Tampilan GUI *Agent* Departemen

Tabel 4.5. Daftar komponen pada GUI *Agent* Departemen

Nama Komponen	Class	Fungsi
Tombol *poll* kelas	`JButton`	Melakukan *polling* informasi kuliah terhadap *Agent* Kelas yang terdaftar
Tombol batal kelas	`JButton`	Menampilkan jendela untuk mengisi informasi pembatalan kelas
Tombol ganti kelas	`JButton`	Menampilkan jendela untuk mengisi informasi perubahan dan penambahan jadwal kuliah

Tabel 4.5. Daftar komponen pada GUI *Agent* Departemen (lanjutan)

Nama Komponen	Class	Fungsi
Tabel daftar kuliah	`JTable`	Menampilkan daftar informasi kuliah dari *Agent* Kelas yang terdaftar
Pesan	`AppendingTextPane`	Menampilkan pesan yang diterima dan dikirim *Agent* Depatemen
Tujuan	`JComboBox`	Menampilkan pilihan *Agent* kelas untuk tujuan pesan
Isi pesan	`JTextArea`	Tempat untuk mengisi pesan yang akan dikirim ke *Agent* Kelas
Tombol kirim	`JButton`	Mengirim pesan ke *Agent* Kelas tujuan
Tombol bersihkan	`JButton`	Membersihkan daftar pesan
Submenu *exit*	`JMenu`	Mematikan *Agent* Departemen

BAB V
PENGUJIAN DAN GUI, DAN *AGENT SERVER* PADA SISTEM MANAJEMEN KELAS

5.1 PENGUJIAN GUI

Pengujian terhadap GUI untuk *Agent* Kelas, *Agent* Server, dan *Agent* Departemen dilakukan dengan mencoba semua fitur pada GUI dan mengamati apakah fitur tersebut berfungsi dengan baik. Sebagian besar pengujian fitur melibatkan fungsi dari agent untuk masing-masing GUI, sehingga pengujian terhadap GUI dilakukan setelah fungsi dari masing-masing agent dapat berjalan dengan baik.

Hasil dan persyaratan pengujian terhadap masing-masing GUI ditunjukkan pada Tabel 5.1, 5.2, dan 5.3.

Tabel 5.1. Hasil pengujian GUI *Agent Server*

Kondisi/persyaratan	Terpenuhi?
Dapat menampilan catatan kegiatan yang sudah dilakukan *Agent Server*	Ya
Dapat menambahkan penanda pada catatan kegiatan *Agent Server*	Ya
Dapat menampilkan daftar *Agent* Kelas yang sedang aktif	Ya
Menu *restart* berfungsi dengan benar	Ya
Menu *shutdown* berfungsi dengan benar	Ya
Waktu *uptime* menunjukkan lama *Agent* Server dihidupkan	Ya
System monitor menunjukkan pemakaian memori dan CPU	Ya
Jam menunjukkan waktu dengan benar	Ya

Tabel 5.2. Hasil pengujian GUI *Agent* Kelas

Kondisi/persyaratan	Terpenuhi?
Dapat menampilkan informasi kuliah yang sedang berlangsung	Ya
Dapat menampilkan daftar peserta kuliah beserta status absennya	Ya

Tabel 5.2. Hasil pengujian GUI *Agent* Kelas (lanjutan)

Kondisi/persyaratan	Terpenuhi?
Tombol absen dapat menampilkan jendela untuk mengisi absen	Ya
Dapat memperbaharui status absen mahasiswa pada daftar peserta kuliah	Ya
Dapat menampilkan pesan yang diterima dari *Agent* Departemen	Ya
Dapat mengirimkan pesan kepada *Agent* Departemen dan menampilkannya pada daftar pesan	Ya
Dapat menghapus daftar pesan yang dikirim dan diterima ke dan dari *Agent* Departemen	Ya
Jam menunjukkan waktu dengan benar	Ya
Menu *exit* berfungsi dengan benar	Ya

Tabel 5.3. Hasil pengujian GUI *Agent* Departemen

Kondisi/persyaratan	Terpenuhi?
Dapat menampilkan daftar informasi perkuliahan yang aktif untuk satu departemen	Ya
Dapat memperbaharui status daftar informasi perkuliahan yang sedang aktif	Ya
Tombol pembatalan kuliah dapat memunculkan jendela untuk mengisi informasi pembatalan kuliah	Ya
Tombol pencarian kelas kosong dapat menampilkan jendela untuk mengisi informasi jadwal kuliah yang akan diganti	Ya
Dapat menampilkan pesan yang diterima dari *Agent* Kelas	Ya
Dapat mengirimkan pesan kepada *Agent* Kelas dan menampilkannya pada daftar pesan	Ya
Dapat menghapus daftar pesan yang dikirim dan diterima ke dan dari *Agent* Kelas	Ya
Jam dapat menunjukkan waktu dengan benar	Ya
Menu *exit* berfungsi dengan benar	Ya

5.2 PENGUJIAN DAN ANALISA *AGENT SERVER*

Tahap pengujian dan analisa kinerja *Agent Server* bertujuan untuk mengukur kemampuan *Agent Server* dalam melayani permintaan *Agent* Kelas Pengujian terhadap *Agent Server* terdiri dari 3 jenis pengujian, yaitu:

- pengukuran kinerja *Agent Server* dalam melayani permintaan *Agent* Kelas,
- perbandingan konsumsi memori dari Agent Server sebelum dan setelah digunakannya class `BehaviourPool`,
- pengukuran besar paket yang perlu dikirim kepada *Agent* Kelas.

Semua jenis pengujian terhadap Agent Server dilakukan pada sebuah komputer dengan CPU berkecepatan 1.3 GHz dan RAM sebesar 512 MB yang menggunakan sistem operasi *Microsoft Windows 2000*, Java Runtime Environment versi 6 update 2, dan JADE versi 3.5.

5.2.1 Pengujian dan Analisa Kinerja *Agent Server*

Pengujian yang dilakukan untuk mengukur kinerja *Agent Server* dilakukan terhadap proses melayani permintaan jadwal kuliah harian dari *Agent* Kelas. Proses melayani permintaan jadwal kuliah harian dipilih untuk pengujian sebab proses ini merupakan proses yang paling banyak menggunakan sumber daya pada komputer, sehingga dapat menunjukkan kinerja dari *Agent* Server yang sebenarnya. Proses pengujian sendiri terdiri dari 2 variabel pengujian, yaitu:

- beban permintaan atau banyaknya permintaan jadwal kuliah harian yang dikirim pada saat yang bersamaan, dan
- pemrosesan dengan dan tanpa *connection pooling*.

5.2.1.1 Skenario Pengujian Kinerja Agent Server

Untuk melakukan pengujian digunakan sebuah *agent* penguji yang meniru bentuk pesan yang dikirimkan oleh *Agent* Kelas untuk meminta jadwal kuliah harian. Pada pengujian terdapat 3 jenis beban permintaan jadwal kuliah harian, yaitu 1 permintaan/detik, 10 permintaan/detik, dan 20 permintaan/detik. Selain itu, untuk masing-masing beban permintaan dilakukan 2 jenis pengujian, dimana

pengujian pertama dilakukan dengan menggunakan *connection pooling* dan pengujian kedua dilakukan tanpa menggunakan *connection pooling*.

Penghitungan waktu pemrosesan dilakukan dengan menggunakan memodifikasi kode sumber dari *Agent Server* agar mencatat waktu sebelum dan sesudah pemrosesan jadwal harian, dan menampilkan selisihnya pada layar.

Dari 6 jenis pengujian yang dilakukan, masing-masing dilakukan sebanyak 10 kali dan dilakukan pencatatan waktu yang ditampilkan. Dari 10 sampel yang didapat kemudian dihitung nilai rata-rata dan standar deviasinya.

5.2.1.2 Hasil Pengujian Kinerja Agent Server

Data hasil keempat jenis pengujian kinerja *Agent Server* diperlihatkan pada Tabel 5.4 dan Gambar 5.1. Setiap jenis pengujian dilakukan sebanyak 10 kali, dan dari data yang diperoleh dihitung nilai rata-rata dan standar deviasi.

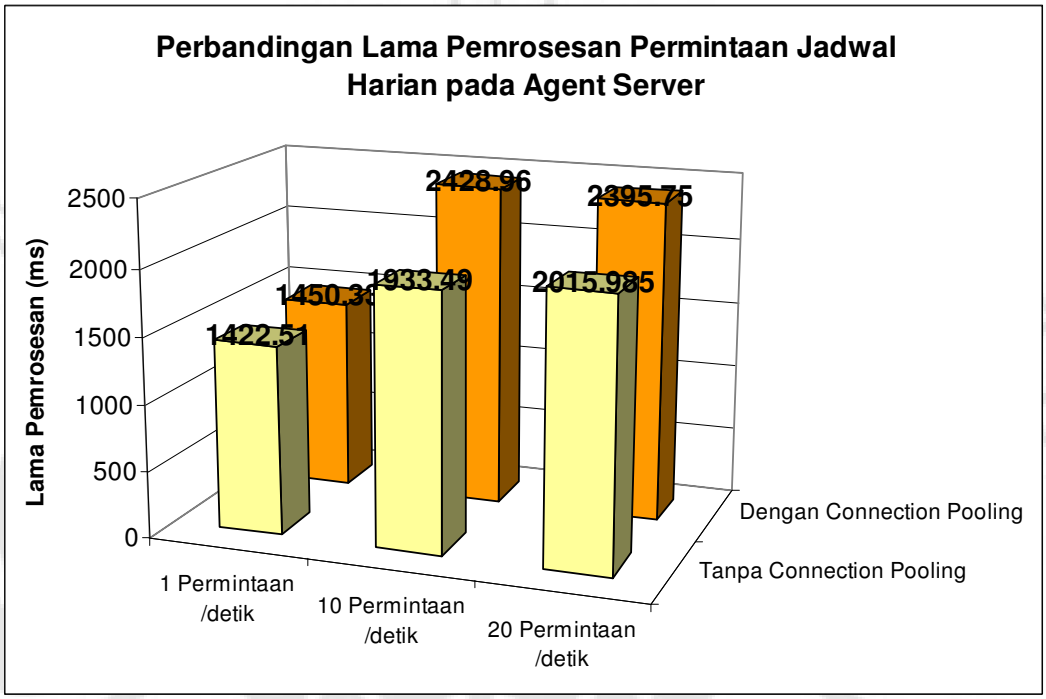
Tabel 5.4. Data hasil pengujian kinerja *Agent Server*

Data ke-	1 Permintaan per detik dengan *Connection Pooling*	1 Permintaan per detik tanpa *Connection Pooling*	10 Permintaan per detik dengan *Connection Pooling*	10 Permintaan per detik tanpa *Connection Pooling*	20 Permintaan per detik dengan *Connection Pooling*	20 Permintaan per detik tanpa *Connection Pooling*
1	1427.3	1454.7	1835.43	2292	1920.4	2315.55
2	1428	1437.6	1796.48	2230.3	2091.6	2380.55
3	1433.4	1453.1	1806.90	2240.8	2045.4	2385.9
4	1428.8	1453.2	1938.06	2543.7	2060.9	2518.85
5	1425.7	1448.5	1846.45	2448.4	2039.05	2564.15
6	1403	1442.1	2072.39	2554.6	2081.5	2253.15
7	1415.4	1464	1822.35	2549.8	1909.35	2234.35
8	1430.3	1453.2	2096.31	2549	2089.85	2352.45
9	1433.4	1451.6	1885.77	2146.8	1842.1	2375.9
10	1399.8	1445.3	2234.85	2734.2	2079.7	2576.65
Rata-rata	1422.51	1450.33	1933.49	2428.96	2015.985	2395.75
Standar Deviasi	12.457	7.001	142.9	182.758	85.839	114.739

5.2.1.3 Analisa Hasil Pengujian Kinerja Agent Server

Berdasarkan data yang diperlihatkan pada Tabel 5.4, waktu yang dibutuhkan untuk memproses permintaan jadwal kuliah harian pada beban 1 permintaan/detik tidak tidak memiliki perbedaan yang berarti antara pengujian dengan menggunakan *connection pooling* dan tanpa menggunakan *connection*

pooling. Penggunaan *connection pooling* hanya meningkatkan kecepatan pemrosesan sebesar 1.918% dari 1450.33 ms jika tidak menggunakan *connection pooling* menjadi 1422.51 ms jika menggunakan *connection pooling*.



Gambar 5.1. Grafik hasil pengujian kinerja *Agent Server*

Sebaliknya pada beban 10 permintaan/detik, penggunaan *connection pooling* dapat mempercepat waktu pemrosesan sebesar 20.398% dari 2428.96 ms pada pengujian tanpa *connection pooling* menjadi 1933.4 ms pada pengujian dengan *connection pooling*. Namun efisiensi penggunaan *connection pooling* pada beban 20 permintaan/detik mengalami penurunan menjadi 15.852% dari waktu pemrosesan selama 2395.75 ms pada pengujian tanpa *connection pooling* menjadi 2015.985 ms pada pengujian dengan *connection pooling*.

Dari hasil perbandingan lama pemrosesan di atas dapat diketahui bahwa penggunaan *connection pooling* pada beban 1 permintaan/detik tidak memberikan perbedaan yang berarti antara pengujian dengan dan tanpa menggunakan *connection pooling*. Manfaat penggunaan *connection pooling* baru terlihat pada beban 10 permintaan/detik, dimana lama pemrosesan dengan menggunakan *connection pooling* 20.398% lebih cepat bila dibandingkan dengan pengujian

tanpa *connection pooling*. Namun pada beban 20 permintaan/detik, efisiensi penggunaan *connection pooling* tidak mengalami perubahan yang berarti, bahkan hasil pengujian menunjukkan bahwa pengurangan lama pemrosesan menurun menjadi 15.852%. Dari hasil pengujian ini dapat diambil kesimpulan bahwa manfaat penggunaan *connection pooling* semakin dapat dirasakan bila beban permintaan semakin meningkat. Namun efisiensi dari penggunaan *connection pooling* tidak terus meningkat seiring dengan meningkatnya beban permintaan. Untuk memperjelas perubahan efisiensi penggunaan *connection pooling* terhadap beban permintaan, dilakukan pengujian kedua untuk beban permintaan sebesar 2 dan 8 permintaan/detik. Hasil pengujian kedua ditunjukan pada Tabel 5.5.

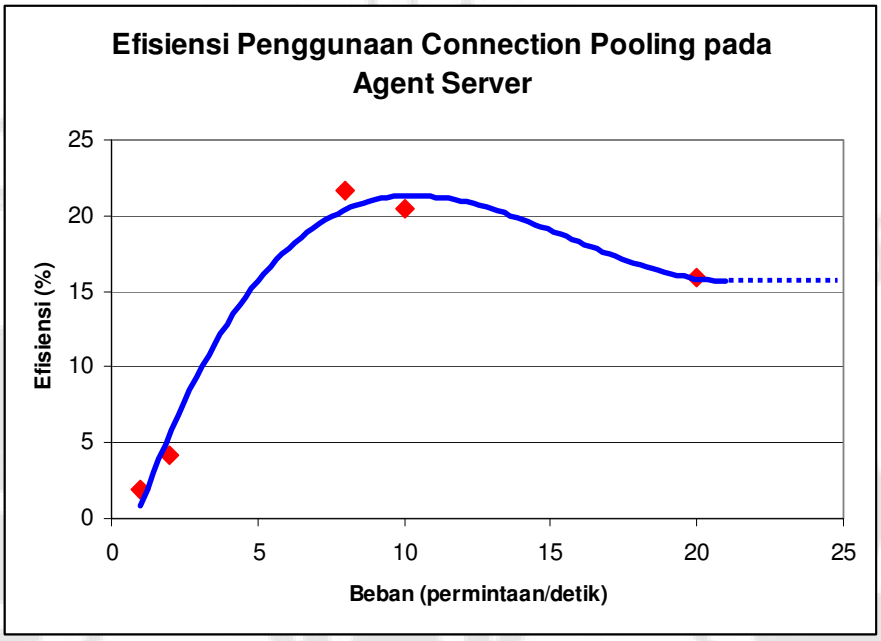
Tabel 5.5. Data hasil pengujian kinerja *Agent Server* yang kedua

Pengujian ke-	2 Permintaan/detik		8 Permintaan/detik	
	Dengan connection pooling	Tanpa connection pooling	Dengan connection pooling	Tanpa connection pooling
1	1438	1516	1500	1484
2	1422	1453	1515	3031
3	1406	1484	1469	1531
4	1437	1547	3063	3156
5	1469	1485	1469	1547
6	1485	1640	3031	3110
7	1438	1516	1515	1500
8	1453	1484	1516	3078
9	1453	1453	1406	1532
10	1469	1516	1532	3031
Rata-rata	1447	1509.4	1801.6	2300

Berdasarkan hasil pengujian kedua, pada beban 2 permintaan/detik terjadi peningkatan kecepatan pemrosesan sebesar 4.13% dari waktu pemrosesan selama 1509.6 ms pada pengujian tanpa *connection pooling* menjadi 1469 ms pada pengujian dengan *connection pooling*. Pada beban 8 permintaan/detik peningkatan kecepatan pemrosesan bertambah sebesar 21.67% dari 2300 ms pada pengujian tanpa *connection pooling* menjadi 1532 ms pada pengujian dengan *connection pooling*.

Grafik perubahan efisiensi penggunaan *connection pooling* pada *Agent Server* terhadap jumlah permintaan setiap detiknya ditunjukkan pada Gambar 5.2.

Pada grafik tersebut dapat dilihat bahwa efisiensi *connection pooling* mulai menurun pada beban permintaan di atas 10 permintaan/detik, dan mulai menuju kestabilan di nilai 16%.



Gambar 5.2. Grafik efisiensi penggunaan *connection pooling* pada *Agent Server*

5.2.2 Pengujian Efisiensi *class* `BehaviourPool`

Pengujian efisiensi dari penggunaan *class* `BehaviourPool` yang digunakan untuk menghentikan *behaviour* yang dijalankan oleh *behaviour* lain saat *Agent Server* di-*restart* bertujuan untuk mengukur sebesar apa manfaat *class* ini terhadap beban pemakaian memori yang ditanggung komputer.

5.2.2.1 Skenario Pengujian Efisiensi class BehaviourPool

Pengujian dilakukan dengan mengamati besar pemakaian memori komputer secara keseluruhan saat menjalankan *Agent Server*. *Agent Server* kemudian di-*restart* sebanyak 10 kali untuk mendapatkan data perubahan jumlah pemakaian memori. Pengamatan pemakaian memori dilakukan dengan menggunakan *Task Manager* yang disediakan *Microsoft Windows 2000*.

Pengujian ini dilakukan pada dua jenis *Agent Server* yaitu Agent Server yang sudah menggunakan *class* `BehaviourPool` dan tanpa menggunakan *class*

`BehaviourPool` masing-masing sebanyak 3 kali. Dari 3 sampel yang diperoleh kemudian dicari nilai rata-ratanya untuk masing-masing nilai pemakaian memori setelah restart ke-x.

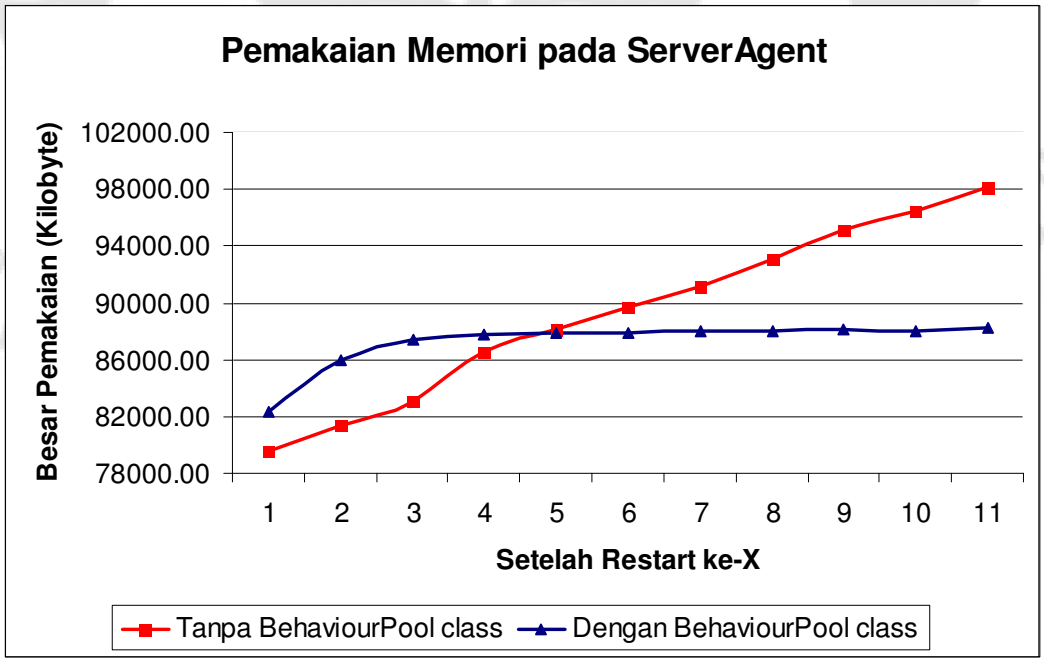
5.2.2.2 Analisa Hasil Pengujian Efisiensi class BehaviourPool

Data hasil pengujian penggunaan memori *Agent Server* ditunjukkan pada Tabel 5.6, dan grafik rata-rata pemakaian memori ditunjukkan pada Gambar 5.3.

Tabel 5.6. Data hasil pengujian efisiensi *class* `BehaviourPool`

Dengan class BehaviourPool				Tanpa class BehaviourPool			
1	2	3	Rata-rata	1	2	3	Rata-rata
80628	81447	85000	**82358.33**	78112	80056	80616	**79594.67**
85852	84203	87860	**85971.67**	80032	81616	82564	**81404**
86972	86051	89136	**87386.33**	81748	83140	84428	**83105.33**
86712	86831	89756	**87766.33**	85276	86600	87884	**86586.67**
86444	86835	90220	**87833**	86920	88340	89288	**88182.67**
86644	86995	89960	**87866.33**	88540	89836	90676	**89684**
86688	87227	90068	**87994.33**	90288	91024	92176	**91162.67**
86876	87239	89880	**87998.33**	92212	93080	93768	**93020**
87040	87287	90212	**88179.67**	94532	95048	95732	**95104**
87108	87011	89920	**88013**	95400	96984	97136	**96506.67**
87148	87495	89932	**88191.67**	97276	98124	98860	**98086.67**

(satuan dalam Kilobyte)



Gambar 5.3. Diagram rata-rata pemakaian memori sebelum dan sesudah penggunaan *class* `BehaviourPool`

Dari data hasil pengujian yang ditunjukkan pada Gambar 5.3 dapat diambil kesimpulan bahwa tanpa penggunaan *class* `BehaviourPool`, jumlah penggunaan memori akan terus bertambah setiap kali *Agent Server* di-*restart*. Dengan penggunaan `BehaviourPool`, semua *behaviour* yang terdaftar akan dihentikan saat *Agent Server* di-*restart*, sehingga jumlah pemakaian memori komputer tidak bertambah.

5.2.3 Pengujian Aliran Data yang Dihasilkan Agent Server

Pengujian besar aliran data yang dihasilkan *Agent Server* bertujuan untuk mengetahui beban yang dihasilkan *Agent Server* terhadap jaringan yang nantinya akan digunakan sebagai medium komunikasi dengan *Agent* Kelas.

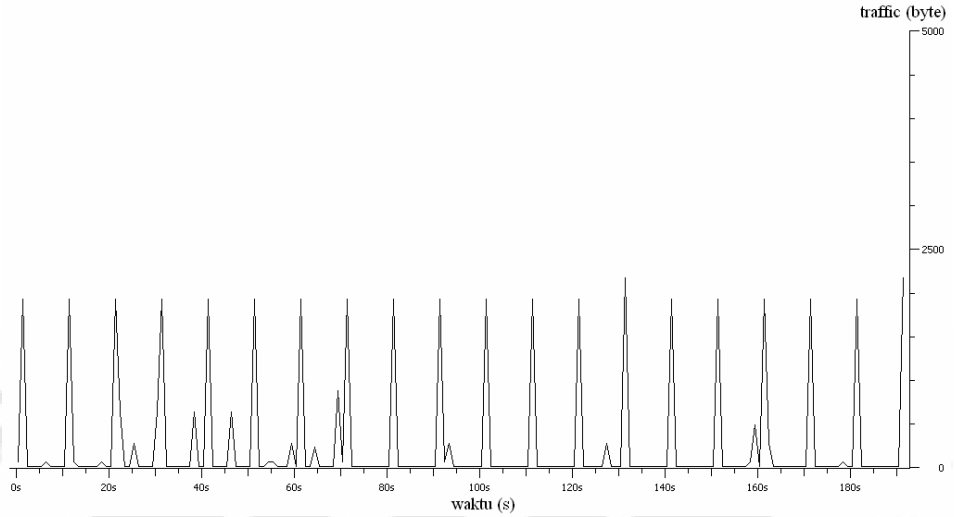
5.2.3.1 Skenario Pengujian Aliran Data yang Dihasilkan Agent Server

Pengujian dilakukan dengan menggunakan sebuah agent penguji yang berfungsi untuk mengirim pesan permintaan jadwal kuliah harian. *Agent* penguji ditempatkan pada komputer yang berbeda dengan *Agent Server*, tetapi masih berada dalam satu LAN (*Local Area Network*). *Agent* penguji tersebut diatur agar mengirim pesan permintaan jadwal kuliah harian setiap 10 detik, sehingga *Agent Server* akan membalas permintaan tersebut dengan mengirimkan jadwal kuliah harian setiap 10 detik.

Paket-paket yang dikirimkan kemudian ditangkap dengan menggunakan *Wireshark*. Besarnya aliran data yang dihasilkan *Agent Server* dapat dilihat dengan menggunakan fitur *I/O Graph* pada *Wireshrak*.

5.2.3.2 Analisa Hasil Pengujian Aliran Data yang Dihasilkan Agent Server

Grafik besar pengiriman data jadwal kuliah harian terhadap waktu yang diukur dengan menggunakan *I/O Graph* pada *Wireshark* ditunjukkan pada Gambar 5.4.



Gambar 5.4. Hasil pengukuran aliran data yang dihasilkan *Agent Server* saat mengirim jadwal kuliah harian dengan mnggunakan *I/O Graph*

Dari grafik yang dihasilkan dapat dilihat bahwa besarnya aliran data yang dihasilkan *Agent Server* untuk mengirim jadwal kuliah harian memiliki nilai yang relatif tetap, yaitu sekitar 2000 *bytes*. Besar aliran data yang dihasilkan ini termasuk kecil bila dibandingkan dengan kecepatan aliran data pada jaringan LAN yang mencapai 100 Mbps, sehingga pengimplementasian Sistem Manajemen Kelas pada beberapa komputer yang terhubung melalui LAN tidak akan membebani jaringan atau mengganggu transmisi data yang lain.

BAB VI
KESIMPULAN

Setelah melakukan pengembangan dan pengujian terhadap aplikasi Sistem Manajemen Kelas didapat beberapa kesimpulan sebagai berikut:

1. Efisiensi penggunaan connection pooling dalam memproses permintaan jadwal kuliah harian memiliki nilai efisiensi saturasi sebesar 16% dengan titik optimal pada beban sebesar 8 permintaan/detik dengan efisiensi tertinggi sebesar 21.67%.
2. Penggunaan *class* `BehaviourPool` untuk menghentikan *behaviour* saat Agent Server di-*restart* dapat menekan bertambahnya penggunaan memori dengan menghentikan semua *behaviour* yang terdaftar saat *Agent Server* di-*restart*.
3. Komunikasi antar *agent* pada Sistem Manajemen Kelas hanya memerlukan sumber daya jaringan yang minimal, sehingga pengaplikasiannya tidak banyak membebani jaringan.
4. Pendekatan *agent* dapat mempermudah pengembangan sebuah perangkat lunak karena dapat memecah permasalahan menjadi bagian-bagian yang dikerjakan oleh *agent* yang berbeda.
5. Penggunaan JADE sangat mempermudah proses pengembangan aplikasi berbasis *agent* dengan tersedianya berbagai alat bantu tambahan beserta API yang disediakan oleh bahasa pemrograman *Java*.

DAFTAR ACUAN

[1] Clark S. Lindsey, (2004). *History of Java*. Diakses 5 Juni 2008. http://www.particle.kth.se/~lindsey/JavaCourse/Book/Part1/Java/Chapter01/history.html

[2] James Gosling, Henry McGilton (1996). *The Java Language Environment*. Diakses 5 Juni 2008. http://java.sun.com/docs/white/langenv/Intro.doc2.html#349

[3] (2008). *About the Java Technology*. Diakses 5 Juni 2008. http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html

[4] Hyacinth S. Nwana, (2008). *Software Agents: An Overview*. Diakses pada 6 Juni 2008. http://agents.umbc.edu/introduction/ao/

[5] Bellifemine, Fabio *et.al*., *Developing Multi Agent System with JADE* (John Wiley &Sons, Ltd, 2007) hal 3-4

[6] Fabio Bellifemine, Giovanni Caire, dan Tiziana Trucco, *JADE Administrator's Guide,* diakses 5 Juni 2008. http://jade.tilab.com/doc/administratorsguide.pdf

[7] (2008). *MySQL 5.1 Reference Manual*. Diakses pada 9 Juni 2008. http://dev.mysql.com/doc/refman/5.1/en/introduction.html

[8] Delisle, Marc (2006). *Creating Your MySQL Database: Practical Design Tips and Techniques* (Packt Publishing, 2006) hal 10

[9] Nikraz, Magid *et.al., A Methodology for the Analysis and Design of Multi-Agent System Using JADE* (Telecom Italia Lab, 2006) hal 13-14

DAFTAR PUSTAKA

Byous, Jojn (2003), *Java Technology: The Early Years*, (http://java.sun.com/features/1998/05/birthday.html), diakses 5 Juni 2008

Caire, Giovanni, Developing multi-agent applications with JADE. Tutorial for beginners, (http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf), diakses 5 Juni 2008

Delisle, Marc (2006). *Creating Your MySQL Database: Practical Design Tips and Techniques*, Birmingham, Packt Publishing, 2006

Fabio Bellifemine, Giavanni Caire, dan Dominic Greenwood, *Developing Multi Agent System with JADE* , New Jersey, John Wiley &Sons, Ltd, 2007

Fabio Bellifemine, Giovanni Caire, dan Tiziana Trucco, *JADE Administrator's Guide,* (http://jade.tilab.com/doc/administratorsguide.pdf), diakses 5 Juni 2008

Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, dan Giovanni Rimassa, *JADE Programmer's Guide*, (http://jade.tilab.com/doc/programmersguide.pdf), diakses 5 Juni 2008

Feizabadi, Shahrooz (1996), *History of Java,* (http://ei.cs.vt.edu/~wwwbtb/book/chap1/java_hist.html), Diakses 5 Juni 2008

Grimshaw, David, *JADE Administrative Tutorial*, (http://jade.tilab.com/doc/tutorials/JADEAdmin/index.html), diakses 5 Juni 2008

Jean Vaucher dan Ambroise Ncho, *JADE Tutorial and Primer*, (http://www.iro.umontreal.ca/~vaucher/Agents/Jade/JadePrimer.html), diakses 5 Juni 2008

Jovanovic, Jelena, *Java Agent Development Framework*, 2007

Luke Welling dan Laura Thomson, *MySQL Tutorial*. Indianapolis, Indiana, Sams Publihing, 2004

H.M. Deitel, *Java How to Program, Sixth Edition*, New Jersey, Pearson Education, 2005

Mark Matthews, Jim Cole, Joseph D. Gradecki, *MySQL and Java Developer's Guide*, Indianapolis, Indiana, Wiley Publishing, 2003

Magid Nikraz, Giovanni Caire ,dan Parisa A. Bahri., *A Methodology for the Analysis and Design of Multi-Agent System Using JADE*,Telecom Italia Lab, 2006

Rusty Harold, Elliotte (1997), *The Prehistory of Java*, (http://www.cafeaulait.org/slides/hope/02.html), diakses 5 Juni 2008

Topley, Kim, *Core SWING Advanced Programming*, New Jersey, Prentice Hall, 1999