PERANCANGAN DAN PEMBUATAN SISTEM TERDISTRIBUSI MULTI-TIER BERBASIS ENTERPRISE JAVA BEAN UNTUK APLIKASI PEMBAYARAN ONLINE

TUGAS AKHIR

OLEH :

NOVA IKAWARDHANA
NRP : 5196 100 025

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2002

PERANCANGAN DAN PEMBUATAN SISTEM TERDISTRIBUSI MULTI-TIER BERBASIS ENTERPRISE JAVA BEAN UNTUK APLIKASI PEMBAYARAN ONLINE

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer
Pada
Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya

Mengetahui / Menyetujui,

Dosen Pembimbing I	Dosen Pembimbing II
Dr. Ir. Djoko Lianto Buliali	**Dwi Sunaryono S.Kom**
NIP. 131 996 151	NIP. 132 163 671

SURABAYA
PEBRUARI, 2002

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ ﴿١﴾

ٱلْحَمْدُ لِلَّهِ رَبِّ ٱلْعَٰلَمِينَ ﴿٢﴾

ٱلرَّحْمَٰنِ ٱلرَّحِيمِ ﴿٣﴾

مَٰلِكِ يَوْمِ ٱلدِّينِ ﴿٤﴾

إِيَّاكَ نَعْبُدُ وَإِيَّاكَ نَسْتَعِينُ ﴿٥﴾

ٱهْدِنَا ٱلصِّرَٰطَ ٱلْمُسْتَقِيمَ ﴿٦﴾

صِرَٰطَ ٱلَّذِينَ أَنْعَمْتَ عَلَيْهِمْ غَيْرِ ٱلْمَغْضُوبِ عَلَيْهِمْ
وَلَا ٱلضَّآلِّينَ ﴿٧﴾

Abstrak

Sejalan dengan semakin kompleksnya proses yang terjadi pada suatu sistem maka semakin dituntut pula untuk menyederhanakan sistem tersebut dengan memecahnya menjadi sistem-sistem yang lebih kecil. Hal ini identik dengan pengubahan arsitektur dari client-server menjadi multi-tier dimana terjadi pemecahan suatu servis menjadi beberapa servis yang masing-masing memiliki fungsi tertentu.

Pada arsitektur multi-tier dipisahkan antara lapisan bisnis yang terletak pada server aplikasi dengan lapisan user yang memungkinkan adanya penyebaran obyek-obyek dari lapisan bisnis ke lapisan user. Lapisan bisnis terpisah secara fisik dengan lapisan data sehingga lapisan user dapat mengakses data melalui obyek-obyek yang disediakan oleh lapisan bisnis berdasarkan aturan-aturan tertentu.

Dalam tugas akhir ini dibuat prototype aplikasi pembayaran online, yang merupakan interaksi antara bank yang menyediakan obyek bisnis berbasis Enterprise Java Bean yang diakses oleh ecommerce sarana pelayanan pembayaran online konsumennya dalam lingkungan multi-tier.

KATA PENGANTAR

Segala puji dan syukur semata ditujukan kehadirat ALLAH SWT yang telah memberikan rahmat dan hidayah-Nya kepada seluruh umat sehingga penulis dapat menyelesaikan tugas akhir ini yang berjudul "**Perancangan dan Pembuatan Sistem Terdistribusi Multi-tier Berbasis Enterprise Java Bean untuk Aplikasi Pembayaran Online**".

Mata kuliah tugas akhir memuat beban 4 satuan kredit sebagai salah satu syarat menyelesaikan program strata satu (S-1) pada jurusan Teknik Informatika di Institut Teknologi Sepuluh Nopember Surabaya.

Dalam menyelesaikan tugas akhir ini, penulis menggunakan seluruh ilmu yang penulis peroleh selama duduk dibangku kuliah dengan tidak terlepas dari bantuan, bimbingan dan dukungan dari berbagai pihak.

Sebagai hasil karya seorang manusia, penulis menyadari bahwa tugas akhir ini jauh dari kesempurnaan bahkan mengandung banyak kekurangan sehingga dengan kerendahan hati, insya Allah penulis menerima segala saran dan kritik membangun dari pembaca nan budiman.

Surabaya, Januari 2002

Penulis

UCAPAN TERIMA KASIH

Dengan mengucap syukur Alhamdulillah kepada ALLAH SWT, penulis menyampaikan rasa penghormatan yang setingginya serta rasa ucapan terima kasih kepada pihak yang telah memberi bantuan baik secara langsung dan tidak langsung, secara moril dan material kepada

1. Ayahanda dan Ibunda tercinta yang telah membesarkan, mengasuh, dan mendidik penulis dengan mencurahkan segala doa, bimbingan, pengorbanan, keikhlasan, dan kasih sayang nan tulus.
2. Bapak Dr. Ir. Djoko Lianto dan Bapak Dwi Sunaryono S.Kom sebagai dosen pembimbing tugas akhir kami atas segala bimbingan dan arahannya.
3. Bapak Agus Zainal Arifin S.Kom M.Kom sebagai Ketua Jurusan Teknik Informatika ITS.
4. Bapak Ir. M. Husni sebagai dosen wali selama penulis duduk dibangku kuliah.
5. Bapak dan Ibu dosen Jurusan Teknik Informatika yang telah memberikan ilmu yang bermanfaat kepada kami.
6. Seluruh staf dan karyawan Jurusan Teknik Informatika yang memberi kemudahan sarana perkuliahan.
7. Seluruh rekan-rekan TA-mania dan admin serta warga lab-komputing, lab-sisfo, dan lab-RPL atas segala bantuan dan kebersamaannya.
8. Seluruh rekan-rekan c0c atas kebersamaan kita selama ini dan masa mendatang.

9. Abas, Agung, Ajie, Andik, Andrie, Ariv, Cahyono, Cak Yul, Danar, Julius, Ocal, xHenry, Aditya, Bayu, dan seluruh rekan-rekan warga Keputih ID/48 Surabaya atas dukungannya.
10. Rekan-rekan yang tidak dapat penulis sebutkan satu-persatu yang telah memberikan kontribusi dalam pengerjaan tugas akhir ini.

Tiada untaian kata nan terindah yang dapat penulis sampaikan sebagai balasan atas jasa yang telah penulis terima, melainkan hanyalah doa dan harapan semoga ALLAH SWT membalas semua amal tersebut, Jazakumullah Khairan Katsiran.

Daftar Isi

Daftar Gambar

Daftar Tabel

BAB I

PENDAHULUAN

1.1 LATAR BELAKANG

Seiring dengan semakin kompleksnya sistem komputerisasi berarsitektur jaringan maka dibutuhkan suatu arsitektur sistem yang memiliki kemampuan untuk menyebarkan sumber dayanya melintasi jaringan komputer.

Pemisahan suatu aplikasi menjadi beberapa aplikasi berdasarkan servis-servisnya mampu menggantikan kekurangan-kekurangan pada arsitektur klien-server *2-tier*. Arsitektur yang dimaksud adalah *multi-tier* dimana terjadi pemisahan antara lapisan user, logika proses bisnis, dan tempat penyimpanan data yang keseluruhannya saling terintegrasi.

Arsitektur *multi-tier* dapat diimplementasikan dengan menggunakan teknologi *Enterprise Java Bean*, yang memungkinkan pembuatan aplikasi berskala *enterprise* dengan waktu yang relatif cepat dan mudah karena tergolong teknologi komponen perangkat lunak sisi server yang dapat dibangun dengan cepat (*Rapid Application Development*). EJB memungkinkan pembuatan komponen dengan menitikberatkan hanya pada proses bisnisnya saja tanpa harus melakukan pemrograman kode-kode jaringan dan algoritma transaksi.

Pembayaran online memiliki karakteristik yang dapat diimplementasikan dengan menggunakan teknologi EJB. Bank memiliki sistem yang menyediakan sumber daya berupa obyek-obyek sebagai bentuk implementasi aplikasi

pembayaran online yang bersifat baku dan dapat diakses oleh vedor penyedia layanan ecommerce yang terpisah secara fisik oleh jaringan dan dapat dijalankan pada berbagai *platform* sistem operasi apapun.

1.2 PERMASALAHAN

Permasalahan yang dihadapi adalah bagaimana membuat suatu sistem yang mampu menyediakan obyek-obyek terdistribusi dalam jaringan dari sistem yang telah baku yaitu bank dengan kemampuan agar dapat diakses oleh sistem yang berkomunikasi dengannya. Selain itu dibutuhkan pembuatan suatu aplikasi kompleks dengan relatif cepat dan mudah, yang hanya menitikberatkan pada pembuatan perangkat lunak yang mengimplementasikan proses bisnis dengan kemampuan penanganan transaksi. Dalam membangun aplikasi klien diharapkan tidak harus digunakan komputer yang sama dengan komputer dimana obyek disebar dan tetap mempertimbangkan kesamaan proses bisnis selanjutnya.

1.3 TUJUAN

Tujuan dari tugas akhir ini adalah untuk membangun suatu sistem yang dengan kemampuan:

1. Menyebarkan obyek-obyek enterprise bean dalam lingkungan *multi-tier*.
2. Dapat melakukan transaksi yang dapat menjamin ACID properties (*Atomicity, Consistency, Isolation*, dan *Durability*).

Dalam mendukung tujuan diatas dapat dimanfaatkan untuk mengimplementasikan studi kasus transaksi pembayaran on-line.

1.4 PEMBATASAN PERMASALAHAN

Fokus tugas akhir ini terletak pada pembuatan prototype aplikasi komponen berbasis EJB yang menyediakan sumber daya obyek terdistribusi untuk diakses oleh aplikasi ecommerce berbasis Java Server Pages (JSP) yang terpisahkan oleh jaringan komputer yang berada didalam suatu jaringan lokal (*intranet*), dengan memperhatikan otentikasi dan otorisasi user terhadap obyek serta proses transaksi yang mampu ditangani oleh teknologi EJB. Selain itu dijelaskan mengenai pembuatan rancang bangun sistem pembayaran online dalam lingkungan *multi-tier* dengan pendekatan *Unified Modelling Language* (UML). Aplikasi yang dibangun merupakan prototype dari simulasi sistem pembayaran *online*.

1.5 METODOLOGI

Metode penelitian yang akan dilakukan dalam tugas akhir ini terdiri dari tahap-tahap sebagai berikut

1. Studi literatur.

   Pada tahap ini, dipelajari konsep obyek terdistribusi dalam lingkungan *multi-tier*, teknologi EJB yang digunakan dalam pengimplementasian sistem, dan sistem pembayaran online.

2. Perancangan arsitektur komponen.

   Pada tahap ini dibangun suatu rancang bangun yang mengimplementasikan contoh kasus pembayaran online berdasarkan konsep lingkungan *multi-tier* dengan kemampuan menyediakan obyek yang terdistribusi dengan pendekatan UML.

3. Pembuatan komponen.

Berdasarkan hasil dari tahapan sebelumnya, dibuat suatu aplikasi perangkat lunak yang mampu mendistribusikan obyek dalam lingkungan *multi-tier*.

4. Test dan evaluasi.

   Pada tahap ini, program yang telah dibuat diuji berdasarkan tujuan yang hendak dicapai seperti yang tertulis diatas.

5. Penyusunan buku tugas akhir.

   Pada tahap ini, disusun suatu dokumentasi laporan dari pelaksanaan tugas akhir yang dikerjakan.

1.6 SISTEMATIKA PEMBAHASAN

Pembahasan dalam Tugas Akhir ini dibagi menjadi beberapa bab sebagai berikut

- Bab I, Pendahuluan, berisi latar belakang, permasalahan, tujuan, pembatasan per masalahan, metodologi dan sistematika pembahasan.
- Bab II, Teori Penunjang, pada bagian ini dibahas mengenai konsep dan implementasi yang mendukung pembahasan tugas akhir ini, seperti obyek terdistribusi dalam lingkungan *multi-tier*, *Enterprise Java Bean* dengan teknologinya, transaksi, dan *Unified Modelling Language* (UML).
- Bab III, Perancangan Sistem, yaitu pembuatan rancang bangun studi kasus pembayaran online berdasarkan pendekatan UML, dibahas pula tentang peracangan basis data, servis bisnis, dan servis user.
- Bab IV, Pembuatan Prototipe Sistem, pada bagian ini diimplementasikan skrip basis data yang dihasilkan kedalam basis data, servis bisnis, servis user, dan arsitektur 3-*tier* dalam aplikasi pembayaran online yang melibatkan bank

dengan menggunakan komponen EJB dan ecommerce yang diimplementasikan menggunakan Java Server Pages (JSP) dalam lingkungan *multi-tier* yang disebarkan oleh suatu server aplikasi.

- Bab V, Uji Coba dan Evaluasi, pada bagian ini dilakukan uji coba pada implentasi sistem yang dibuat yaitu pada aplikasinya termasuk pada distribusi obyek dan transaksi, berdasarkan parameter-parameter nilai yang diberikan kemudian dilakukan evaluasi terhadap hasil uji coba tersebut.
- Bab VI, Penutup, berisi kesimpulan yang diambil dari Tugas Akhir ini beserta saran untuk pengembangan selanjutnya.

BAB II

TEORI PENUNJANG

2.1 Sistem Terdistribusi Dalam Lingkungan Arsitektur *Multi-Tier*

2.1.1 Pengenalan Sistem Terdistribusi

Pada sistem terdistribusi terdapat sekumpulan komputer yang dihubungkan melalui suatu jaringan dan perangkat lainnya yang memungkinkan komputer-komputer tersebut untuk mengkoordinasikan aktivitas dan mampu membagi sumber daya seperti perangkat keras, lunak lunak, data, atau obyek-obyek kepada sistem[1]. Sumber daya dalam sistem terdistribusi tersebut dilindungi oleh suatu komputer sehingga komputer yang lainnya hanya dapat mengaksesnya melalui komunikasi. Sumber daya tersebut diatur oleh pengatur sumber daya yang merupakan bagian terpenting dalam sistem terdistribusi. Dalam mengakses sumber daya tersebut klien berkomunikasi dengan pengatur sumber daya supaya dapat mengakses sumber daya yang telah disediakan sistem. Sistem terdistribusi dapat diimplementasikan menjadi 2 model yaitu klien/server dan berbasis obyek.

Dalam model klien/server, seluruh sumber daya yang dibagi ditangani dan diatur oleh proses server. Suatu proses klien dapat berkomunikasi dengan proses server untuk bertukar informasi. Penetapan suatu proses sebagai proses server dan proses lainnya sebagai klien ini masing-masing tergantung pada tugasnya masing-masing.

---

[1] [Mahmoud 99] halaman 7

Sistem terdistribusi berbasis obyek merupakan sekumpulan obyek yang terjadi isolasi antara peminta layanan (klien) dari penyedia layanan (server) melalui suatu antar muka. Dengan kata lain bahwa klien terisolasi dari implementasi servis yang merupakan representasi dari data dan kode-kode program yang dieksekusi. Klien mengirim pesan ke suatu obyek yang menginterprestasikannya untuk memutuskan servis yang harus dilaksanakan. Dalam sistem berorientasi obyek terjadi pengisolasian antara klien dan server melalui suatu antar muka.

2.1.2 Kebutuhan Arsitektur Komponen Sisi Server

Sistem terdistribusi berorientasi obyek dapat diimplementasikan dengan menggunakan teknologi komponen yang merupakan suatu perangkat lunak yang terdiri dari kode-kode program yang mengimplementasikan kumpulan antar muka[2]. Komponen bukanlah merupakan suatu keseluruhan aplikasi yang dapat berjalan sendiri tetapi suatu bagian dari suatu aplikasi perangkat lunak yang lebih kompleks.

Pada pemrograman berorientasi obyek terjadi pemisahan antara antar muka komponen dengan implementasinya. Antar muka komponen mendefinisikan kerangka untuk kode-kode program yang memanggilnya. Sedangkan implementasi komponen merupakan logika pemrograman komponen yang mengimplementasikan antar muka-nya. Dengan pemisahan ini, pengkodean dan

---

[2] [Roman 99] halaman 3

algoritma proses bisnis komponen dapat diubah tanpa harus mengubah dan mengkompilasi ulang kode-kode program kliennya.

Java merupakan bahasa pemrograman berorientasi obyek yang ideal dalam pembuatan komponen karena mendukung pemisahan antara antar muka dan implementasinya. Salah satu komponen berbasis *Java* adalah *Java Bean.* Komponen ini merupakan suatu arsitektur komponen berbasis *Java* yang dipaketkan dalam suatu komponen sehingga membentuk suatu komponen dengan kompleksitas yang lebih besar. Tetapi *Java Bean* tidak dapat disebar (*deployed*) dalam suatu server aplikasi (*non-deployable component*). Untuk memenuhi kebutuhan arsitektur komponen yang tersebar maka *Java* membuat teknologi komponen lain yang disebut *enterprise beans. Enterprise Java Bean* dapat disebar melalui suatu kontainer yang berupa server aplikasi baik secara individual maupun disatukan dengan komponen lainnya menjadi suatu sistem aplikasi yang lebih besar.

Enterprise Java Bean (EJB) dapat dipanggil oleh aplikasi berbasis *Java* maupun oleh komponen EJB lainnya. Salah satunya adalah *Java Servlet* dan *Java Server Pages* (JSP). Servlet merupakan komponen berbasis jaringan yang menggunakan fungsionalitas dari dari server web yang berorientasi *response/request* dari host klien sedangkan JSP merupakan skrip yang kemudian di kompile menjadi servlet[3].

---

[3] [Roman99] halaman 37-38

2.1.3 Arsitektur *Multi-Tier*

Penyebaran komponen mendukung konkurensi user dalam melakukan operasi yang simultan, aman, reliabel, dan efisien. Contoh dari penyebaran komponen sisi server yang terdistribusi adalah

- **Bank**, dimana terdapat banyak mesin ATM yang terhubung ke server pusat bank.
- **Web site,** dimana ribuan bahkan jutaan user terhubung ke server web yang terhubung juga dengan suatu server pusat dalam mengakses data dan logika tertentu.

Banyak masalah yang akan timbul, seperti skalabilitas, pemeliharaan, keamanan, reliabilitas, dan lainnya dalam menyebarkan komponen sisi server. Banyak klien yang mengakses obyek-obyek komponen sehingga adanya ketergantungan yang utama pada kondisi dan kinerja server tersebut. Sehingga dalam mennyebarkan aplikasi sisi server dilakukan melalui berbagai pengujian agar dapat berjalan dengan baik pada suatu lingkungan yang kompleks.

Pemisahan lapisan didasarkan pada tanggung jawab dan fungsional tiap-tiap lapisan, biasanya terjadi pemisahan seperti sebagaimana berikut

- **Lapisan Presentasi** memuat komponen yang berurusan dengan antar muka dan interaksi dengan pengguna. Lapisan ini dapat berupa suatu aplikasi yang berdiri sendiri dan ditulis dengan suatu bahasa pemrograman seperti *Java Servlet*, *Java Servlet Pages (JSP)*, dan/atau *Java Applet*.

- **Lapisan Logika Bisinis** memuat komponen yang bekerja dalam mengimplementasikan logika dari proses bisnis. Komponen ini biasanya ditulis dalam suatu tipe bahasa seperti *Java* atau *C++*.
- **Lapisan Data** digunakan oleh lapisan logika untuk menyimpan data secara permanen. Keutamaan dari lapisan ini adalah penggunaan basis data sebagai tempat menyimpan data.

Fungsi pemisahan suatu aplikasi kedalam beberapa lapisan-lapisan logika adalah mengisolasi tiap-tiap lapisan dari yang lainnya. Dalam arsitektur *2-tier* terjadi dua pemisahan sistem secara fisik antara srver dan kliennya sedangkan arsitektur *3-tier* atau *n-tier* memisahkan tiap-tiap lapisan logika secara fisik menjadi tiga atau lebih lapisan *(tier)* yang terdistribusi. Tiap lapisan tersebut dipisahkan dari yang lainnya oleh beberapa batas fisik, seperti mesin, atau jaringan.

2.1.2.1 Arsitektur 2-*tier*

Sebelumnya, hampir keseluruhan metode penyebaran yang digunakan berbasis arsitektur *2-tier* yang mengkombinasikan lapisan logika bisnis dengan salah satu dari dua layer lainnya. Ada dua kombinasi yang mungkin yaitu mengkombinasikan lapisan presentasi dengan lapisan logika bisnis atau menyatukan beberapa logika bisnis kedalam lapisan data.

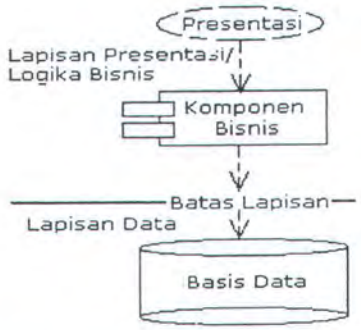
Arsitektur 2-*tier* memiliki karakteristik sebagai berikut

- **Biaya penyebaran yang tinggi**. Dalam instalasi aplikasi, *driver* basis data harus diinstal dan dikonfigurasi pada masing-masing lapisan klien yang jumlahnya bisa mencapai ratusan bahkan ribuan mesin. Sebagai contoh

apabila koneksi yang digunakan melalui jembatan *Open Database Connection* (ODBC) maka driver basis data yang digunakan harus dikenali oleh ODBC tersebut dalam membentuk koneksi dengan basis data yang letaknya mungkin dibatasi secara fisik oleh jaringan.

- **Biaya perubahan *driver* basis data yang tinggi**. Apabila setelah beberapa saat pemakaian diputuskan untuk mengganti *driver* basis data dengan lainnya (misalnya untuk versi yang lebih baru) dibutuhkan instalasi ulang tiap-tiap mesin basis data klien. Hal tersebut membutuhkan biaya yang relatif tinggi bila dilihat dari segi pemeliharaannya karena pada lapisan klien dapat terdiri dari ratusan atau bahkan ribuan mesin.
- **Biaya yang tinggi dalam perubahan skema basis data**. Pada klien/server terjadi pengaksesan secara langsung terhadap basis data melalui JDBC, SQL/J, atau ODBC, berarti bahwa klien-klien berhubungan secara langsung dengan skema basis data. Jika suatu saat diputuskan untuk mengubah skema untuk menangani bisnis proses yang baru maka dibutuhkan untuk membangun dan menyebarkan kembali tiap-tiap klien.
- **Biaya yang tinggi pada perubahan tipe basis data.** Jika diputuskan untuk mengubah tipe basis data, misalnya dari basis data relasional menjadi berbasis obyek maka yang dibutuhkan tidak hanya menyebarkan aplikasi kembali untuk masing-masing klien tetapi secara keseluruhan mengubah kode-kode klien supaya dapat menangani tipe basis data yang baru.
- **Biaya yang tinggi pada perubahan logika bisnis**. Pengubahan business logic layer melibatkan kompilasi dan penyebaran kembali tiap-tiap klien.

- **Biaya koneksi basis data yang tinggi**. Tiap-tiap klien basis data membutuhkan koneksi dalam mengakses basis data. Untuk memenuhi koneksi tersebut ada keterbatasan sehingga ketika klien sedang/sudah tidak menggunakan basis data tersebut, koneksi seringkali masih tertahan dan tidak dapat digunakan oleh yang lain.
- **Kinerja jaringan yang buruk**. Ketika dibutuhkan hubungan dengan basis data, dibutuhkan sejumlah perjalanan dalam jaringan yang memisahkan lapisan logika bisnis dan lapisan data. Rintangan ini dapat menghambat total waktu dari suatu operasi basis data dan dapat menyebabkan tersumbatnya jaringan sehingga mengurangi besarnya *bandwith* dalam jaringan.



Gambar 2.2 Mengkombinasikan presentation dengan business logic layer dalam arsitektur 2-*tier*.

2.1.2.1.1 Kombinasi antara lapisan presentasi dengan lapisan logika bisnis

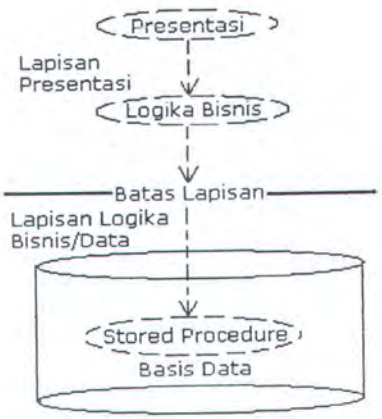
Lapisan presentasi dan lapisan logika bisnis dirangkai secara bersama kedalam suatu lapisan tunggal dengan cara memasukkan lapisan akses data kedalam suatu lapisan tersendiri. Keadaan ini diilustrasikan dengan adanya batas lapisan antara lapisan logika bisnis dengan data sebagaimana yang diperlihatkan pada gambar 2.2. Jika lapisan pertama dijadikan sebagai klien dan kedua sebagai

server, arsitektur ini secara efektif menjadikan klien membesar dan server mengecil.

Dalam arsitektur *2-tier*, aplikasi klien berkomunikasi dengan lapisan data melalui sebuah "jembatan" penghubung basis data API seperti Open Basis data conectivity (ODBC) atau Java Basis data Conectivity (JDBC).

2.1.2.1.2 Menyatukan lapisan logika bisnis kedalam lapisan data

Penyebaran dapat dilakukan dengan mengkombinasikan bagian-bagian lapisan logika bisnis dengan lapisan data menjadi lapisan yang terpisah, seperti yang diperlihatkan gambar 2.3.



Gambar 2.3 Menyatukan lapisan data kedalam lapisan kedua dalam arsitektur 2-*tier*.

Dalam arsitektur ini, lapisan pertama dijadikan sebagai "klien" dan lapisan kedua sebagai "server", hal ini menjadikan klien mengecil sedangkan server membesar. Pada skenario arsitektur ini beberapa logika bisnis dimasukkan kedalam basis data. Basis data memungkinkan untuk mengeksekusi logika dalam konteks basis data dengan menulis sejumlah modul-modul yang disebut *stored procedure*. Dengan memasukkan logika-logika kedalam *stored procedure*, dapat

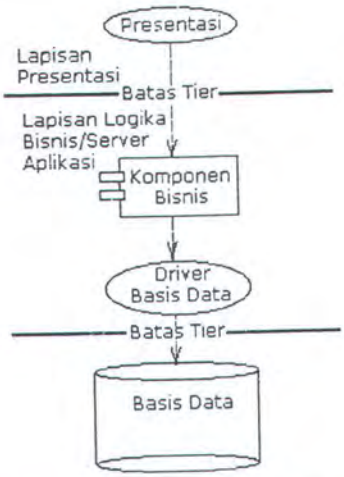
diperoleh sejumlah peningkatan kemampuan dan kinerja. Juga dengan memasukkan beberapa logika kedalam basis data, kesibukan lalu lintas dalam jaringan dari logika ke basis data dapat diminimalisasi. Daripada melakukan *n* query basis data, lebih baik dengan memanggil satu prosedur yang disimpan dalam basis data, dimana pada prosedur melakukan *n* query. Hal ini akan meningkatkan kecepatan operasi basis data dan mengurangi total kepadatan lalu lintas sehingga memungkinkan klien yang lain untuk melakukan operasi dalam jaringan menjadi lebih cepat. Dengan logika yang spesifik untuk tabel dalam basis data itu sendiri, akan dapat meningkatkan kinerja. Beberapa permasalahan yang dihadapi oleh arsitektur 2-tier masih dapat terjadi. Pembuatan *stored procedure*-pun tidak dapat memecahkan permasalahan yang ada. Misalnya, bahasa pemrograman pada *stored procedure* seringkali khusus digunakan pada sistem basis data tertentu. Keuntungan yang diperoleh yaitu dengan menggunakan penghubung basis data seperti ODBC atau JDBC yang memungkinkan beberapa basis data untuk terhubung.

2.1.2.2 Arsitektur n-tier

Arsitektur n-tier menambahkan satu atau lebih lapisan kedalam model arsitektur 2-tier. Dalam mendistribusikan n-tier, lapisan presentasi, lapisan logika bisnis, dan lapisan data dipisahkan kedalam suatu lapisan fisik tertentu. Contoh arsitektur n-tier adalah arsitektur 3-tier berbasis web seperti yang diilustrasikan pada gambar 2.4. Terjadi pemisahan seperti berikut

- **Lapisan Presentasi** menggunakan satu atau lebih server web, dimana bisa berupa *Java Servlet, Java Server Pages, Active Server Pages*, dan lainnya.

- **Lapisan logika Bisnis** menggunakan ruang alamat satu atau lebih server aplikasi. Server aplikasi menyediakan suatu lingkungan yang digunakan untuk memuat dan menjalankan (*runtime*) komponen logika bisnis.
- **Lapisan Data** terdiri dari satu atau lebih basis data dan biasanya berisi suatu logika relasional dengan data dalam bentuk *stored procedure*.



Gambar 2.4 Arsitektur n-tier

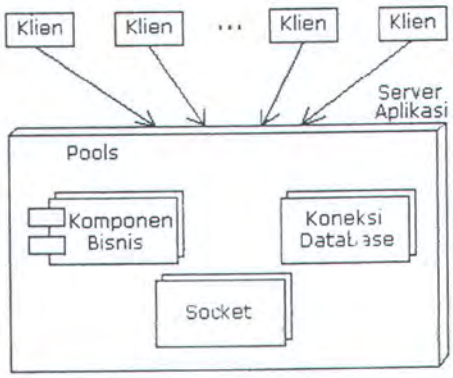
Sebagai contoh n-tier adalah aplikasi perangkat lunak pada bank yang memiliki banyak lapisan yang masing-masing mewakili suatu departemen bank. Arsitektur n-tier memiliki karakteristik sebagai berikut

- **Biaya penyebaran yang murah**. Driver basis data lebih memungkinkan untuk diinstal dan dikonfigurasi di sisi server daripada pada klien. Akan lebih murah untuk menyebar dan mengkonfigurasi perangkat lunak dalam suatu lingkungan sisi server yang terkontrol daripada menyebarkan perangkat lunak pada ribuan laptop ataupun terminal klien.
- **Biaya perubahan basis data yang rendah**. Klien tidak lagi mengakses basis data secara langsung, tetapi melalui suatu lapisan *middle* dalam

mengakses data. Hal ini memungkinkan untuk memindahkan skema basis data, mengubah ke tipe driver basis data yang berbeda, atau bahkan mengubah tipe tempat penyimpanan tanpa perlu untuk menyebarkan kembali masing-masing kliennya.

- **Biaya pengubahan logika bisnis yang rendah**. Pengubahan logika bisnis dapat dilakukan tanpa adanya pengompilasian dan pendistribusian ulang masing-masing kliennya.
- **Dapat mengamankan bagian-bagian yang disebar dengan *firewall*.** Banyak proses bisnis yang sensitif terhadap proteksi data. Sebagai contoh dalam penyebaran berbasis web, terdapat suatu skenario proses bisnis yang tidak memungkinkan untuk memberikan akses lapisan bisnis secara langsung kepada user yang berada diluar, tetapi kebutuhan sistem mengharuskan pemberian akses lapisan presentasi pada dunia luar. Pemecahannya adalah meletakkan *firewall* antara lapisan presentasi dan bisnis.
- **Sumber daya dapat di-*pooling* secara efisien untuk *reused*.** Pada arsitektur n-tier, koneksi ke sumber daya eksternal dapat di atur secara efisien. Pooling sumber daya dilakukan berdasarkan kenyataan bahwa klien sering kali melakukan aktifitas yang lain ketika sedang menggunakan suatu sumber daya. Daripada seringkali memperoleh dan melepaskannya kembali koneksi pada sumber daya seperti basis data maka dapat dilakukan *pooling* terhadap obyek dalam memenuhi permintaan klien yang lain. Jumlah koneksi ke basis data yang dilakukan jauh dibawah dari jumlah total obyek

dari sistem yang disebar. Berdasarkan kenyataan bahwa koneksi basis data sangat mahal maka dibutuhkan cara untuk meningkatkan kinerja sistem. Terlebih lagi, hubungan ke sumber daya tidak perlu untuk dilakukan secara terus menerus untuk meningkatkan kinerja aplikasi. *Pooling* sumber daya dapat diterapkan pada sumberdaya yang lainnya dengan baik, seperti pada koneksi *socket* dan *thread*. Kenyataannya, pada sebuah arsitektur n-tier, komponen bisnis dapat dikumpulkan dan digunakan kembali oleh banyak klien. *Pooling* komponen berarti bahwa tidak dibutuhkan suatu obyek khusus untuk tiap-tiap klien, sebagaimana seperti yang digunakan pada 2-tier klien, seperti yang diilustrasikan pada gambar 2.6.



Gambar 2.6 Pooling sumber daya dalam sebuah pendistribusian n-tier.

- **Melokalisir kemunduran kinerja**. Jika suatu lapisan kelebihan beban maka lapisan lainnya masih dapat berfungsi dengan baik, pengguna masih mungkin untuk mengakses halaman web meskipun server aplikasi sedang kelebihan beban (*overburdened*).
- **Melokalisir error (kekeliruan)**. Jika suatu kritikal error terjadi maka dapat dilokalisir hanya pada suatu lapisan saja sedangkan lapisan lainnya masih

dapat berfungsi dengan baik. Ketika server aplikasi rusak, server web dapat melaporkannya kepada klien melalui browser web.

- **Kinerja komunikasi menurun**. Adanya pemisahan lapisan secara fisik maka harus ada komunikasi yang menyeberangi batas-batas fisik tersebut seperti batas proses, mesin, atau bahkan batas tempat. Dengan mendesain aplikasi sistem yang terdistribusi dengan benar maka dapat diperoleh pendistribusian yang baik dan efisien.
- **Biaya pemeliharaan yang tinggi**. Biaya instalasi, biaya upgrade perangkat lunak, biaya pendistribusian, dan biaya administrasi meningkat secara signifikan.

Kebutuhan suatu arsitektur standar berdasarkan komponen sisi server telah muncul. Arsitektur ini perlu dibuat suatu antar muka yang baik antara server application yang berisi komponen. Antar muka (interface) memungkinkan komponen untuk dapat diatur secara portabel, dalam artian memungkinkan untuk menukar server aplikasi tanpa harus mengubah kode atau mengkompilasi ulang komponen tersebut.

2.2 Enterprise Java Bean

Enterprise bean merupakan perangkat lunak komponen yang disebar dalam suatu lingkungan multi-tier yang terdistribusi. Suatu *enterprise bean* bisa jadi terdiri dari satu atau lebih obyek Java, karena bisa jadi terdiri dari lebih dari satu obyek yang sederhana. Tanpa melihat komposisi suatu *enterprise bean*, klien dari bean berhubungan dengan suatu antar muka komponen. Antar muka ini, sebagaimana *enterprise bean* haruslah memenuhi spesifikasi EJB. Klien dari

enterprise bean dapat berupa aplikasi berbasis Java atau bahkan obyek enterprise bean lainnya.

2.2.1 Platform Sun Microsystem Java 2, Enterprise Edition

Sun Microsystem menyadari kebutuhan arsitektur komponen sisi server. Pada sisi klien Java terdapat banyak permasalahan, seperti ketidakkonsistenan antar muka user dengan platform, antar muka user yang memiliki kecepatan yang rendah, serta perbedaan versi dari mesin virtual Java (*Java Virtual Machine*) yang berjalan pada mesin klien. Tetapi bila diterapkan di sisi server, platform Java merupakan bahasa pemrograman yang ideal. Penyebaran pada sisi server berjalan dalam suatu lingkungan yang terkontrol yaitu selalu menggunakan versi Java yang konsisten (sama). Kecepatan Java juga topik masalah yang kurang berarti pada server karena pada 80 % atau lebih dari waktu aplikasi *n-tier* dihabiskan pada level basis data atau jaringan. Java juga merupakan bahasa yang sangat cocok untuk digunakan sebagai bahasa yang ditulis pada komponen sisi *server* dengan alasan bahwa pangsa pasar sisi server dikuasai oleh mesin UNIX dan mainframe. Hal ini berarti bahasa yang *cross-platform* untuk menulis komponen pada sisi *server* dapat meningkat nilainya karena pembuat perangkat lunak dapat menulis komponen sekali saja dan mendistribusikannya pada lingkungan server apa saja.

Dalam rangka komputasi sisi *server* dengan Java, Sun memproduksi platform yang disebut *Java 2 Platform, Enterprise Edition.* Misi dari J2EE adalah menyediakan suatu perangkat yang tidak tergantung pada *platform*, portable, multi user, aman, dan berkelas standar *enterprise* (bisnis) yang digunakan dalam penyebaran sisi server yang ditulis dalam bahasa pemrograman Java. J2EE

menyederhanakan banyak kekompleksan yang ada dalam pembuatan suatu aplikasi sisi *server* berbasis komponen.

2.2.2 Session Bean

Session bean merupakan obyek bisnis yang mengimplementasikan logika bisnis, aturan bisnis, atau aliran kerja suatu sistem. Sebagai contoh, suatu session bean dapat melakukan pengisian pesanan, kompresi file, transaksi bank, operasi basis data, kalkulasi yang kompleks, dan sebagainya. Session bean merupakan komponen yang dapat di *reused* dalam *pool* yang memiliki jangka waktu hidup hanya selama suatu sesi yaitu selama pemanggilan session bean oleh klien. Contohnya, jika klien mengontak suatu session bean untuk melakukan sesuatu, server aplikasi (*container/server EJB*) bertanggung jawab atas pembuatan sebuah instan dari komponen session bean. Ketika klien memutuskan hubungan (*disconnect*), server aplikasi memusnahkan (*destroy*) instan session bean yang terhubung dengan suatu obyek data. Session bean digunakan oleh satu klien pada suatu saat sehingga tidak dapat dibagi (*share*) antar klien. Ketika sebuah klien menggunakan suatu session bean, hanya klien tersebut yang berhubungan dengan session bean tersebut. Hal ini berkebalikan dengan entity bean yang bisa digunakan untuk melayani banyak klien.

Server EJB bertanggung jawab dalam mengatur waktu hidup bean. Klien tidak secara langsung menginstankan bean, container EJB malakukan hal ini secara otomatis. Demikian pula, EJB container memusnahkan (*destroy*) session bean pada saat yang tepat. Hal ini memungkinkan bean untuk di-*pooling* dan

digunakan oleh klien lainnya. Ada dua tipe dari session bean, yaitu *sateful session bean* dan *stateless session bean.*

2.2.3 Stateful Session Bean

Seperti yang disebutkan sebelumnya, session bean merepresentasikan proses bisnis. Beberapa proses bisnis dapat dilakukan oleh suatu permintaan metode (*method request*). Sebagaimana tiap-tiap user menggunakan sebuah situs web ecommerce yang online, user dapat menambah produk kedalam keranjang belanja. Hal ini mempengaruhi proses bisnis dalam memenuhi bermacam-macam permintaan bisnis. Konsekuensinya, tiap proses bisnis harus memeriksa status user dari suatu permintaan ke permintaan yang lain. Untuk memenuhi proses bisnis yang melayani bermacam-macam permintaan yang dibutuhkan penyimpanan data tiap-tiap klien dalam suatu sesi maka dibutuhkan peran *stateful session bean.* Jika suatu *stateful session bean* data berubah selama pemanggilan metode, data tersebut akan tersedia untuk klien selama jangka waktu pemanggilan sesi tersebut.

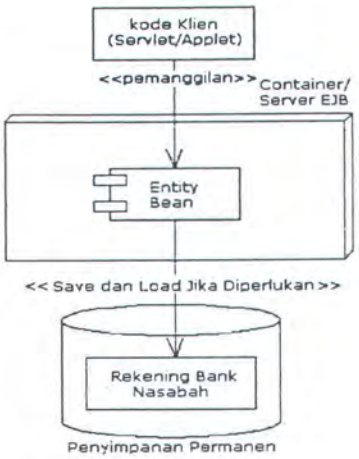
2.2.4 Stateless Session Bean

Beberapa proses bisnis secara umum digunakan untuk memenuhi konsep permintaan tunggal (*single request*). Pada permintaan tunggal terhadap proses bisnis ***tidak*** diperlukan peyimpanan data (status) selama terjadinya pemanggilan method tersebut. Stateless bean merupakan komponen-komponen yang dapat mengakomodasi tipe-tipe permintaan tunggal dari proses bisnis. Stateless session bean merupakan komponen bean dengan kinerja yang tinggi yang dapat memecahkan operasi matematika yang kompleks. Setelah menyelesaikan

tugasnya maka bean tersebut siap melayani permintaan yang berbeda tanpa menyimpan status/data dari permintaan yang sebelumnya.

2.2.5 Entity Bean

Bagian lain dari proses bisnis adalah data yang digunakan oleh proses bisnis. Dalam suatu skenario, komponen proses bisnis butuh untuk memanipulasi data yang ada dalam beberapa tempat penyimpanan data, seperti basis data relasional. *Entity bean* adalah suatu komponen yang merepresentasikan data yang bersifat permanen. Entity bean merepresentasikan obyek data yang nyata, seperti halnya entity dalam basis data. Entity bean tidak berisi logika proses bisnis sebagai model data. Session bean menangani proses bisnis yang menggunakan entity bean untuk merepresentasikan data yang digunakannya. Entity bean merupakan representasi obyek dari data-data tersebut yang membaca dari sekumpulan data yang terdapat dalam basis data yang disimpan kedalam suatu obyek entity bean yang dapat dimanipulasi dengan cara memanggil method yang ada. Sebagai contoh, *method bayar*() pada sebuah entity bean Nasabah, yang akan mengurangi jumlah uang dari suatu rekening bank dengan mengurangi jumlah dari saldonya. Sekali data entity bean dari suatu rekening bank disimpan maka dalam basis data akan berisi data saldo rekening yang baru. Pada dasarnya, entity bean mengimplementasikan lapisan logika akses data dalam arsitektur multi-tier seperti yang diperlihatkan gambar 2.7.



Gambar 2.7 Entity bean sebuah sudut pandang kedalam sebuah peyimpanan data utama.

Karena merupakan permodelan data yang permanen, jangka waktu hidup entity bean dapat bertahan lama, yang tahan terhadap *critical failures*, seperti kerusakan pada server aplikasi dan *fault-toleran.* Jika suatu mesin mengalami kerusakan, entity bean dapat direkonstruksi lagi kedalam memory dengan mudah yaitu dengan jalan membaca kembali data dari basis data permanen. Hal tersebut merupakan perbedaan yang utama antara session bean dan entity bean, dimana entity bean berumur lebih lama daripada sesi klien, bahkan tergantung pada berapa lamanya data berada dalam basis data.

Entity bean berbeda dari session bean dari segi kuantitas klien yang menggunakan entity bean secara simultan. Banyak klien memanipulasi data yang sama dalam suatu basis data secara bersamaan sehingga entity bean mampu mengisolasi suatu klien terhadap klien lainnya dengan mengimplementasikan

transaksi. Apabila digunakan dengan benar, transaksi menjamin bahwa banyak klien melakukan operasi peyimpanan data secara konsisten.

Kenyataannya, *record-record* basis data merepresentasikan sebuah obyek yang telah ada sebelum penggunaan solusi berbasis java karena pada struktur basis data dapat tidak tergantung pada bahasa pemrograman apapun (*legacy system*). *Record-record* basis data dapat dibaca dan diinterpretasikan sebagai obyek-obyek dengan platform berbagai macam bahasa pemrograman. EJB menggunakan situasi ini dengan cara memperbolehkan pengubahan suatu struktur data menjadi obyek java. Ada dua sub tipe entity bean yaitu *bean-managed persistent* entity bean dan *container-managed persistent* entity bean.

2.2.6 Bean-Managed Persistent (BMP) Entity Beans

Entity bean merupakan komponen yang bersifat *persistent* karena obyek datanya disimpan dalam suatu basis data. Entity bean melakukan pemetaan terhadap obyek data relasional yaitu meletakkan suatu obyek dalam memori dan memetakan obyek tersebut kedalam suatu tempat penyimpanan *record-record* basis data tertentu agar dapat diakses pada suatu saat.

Bean-managed persistent entity bean merupakan entity bean yang melakukan operasionalnya secara manual, dengan kata lain programmer perangkat lunak harus menulis kode yang bertujuan untuk mentranslasikan (memetakan) field-field memori kedalam tempat penyimpanan, seperti pada basis data relasional atau sebuah basis data berorientasi obyek.

2.2.7 Container-Managed Persistent (CMP) Entity Bean

EJB mengijinkan programmer *entity bean* untuk tidak perlu khawatir mengenai pengkodean logika (algoritma). Contaner/server melakukan seluruh tugas-tugas pengaksesan data dalam basis data seprti penyimpanan, pengambilan, dan pencarian obyek data. Dengan CMP ini, programmer tidah perlu menuliskan kode-kode program seperti yang dilakukan ketika menggunakan metode BMP entity bean karena logika pemrograman telah dituliskan kedalam suatu *file property*.

2.2.8 Komponen EJB

Komponen enterprise bean terbentuk dari sejumlah file yang bekerja secara bersamaan yang mengikuti suatu aturan tertentu. File-file tersebut adalah sebagai berikut

2.2.8.1 Enterprise Bean Class

Enterprise bean class mengimplementasikan *javax.ejb.SessionBean* yang yang mengimplementasikan *javax.ejb. EnterpriseBean.* Antar muka *javax.ejb. EnterpriseBean* mengimplementasikan *java.io.Serializable* yang berarti bahwa semua enterprise bean dapat dikonversi menjadi *bit-blob* dan membagi obyek yang bersifat *serializable.*

2.2.8.2 Obyek EJB

Ketika klien akan menggunakan suatu instan dari *enterprise bean class,* klien tidak pernah memanggil *method* secara langsung pada instan bean yang sebenarnya, tetapi lebih dahulu dicegat oleh container EJB baru kemudian diteruskan kepada instan bean koresponden. Container EJB bertindak sebagai

suatu lapisan antara kode klien dan bean. Lapisan ini disebut obyek EJB yang berkemampuan menangani jaringan, transaksi, dan lainnya.

2.2.8.3 Remote Interface

Klien memanggil Obyek EJB yang berisi *method-method* bisnis yang yang ada dalam *bean class*. Antar remote muka bertugas menduplikasi seluruh method logika bisnis tersebut yang diharuskan memenuhi aturan-aturan spesifikasi EJB yang ada dalam antar muka *javax.ejb.EJBObject*. Ketika klien bean memanggil *method* bisnis tersebut, Obyek EJB meneruskan pemanggilan tersebut kepada implementasi bean yang sesungguhnya.

2.2.8.4 Obyek Home

Klien tidak bisa menginstan obyek EJB secara langsung karena obyek EJB kemungkinan dapat berada pada server yang berbeda. Klien diharuskan meminta obyek EJB dari obyek home yang menghasilkan obyek EJB yang bertanggung jawab untuk menginstan dan menhancurkan (*destroying*) obyek EJB.

2.2.8.5 Interface Home

Antar muka home mendefinisikan *method* yang membuat, menghancurkan, dan menemukan obyek EJB. Tiap EJB harus mengimplementasikan antar muka *javax.ejb.EJBHome* yang diturunkan dari *java.rmi.Remote* yang berati bahwa antar muka home berkemampuan sebagaimana obyek-obyek java *Remote Method Invocation* (*RMI*) lainnya yang dapat dipanggil menyebrangi mesin virtual Java (*JVM*) yang berlainan.

2.2.8.6 Deployment Descriptor

Deployment descriptor bertugas menyediakan informasi layanan yang disediakan komponen EJB. File ini berisi layanan-layanan seperti kebutuhan menajemen dan siklus hidup bean, transaksi, dan keamanan.

2.2.8.7 File EJB-jar

Keseluruhan file yang ada diatas dikompresi kedalam format *.zip* membentuk file EJB-jar. File ini siap untuk didistribusikan dalam suatu server aplikasi tertentu.

2.3 TRANSAKSI

Transaksi merupakan metode yang relatif aman dalam mengikutsertakan komponen dalam suatu operasi sistem yang terdistribusi. Suatu transaksi merupakan rangkaian dari operasi-operasi yang dieksekusi sebagai suatu operasi yang besar dan bersifat *atomic*. Transaksi memungkinkan banyak user untuk membagi data yang sama dan menjamin bahwa sejumlah data yang di*update* (diperbaharui) akan secara keseluruhan atau sebagian disimpan tanpa ada pengaruh *update* oleh transaksi yang dilakukan oleh klien lainnya. Transaksi merupakan suatu bentuk kontrol konkurensi yang rumit. Dalam konteks penggunaan, transaksi memastikan bahwa basis data akan disimpan secara konsisten. Transaksi juga memastikan bahwa dua operasi dari komponen basis data diisolasi antara satu terhadap yang lainnya. Tanpa menggunakan transaksi, basis data dapat mengalami korupsi sehingga tidak dapat diimplementasikan kedalam suatu suatu aplikasi yang rumit sebagaimana pada aplikasi bertaraf *enterprise* (bisnis). Kontainer/server EJB menangani operasi-operasi transaksi,

mengkoordinasikan proses-proses dibelakang layar dari transaksi-transaksi yang berpartisipasi. Keutungan yang diperoleh dari penggunaan EJB disini adalah transaksi dapat dilakukan secara implisit dan otomatis. Ada beberapa faktor yang mendorong penggunaan transaksi seperti berikut ini

- **Operasi atomic**. Dalam melakukan banyak operasi yang bersifat diskrit, perlu diketahui apakah operasi-operasi tersebut telah dieksekusi baik secara luas, terus menerus, dan bersifat *atomic*. Sebagai contoh adalah transfer sejumlah saldo antar rekening nasabah. Pada kasus ini, terjadi pengurangan jumlah dana dari satu rekening dan menyimpannya ke rekening yang lain. Kondisi yang ideal adalah apabila kedua operasi sukses, tetapi jika terjadi suatu kekeliruan maka keduanya seharusnya digagalkan, jika tidak maka akan terdapat jumlah saldo yang salah pada rekening bank tersebut. Transaksi harus dilakukan secara keseluruhan atau tidak sama sekali, jika suatu operasi sukses sedangkan yang mengalami kegagalan maka transaksi tersebut harus dilakukan berdasarkan sifat atomic.
- **Kerusakan Jaringan atau Mesin**. Pendistribuasian dalam lingkungan multi-tier memerlukan keamanan, skalabilitas, dan modularitas. Mendistribusikan aplikasi melintasi jaringan memungkinkan terjadinya kerusakan dan masalah reliabilitas. Sebuah *exception* akan dilemparkan ketika jaringan *crash* dan dikirimkan kepada klien, akan tetapi *exception* ini bisa jadi membingungkan. Jaringan bisa saja akan mengalami kerusakan sebelum suatu data diubah. Mungkin juga pada jaringan terjadi kerusakan setelah adanya pengubahan data. Tidak ada cara untuk membedakan kasus

tersebut, semua kode klien melemparkan *exception* kerusakan jaringan sehingga tidak dapat secara pasti diketahui data yang benar. Kenyataannya, jaringan mungkin bukanlah satu-satunya sumber permasalahan, karena permasalahan bisa timbul pada suatu data yang merupakan informasi yang bersifat persisten yang berada dalam suatu basis data, akan mungkin kalau data tersebut mengalami kerusakan. Pada mesin dimana data distribusikan dapat juga terjadi kerusakan. Jika suatu kerusakan terjadi selama penulisan ke basis data, maka dalam basis data tersebut sangat mungkin mengalami ketidakkonsistenan data. Hal tersebut tidak dapat diterima oleh aplikasi enterprise yang bersifat *mission-critical*. Sistem mainframe maupun sistem yang lebih baik lainnya yang tersedia menawarkan tindakan preventif agar dapat menghindari kerusakan sistem, akan tetapi pada kenyataannya tidak ada yang sempurna. Mesin, proses, ataupun jaringan akan selalu mungkin mengalami kerusakan sehingga harus ada proses perbaikan untuk menanganinya.

- **Banyak user yang saling membagi data**. Dalam sistem terdistribusi pada level enterprise, terdapat susunan yaitu banyak klien yang berhubungan dengan banyak server aplikasi yang berhubungan dengan data dalam sebuah basis data. Sangat mungkin jika beberapa klien membutuhkan penggunaan suatu peralatan secara simultan, operasi tersebut sangat mungkin dapat saling mendahului, sehingga memungkinkan basis data berisi data yang tidak konsisten. Hal ini seringkali merusak konsistensi data. Kekeliruan data dapat berdampak terhadap keadaan data-data lainnya, sehingga diperlukan

adanya suatu mekanisme dalam memenuhi interaksi antara banyak user secara konkuren dalam memodifikasi data. Harus ada jaminan user yang melakukan operasinya secara konkuren untuk mengupdate data tanpa membuat data tersebut keliru.

2.3.1 Keuntungan Transaksi

Permasalahan ketidakkonsistenan data dapat dihindari dengan mengimplementasikan transaksi. Suatu transaksi merupakan rangkaian operasi yang digunakan untuk mengeksekusi suatu operasi yang luas dan *atomic*. Transaksi menjamin perubahan nilai yang bersifat *all-or-nothing* (keseluruhan atau tidak ada sama sekali): keseluruhan operasi yang dilakukan akan sukses atau keseluruhan sama sekali gagal. Transaksi memungkinkan banyak user membagi data yang sama dan menjamin bahwa suatu kumpulan data yang di*update* akan ditulis keseluruhan atau sebagian dengan tanpa adanya *update* dari klien lainnya. Sebagai contoh, jika dua klien membaca atau menulis dari data pada basis data yang sama, akan ada *mutually exclusive* dalam penggunaan transaksi. Sistem basis data akan secara otomatis melakukan pengontrolan konkurensi yang dibutuhkan (seperti *locking*) pada basis data untuk menjaga *thread* klien dari pengaruh transaksi yang lain.

Ketika transaksi digunakan dengan benar, operasi akan selalu dieksekusi dengan empat jenis jaminan. Keempat jaminan tersebut diketahui secara umum sebagai properti transaksi ACID. Kata ACID merupakan singkatan dari *Atomicity, Consistency, Isolation,* dan *Durability*.

- *Atomicity* menjamin bahwa banyak operasi yang dilakukan secara bersamaan dan tampil sebagai satu unit kerja yang terus menerus. Dalam permasalahan bank, ketika mentransfer uang dari satu rekening bank ke rekening yang lain, maka terjadi pengurangan jumlah saldo pada suatu rekening dan penambahan saldo pada rekening yang lain. Proses tersebut seharusnya terjadi atau tidak sama sekali. Atomicity menjamin kalau operasi yang dilakukan dengan menggunakan transaksi dilakukan berdasarkan paradigma *all-or-nothing* (keseluruhan atau tidak sama sekali) – keseluruhan basis data diupdate atau tidak ada yang sama sekali yang terjadi jika pada suatu saat muncul kekeliruan.
- **Konsistensi** menjamin bahwa sebuah transaksi akan meninggalkan keadaan sistem untuk konsisten setelah sebuah transaksi secara utuh dilaksanakan. Misalnya dengan asumsi bahwa suatu keadaan sistem bank konsisten jika mengikuti asumsi bahwa "saldo rekening bank haruslah selalu bernilai positif". Ketidakkonsistenan dapat terjadi, misalnya komponen enterprise bean pada suatu saat menghasilkan saldo rekening yang bernilai negatif selama penarikan saldo. Ketika transaksi selesai, keadaan kembali konsisten dimana Bean tidak pernah meninggalkan nilai negatif pada saldo rekening. Meskipun secara sementara keadaan yang terjadi mungkin tidak konsisten, hal ini tidak masalah karena transaksi dilakukan secara *atomic*, berdasarkan unit kerja yang berurutan. Atomicity selalu memaksakan kondisi agar selalu konsisten.

- **Isolasi** melindungi secara konkuren dalam mengeksekusi transaksi dari pengaruh transaksi lainnya yang tidak komplet. Isolasi memungkinkan keadaan banyak transaksi dibaca dan ditulis pada suatu basis data tanpa perlu mengetahui keadaan yang lainnya, karena tiap transaksi terisolasi dari yang lainnya. Tanpa isolasi, keadaan aplikasi mungkin dapat menjadi tidak konsisten dimana hal ini akan sangat berguna untuk banyak klien dalam memodifikasi suatu basis data. Untuk tiap-tiap klien akan muncul layaknya hanya ada satu klien yang memodifikasi basis data pada tiap saat tersebut. Sistem transaksi diisolasi dengan menggunakan protokol sinkronisasi *low-level* (level rendah) pada suatu basis data. Sinkronisasi ini mengisolasi kerja antara suatu transaksi dengan yang lainnya. Selama suatu transaksi, terjadi *locking* (penguncian) data secara otomatis dan dilakukan seperlunya. Jika suatu transaksi mengunci suatu data maka penguncian tersebut menahan transaksi secara konkuren dari interaksi lainnya terhadap data tersebut sampai *locking* dilepaskan. Sebagai contoh, saat menulis rekening data kedalam basis data, transaksi melakukan *lock pada* tabel atau record rekening bank. *Lock* menjamin bahwa selama transaksi terjadi, tidak ada konkurensi pengupdatean yang dapat memperungaruhi dimana hal ini memungkinkan banyak user untuk memodifikasi kumpulan record-record basis data secara simultan tanpa mempermasalahkan akan terjadi operasi basis data yang mungkin dapat saling *interleaving* (mendahului).
- **Durability** menjamin peng-*update*-an terhadap sumber daya seperti halnya *record-record* basis data haruslah dapat menghindari kekeliruan. Beberapa

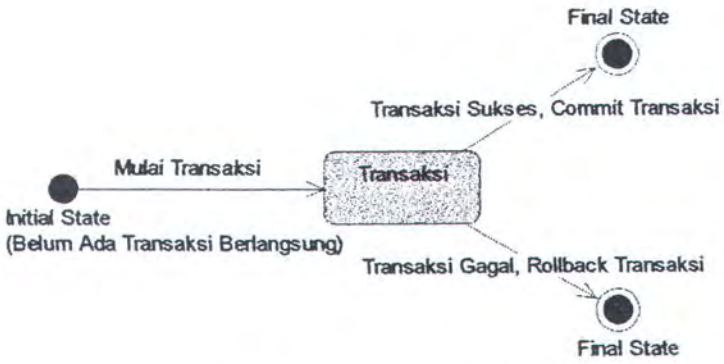
contoh kekeliruan tersebut misalnya kerusakan mesin, jaringan, atau hard disk. Sumber daya yang *recoverable* (mampu memperbaiki /memulihkan diri) menyimpan suatu *log* (catatan) transaksi dalam memenuhi tujuan ini. Jika sumber daya *crash*, data permanen dapat diperbaiki kembali dengan mengulangi langkah-langkah yang ditulis dalam dalam *log* tersebut.

2.3.2 Model Transaksi

Ada banyak permodelan transaksi yang berbeda, tiap model memiliki kelebihannya dan ciri khas tersendiri. Dua model yang paling populer yaitu *flat transaction* dan *nested transaction.*

Flat transaction merupakan model transaksi paling sederhana dan mudah dipahami. *Flat transaction* merupakan kumpulan operasi yang dilakukan secara *atomic* sebagai suatu unit kerja yang tunggal. Setelah suatu *flat transaction* mulai dilakukan, aplikasi melakukan beberapa operasi. Beberapa operasi tersebut mungkin operasi yang bersifat persisten dan yang lainnya tidak. Ketika pada akhirnya diputuskan untuk mengakhiri transaksi, selalu ada hasil yang bersifat biner, yaitu sukses atau gagal. Transaksi yang sukes akan di *commit* (disimpan), sedangkan transaksi yang gagal akan digagalkan pula. Ketika suatu transaksi di *commit* (disimpan), seluruh operasi persisten menjadi perubahan yang telah permanen. Jika tansaksi *aborted* (digagalkan), maka tidak ada pengupdatean pada sumber daya yang dilakukan, dan perubahan yang dihasilkan akan di-*rollback.* Kondisi ini memenuhi konsep "*all or nothing*" yang berarti bahwa saldo suatu rekening bank dapat diambil dan disimpan pada suatu rekening bank yang lain

berdasarkan pada informasi bahwa operasi tersebut sukses (*commit*) atau gagal (*rollback*) seperti yang diilustrasikan pada gambar 2.9.



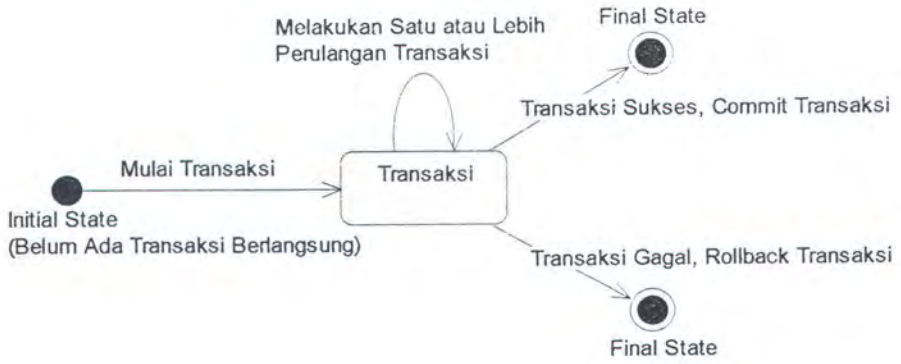
Gambar 2.8 Flat Transaction (Transaksi flat)

Nested transaction lebih mudah dipahami jika diikuti ilustrasi skenario sebagai berikut,

- Aplikasi X harus dapat membeli tiket kereta api dari Surabaya ke Malang.
- Aplikasi X harus dapat membeli tiket pesawat dari Malang ke Jakarta
- Aplikasi X harus dapat membeli tiket penerbangan balon udara dari Jakarta ke Denpasar.
- Aplikasi menemukan bahwa tidak ada penerbangan ke Denpasar.

Jika masalah pemesanan sekuensial ini dilakukan berdasarkan transaksi *flat*, maka aplikasi harus melakukan *rollback*. Karena tidak ada tiket penerbangan balon udara ke Denpasar, maka aplikasi akan menggagalkan keseluruhan pemesanan. Akan tetapi, ada suatu cara untuk memindahkan penerbangan balon udara dengan perjalanan yang lainnya yang dapat menyelamatkan pembelian tiket kereta api dan pesawat. *Nested transaction* dapat memecahkan permasalahan ini, dengan cara

embed (menyatukan) unit-unit kerja *atomic* dengan unit kerja yang lainnya. Unit kerja *nested* (tersarang) dengan unit kerja lainnya dapat *rollback* tanpa memaksakan *rollback*-nya keseluruhan transaksi, hal ini diilustrasikan pada gambar 2.10. Sub transaksi dapat berbentuk *flat* atau *nested.*



Gambar 2.9 Nested transaction (Transksi tersarang)

Jenis transaksi lainnya. Ada jenis transaksi yang lain, tetapi EJB tidak mendukung jenis transaksi tersebut, seperti *chained transaction* dan *sagas.*

2.3.3 Transaksi yang Terdistribusi

Transaksi *flat* yang paling mendasar terjadi pada sebuah *server aplikasi* tunggal yang berdasarkan pada basis data tunggal. Berdasarkan kegunaan pelayanan transaksi yang disajikan server aplikasi, transaksi *flat yang* terdisitribusi tidak memperdulikan aturan yang sama dengan transaksi *flat* biasa – jika suatu komponen pada suatu mesin menggagalkan transaksi, keseluruhan transaksi akan digagalkan. Transaksi *flat* yang terdistribusi dapat berjalan melalui banyak *server aplikasi* dan basis data, yang berarti bahwa kompoenen-kompoenen dapat disebar dalam suatu server aplikasi yang berjalan dengan transaksi yang sama sebagaimana komponen lainnya disebar pada server aplikasi yang lain. Hal

ini penting jika melihat bahwa banyak mesin yang saling berkolaborasi melintasi suatu jaringan dalam memenuhi suatu permasalahan bisnis. Transaksi *flat* yang terdistribusi memperbolehkan banyak *server aplikasi*, ditulis oleh vendor yang berbeda dalam melaksanakan suatu transaksi.

2.3.4 Kebutuhan Pengontrolan Konkurensi

Isolasi menjamin konkurensi user yaitu dengan mengisolasi dari user lainnya, meskipun sama-sama memakai data pada basis data yang sama. Diberikan contoh sebagai berikut, ada dua instan dari komponen yang sama yang dieksekusi secara konkuren yang mungkin juga berada pada dua proses atau *thread* yang berbeda. Diasumsikan komponen ingin mengupdate suatu basis data yang di-*share* (dibagi) menggunakan basis data API seperti JDBC atau SQL/J. Tiap-tiap instan dari komponen melakukan langkah-langkah sebagai berikut

Baca sebuah integer X dari basis data

Tambahkan 10 pada X

Simpan nilai yang baru dari X ke basis data

Jika tiap-tiap langkah diatas dieksekusi secara bersamaan dalam suatu operasi *atomic*, maka hasilnya akan memuaskan. Tidak ada instan yang mempengaruhi operasi instan yang lain, karena telah dijamin algoritma penjadwalan. Jika ada dua instan mengeksekusi ketiga operasi tersebut, maka akan terjadi sebagai berikut

Instan A membaca integer X dari basis data. Basis data sekarang berisi X=0.

Instan B membaca integer X dari basis data. Basis data sekarang berisi X=0.

Instan A menambah 10 pada salinan dari X dan menyimpannya kedalam basis data. Basis data sekarang berisi X=10.

Instan B menambah 10 pada salinan dari X dan menyimpannya kedalam basis data. Basis data sekarang berisi X=10.

Jika ada *interleaving* (saling mendahului) dari operasi basis data, instan B bekerja dengan mengambil nilai salinan dari X: salinan sebelum instan A meyimpan nilainya. Kemudian instan A menghilang, permasalahan ini lebih dikenal dengan *lost update.* Pemecahannya dengan menggunakan *locking* (penguncian) pada basis data untuk menghindari dua komponen dari pembacaan data. Dengan me-*lock* data yang sedang digunakan untuk transaksi, dapat dijamin bahwa hanya transaksi tersebut yang mengakses data tersebut sampai *locking* dilepaskan. Dengan *exclusive lock* (penguncian ekslusif), maka skenarionya menjadi sebagai berikut

Permintaan sebuah lock dari basis data.

Baca sebuah integer X dari basis data.

Tambahkan 10 pada X.

Tulis nilai yang baru dari X ke basis data.

Melepaskan lock pada X.

Jika ada komponen lain yang berjalan secara konkuren bersamaan dengan komponen yang lain, komponen akan menunggu sampai *lock* dilepaskan, yang mana akan memberikan komponen tersebut salinan dari X.

2.3.5 Isolasi dan EJB

Isolasi pada suatu transaksi dapat diset baik secara ketat maupun longgar. Perlu diketahui mengenai lapisan isolasi yang dibutuhkan, EJB menawarkan level isolasi yang berbeda yang dapat memberikan fleksibilitas. Level isolasi memungkinkan untuk menspesifikan kontrol konkurensi pada level yang tinggi. Jika isolasi dilakukan secara ketat, dapat dipastikan bahwa tiap transaksi akan terisolasi dari transaksi yang lain, dengan berimbas pada tingkat kinerja. EJB container mengetahui bagaimana untuk memeriksa deskriptor pendistribusian untuk mengeset level isolasi pada bean yang dipakai.

2.3.6 Permasalahan Dirty Read

Dirty read[4], terjadi ketika membaca data dari suatu basis data yang belum di*commit* pada tempat penyimpanan data yang permanen. Contoh kasus, dua instan dari komponen yang sama yang melakukan pekerjaan sebagai berikut

Membaca integer X dari basis data. Sekarang basis data berisi X=0.

Menambahkan 10 terhadap X dan menyimpannya kedalam basis data. Sekarang basis data berisi X=10. Aktivitas commit belum dilakukan, sehingga perubahan belum bersifat permanen.

Aplikasi lain membaca nilai integer X=10.

Operasi digagalkan, dengan mengembalikan nilai basis data X=0.

Aplikasi yang lain menambahkan 10 kepada X dan menyimpannya kedalam basis data, sekarang basis data berisi X=20.

---

[4] [ECHOLS 88] halamn 184, 467, pembacaan yang kotor(keliru)

Permasalahan yang terjadi adalah aplikasi yang lain membaca update sebelum di *commit.* Pada data X mengalami kekeliruan, hal ini disebabkan karena pembacaan data yang belum di *commit.*

2.3.7 Permasalahan Unrepeatable Read

Pembacaan *unrepeatable* terjadi ketika suatu komponen membaca data dari suatu basis data, tetapi pada proses pembacaan kembali data tersebut, data telah berubah. Hal ini dapat terjadi ketika ada konkurensi lainnya yang mengeksekusi transaksi melakukan modifikasi pada data yang sedang dibaca.

Membaca sebuah data X dari basis data.

Aplikasi lain menumpuki data X dengan nilai yang baru.

Membaca kembali data X dari basis data. Nilainya berubah secara aneh.

Pemecahan permasalahan ini adalah dengan melakukan *lock* terhadap transaksi yang lain dalam memodifikasi data sehingga dapat dijamin permasalahan *unrepeatable read* tidak akan terjadi.

2.3.8 Permasalahan Phantom

Phantom adalah sebuah kumpulan data baru yang tiba-tiba muncul di sebuah basis data antara dua operasi pembacaan. Contohnya

Aplikasi meng-query basis data menggunakan beberapa kriteria dan mendapatkan sekumpulan data.

Aplikasi lainnya menyisipkan data baru yang memenuhi query yang diberikan.

Query dilakukan lagi, dan data baru muncul dengan tiba-tiba.

kedalam dua fase disebut protokol *two-phase commit* atau 2PC[6] yang berguna karena memungkinkan banyak pengatur transaksi dan sumber daya untuk berpartisipasi dalam sebuah transaksi melintasi sebuah pendistribusian. Jika ada pemilihan partisipan yang harus digagalkan oleh transaksi, keseluruhan partisipan harus *rollback* juga.

2.4 UNIFIED MODELLING LANGUAGE

2.4.1 Umum

UML atau Unified Modeling Language merupakan bahasa yang digunakan untuk menspesifikasikan, memvisualisasikan dan membangun produk sistem perangkat lunak. UML merupakan notasi yang bertujuan untuk memodelkan sistem dengan menggunakan pendekatan yang berorientasi obyek.

UML merupakan standar industri untuk model yang berorientasi obyek yang dibangun dari tiga metoda pendahulunya, yaitu metoda Booch (oleh Grady Booch), metoda OMT (Object Modeling Technique oleh James Rumbaugh) dan metoda OOSE (Object Oriented Software Engineering oleh Ivar Jacobson).

Sebuah badan standarisasi industri, OMG (*Object Management Group*) kemudian menetapkan standar UML v1.0 pada November 1997

2.4.2 Diagram UML

UML memiliki sembilan jenis diagram sistem, yaitu :

- Aktifitas / Activity
- Class

---

[6] [Roman 99] halaman 291

- Collaboration
- Component
- Deployment
- Object
- Use case
- Sequence
- State

Kesembilan diagram di atas dapat dikelompokkan ke dalam tiga kategori, yaitu statis, dinamis dan arsitektural.

- *Static Diagram* (diagram statis) menggambarkan struktur dari suatu sistem dan fungsi serta tanggungjawabnya. Yang termasuk ke dalam diagram statis adalah Use case, Class dan Object diagram.
- *Dynamic Diagram* (Diagram dinamis) menggambarkan interaksi atau hubungan-hubungan yang didukung oleh sistem tersebut. Yang termasuk di dalam kategori diagram dinamis ini diantaranya Activity, Collaboration, Sequence, State dan Use Case diagram.
- *Architectural Diagram* (diagram arsitektural) menggambarkan realisasi dari sistem tersebut menjadi komponen-komponen yang dapat dijalankan dan dieksekusi. Yang termasuk ke dalamnya adalah Component dan Deployment diagram.

Pada prakteknya nanti, setiap proyek yang ingin digambarkan, paling tidak akan memiliki minimum tiga diagram, yaitu Use Case, Class dan Sequence Diagram.

BAB III

PERANCANGAN SISTEM

3.1 Umum

Bab ini membahas perancangan prototipe dari sistem pembayaran online yang melibatkan proses login, pembayaran tagihan oleh nasabah, administrasi user, dan pengawasan transaksinya. Permodelan masalah dideskripsikan menggunakan metode UML (*Unified Modelling Language*) dengan tool Rational Rose'98i Enterprise Edition, dengan pendekatan pembuatan perangkat lunak bolak-balik (*Round-Trip Software Development*) sehingga desain yang telah selesai memungkinkan untuk diubah apabila telah memasuki tahap implementasi nantinya.

Dalam arsitektur sistem, dijelaskan implementasi permasalahan menggunakan arsitektur 3-tier beserta konfigurasi perangkat keras, perangkat lunak dan jaringan yang terlibat.

3.2 Metodologi Perancangan

Perancangan sistem melibatkan tahapan sebagai berikut

- Analisa Use Case

  Secara umum, use case diagram menggambarkan kumpulan *properties* dan *methods* yang digunakan secara bersama-sama mendeskripsikan permodelan suatu kasus dengan cara mendefinisikan dan memanipulasi kumpulan diagram class dalam suatu skenario. Dalam konteks ini, use case diagram

digunakan untuk mendeskripsikan hasil analisa sistem pembayaran online yang melibatkan interaksi antara aktor dan use case.

- Perancangan Proses

  Dalam setiap use case terdapat beberapa skenario aliran data yang ada dalam proses disertai dengan objek yang berperan. Skenario aliran data dan obyek tersebut digambarkan secara grafis menggunakan *sequence diagram* dan *collaboration diagram*. *Sequence diagram* menggambarkan interaksi obyek berbasis waktu sedangkan *collaboration diagram* menggambarkan bagaimana hubungan antar tiap-tiap obyek. Dalam tahap ini digambarkan skenario yang terjadi dalam sistem pembayaran online melalui kedua jenis diagram tersebut.

- Perancangan Basis Data

  Obyek-obyek yang yang terlibat dalam sistem berpeluang untuk menjadi entitas basis data dilengkapi dengan atribut-atribut yang menyertainya untuk dipilih dan diimplementasikan berdasarkan metode basis data relational dalam *class diagram*.

- Perancangan Diagram 3-Tier

  Arsitektur sistem diimplementasikan menggunakan model 3-tier yang terjadi pemisahan sistem menjadi 3, yaitu user service, bisnis service, dan data service.

3.3 Arsitektur Sistem

Prototype sistem pembayaran online dikembangkan dengan arsitektur 3-tier seperti terlihat pada gambar 3.1. Model ini yang mendistribusikan obyek-obyek proses bisnis antara bisnis service dengan user service. Pemisahannya adalah

- Aplikasi klien (user service)

  Aplikasi klien dibagi menjadi 2 yaitu aplikasi pembayaran yang diakses secara langsung oleh nasabah dan aplikasi administrasi yang yang dibuat dengan *Java Server Pages* (JSP) yang dijalankan pada suatu server aplikasi

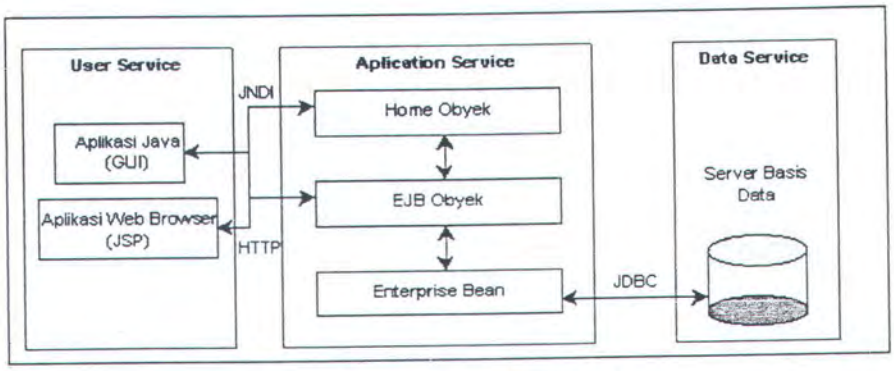
- Aplikasi bisnis service berbasis *Enterprise Java Bean* (EJB) (bisnis service)

  Obyek-obyek EJB disebar dalam suatu server aplikasi untuk melayani *request* klien yang terletak di user service. Komponen EJB dijalankan pada suatu server aplikasi.

- Basis data (data service)

  Data-data disimpan dalam server basis data menggunakan RDBMS Oracle 8i. Basis data ini diakses oleh bisnis service untuk mengelola data.

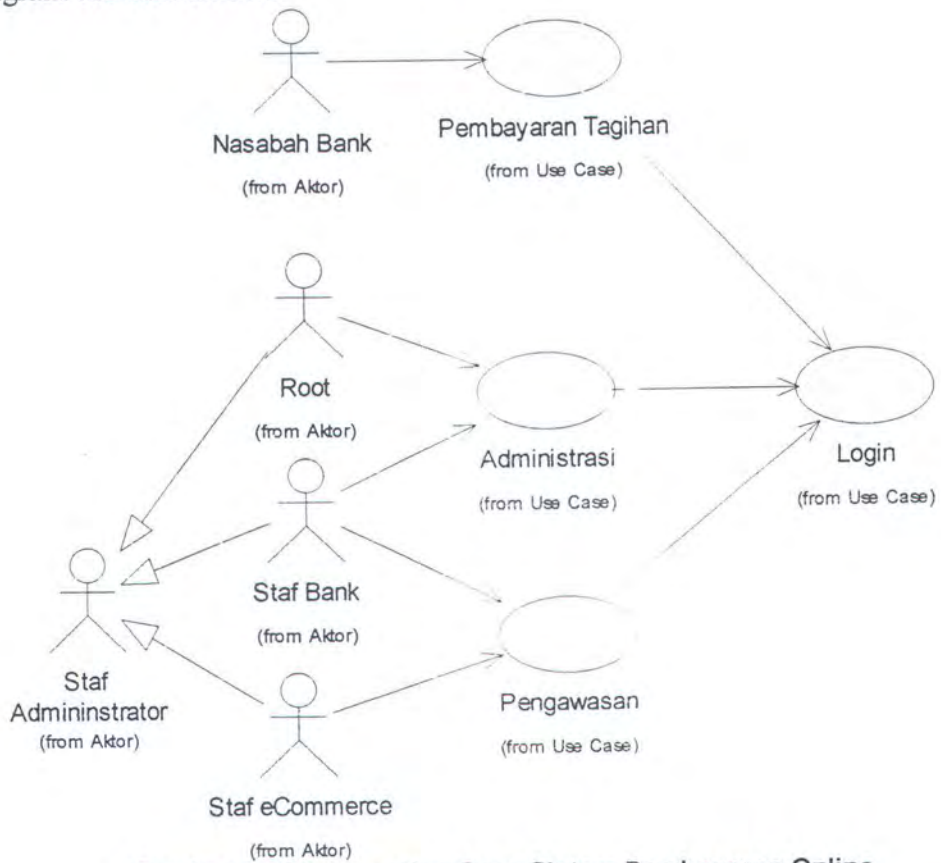
Server aplikasi terhubung dengan web server agar dapat melayani kebutuhan user melaui *web browser* pada situs web yang disediakan oleh penyedia layanan ecommerce.



Gambar 3.1 Diagram arsitektur 3-tier.

3.4 UML Use Case

Secara umum, dalam sistem pembayaran online digambarkan dalam diagram use case berikut



Gambar 3.2 Diagram Use Case Sistem Pembayaran Online

Dalam diagram UML Use Case diatas terdapat 4 proses yang dilambangkan dengan use case, yaitu login, pembayaran tagihan, administrasi, dan pengawasan.

Tiap-tiap proses diharuskan melalui tahapan otentikasi dan otorisasi dengan cara *login* agar dapat dioperasikan oleh aktor (*user*). Aktor yang terlibat adalah nasabah bank, staf administrator yang di-*generalize* menjadi root, staf bank, dan staf ecommerce. Keseluruhan aktor melakukan aktifitasnya melalui web browser.

3.4.1 Deskripsi Aktor

Aktor yang terlibat dalam diagram UML Use Case Pembayaran Online adalah

- Nasabah

  Aktor ini melakukan aktifitas pembayaran. Nasabah memiliki rekening pada bank yang sama dengan pihak eCommerce. *Role* (hak) akses diberikan kepada *method* yang terlibat dalam operasi pembayaran.

- Root

  Aktor root harus ada dalam sistem dan berada dipihak bank yang digolongkan dalam *role* yang dapat melakukan administrasi data-data administrator dan ecommerce. Aktor ini tidak memiliki hak untuk melakukan pengawasan.

- Staf Bank

  Aktor ini digolongkan dalam *role* yang dapat melakukan administrasi dan mengawasi pelaksanaan operasional sistem secara keseluruhan. Bank memiliki minimal satu staf bank sebagai administrator.

- Staf eCommerce

  Aktor ini digolongkan dalam *role* yang bertugas mengawasi operasional sistem berdasarkan id ecommerce yang menjadi tanggung jawabnya. Setiap ecommerce memiliki minimal satu staf ecommerce sebagai administrator.

3.4.2 Aliran Skenario Use Case

Use case digunakan sebagai suatu cara yang spesifik dalam menggambarkan sistem berdasarkan perspektif *user* (aktor). Masing-masing use case memiliki skenario proses yang dideskripsikan dalam satu atau lebih *sequence diagram* dan *collaboration diagram* yang saling bersesuaian.

3.4.2.1 Use Case Login

Masing-masing aktor harus melakukan login dahulu untuk otentikasi user sebelum melakukan aktifitas lainnya. Informasi yang dimasukkan oleh user digunakan untuk otorisasi role sehingga dapat mengakses method tertentu yang menjadi haknya. Terdapat dua skenario login yaitu skenario login administrator dan nasabah. Perbedaannya adalah pada skenario login administrator mengecek dahulu kategori usernya sedangkan untuk login nasabah tidak.

3.4.2.1.1 Skenario Login Administration

Aktor staf administrator memanggil memanggil method BtnLogin_Click() di class USLoginAdministrator. Kemudian class tersebut membuat obyek LoginSession untuk memanggil method Login(). Obyek LoginSession mengotentikasi dan mengotorisasi administrator melalui obyek Role untuk mengecek password, kategori, dan penambahan user kedalam role tertentu.

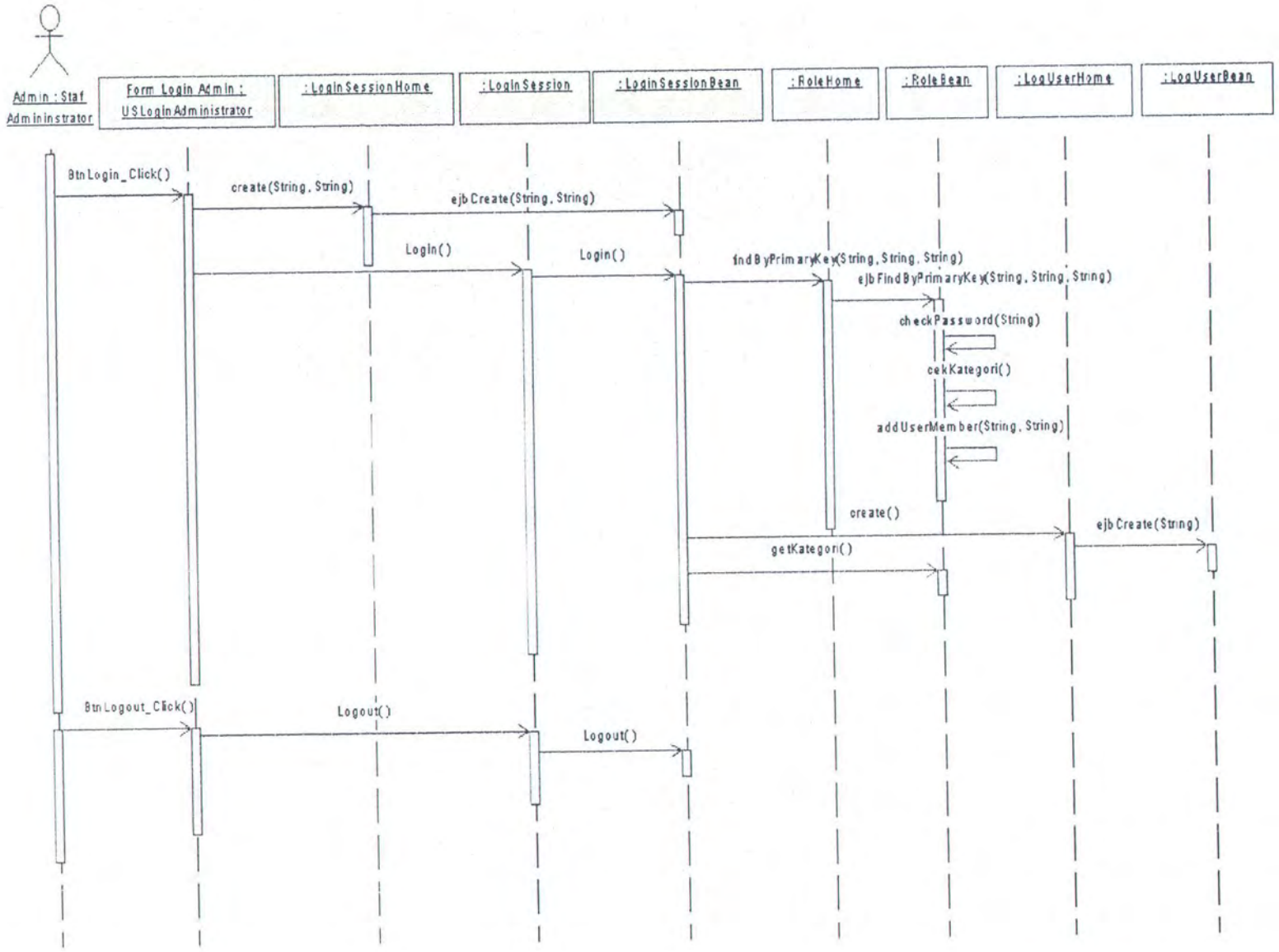
Apabila proses diatas sukses maka data-data login tersebut disimpan pada obyek LogUser. Skenario diakhiri dengan pemanggilan method BtnLogout_Click() yang memanggil remote method Logout() di obyek LoginSession.

Sequence diagram dan collaboration diagram yang mendeskripsikan skenario diatas terlihat pada gambar 3.3 dan gambar 3.4.

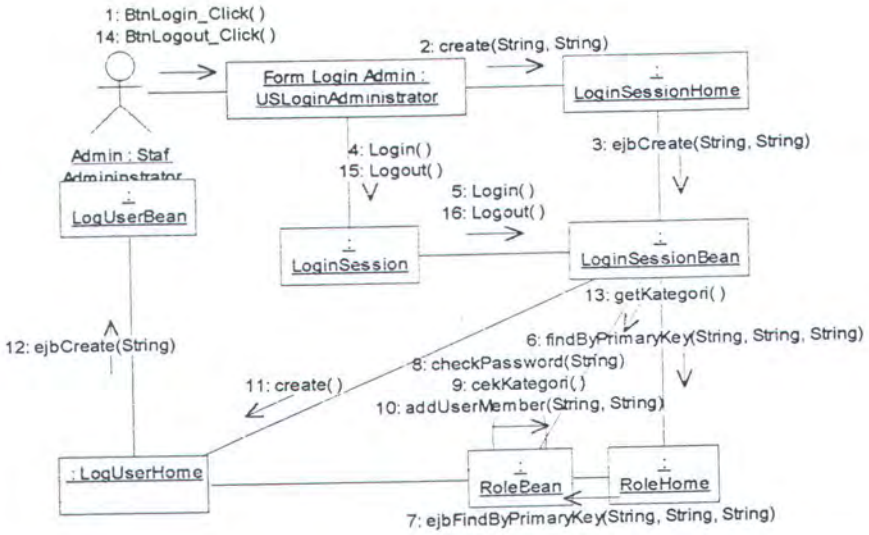
3.4.2.1.2 Skenario Login Nasabah

Login nasabah dimulai oleh aktor nasabah bank yang memanggil method BtnLogin_Click() di class USLoginNasabah. Class ini kemudian membuat obyek LoginSession. Setelah obyek LoginSession tercipta, USLoginNasabah dapat memanggil method Login(). Dalam mengotentikasi dan mengotorisasi user, LoginSession membuat obyek Role. Obyek Role ini yang melakukan pengecekan password dan penambahan user kedalam role nasabah. Apabila proses ini sukses, maka data-data disimpan kedalam obyek LogUser. Skenario diakhiri ketika nasabah memanggil method Logout() pada obyek LoginSession.

Sequence diagram dan collaboration diagram dari skenario login nasabah masing-masing digambarkan pada gambar 3.5 dan gambar 3.6.



Gambar 3.3 Sequence diagram login administrator.



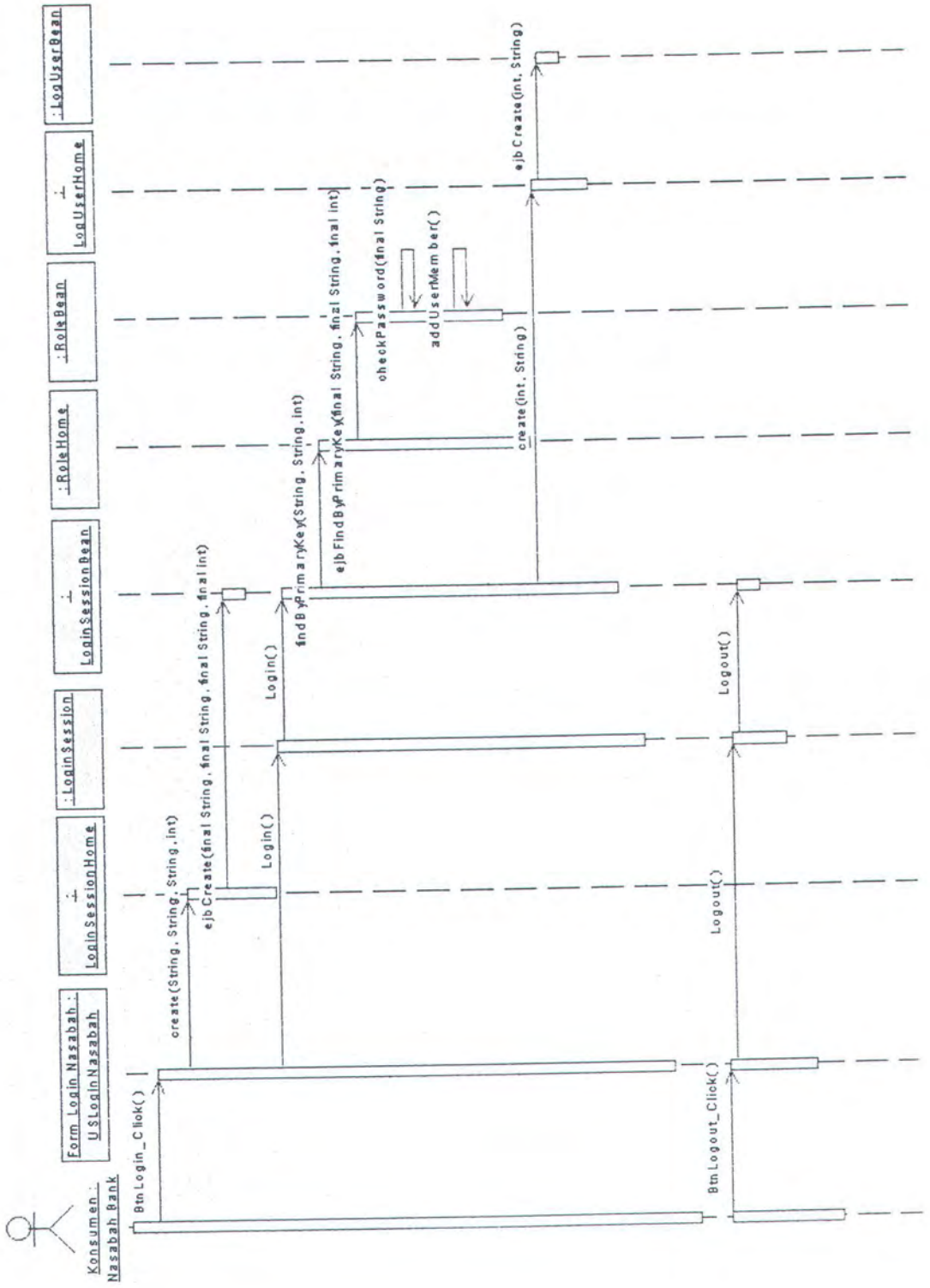
Gambar 3.4 Collaboration diagram login administrator.

3.4.2.2 Use Case Pembayaran Tagihan

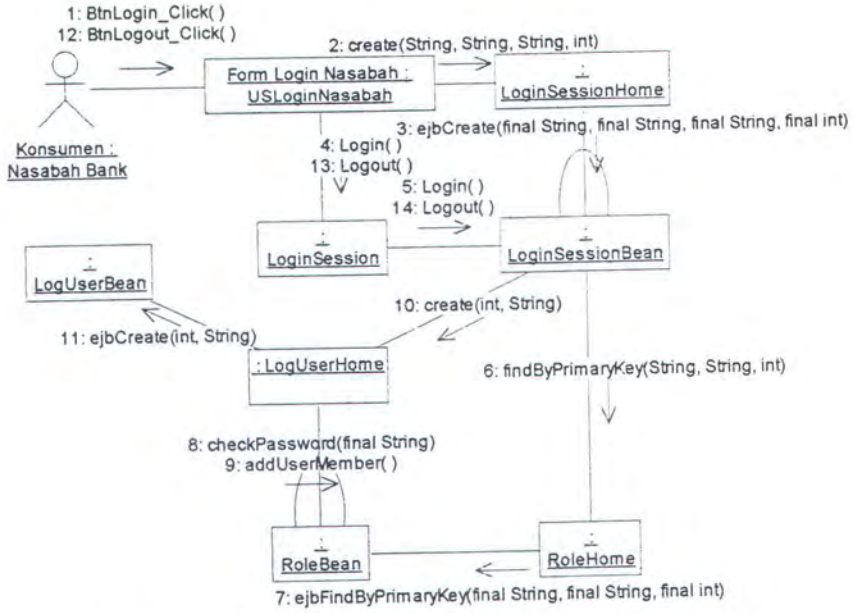
Dalam use case Pembayaran Tagihan, terdapat skenario proses bayar tagihan yang dilakukan aktor nasabah bank melalui web browser yang mengakses halaman web yang disediakan oleh penyelenggara ecommerce.

3.4.2.2.1 Skenario Pembayaran Tagihan

Dalam pembayaran tagihan, aktor memulai proses dengan memanggil method BtnProsesBayarClick() yang ada di class USPembayaranTagihan. Secara Otomatis class USPembayaranTagihan membuat memanggil obyek Nasabah supaya dapat memanggil method Bayar() untuk membayar tagihan. Jika proses tersebut sukses maka data-data transaksi disimpan dalam obyek Transaksi sebaliknya bila mengalami kegagalan maka simpan dalam obyek LogError.



Gambar 3.5 Sequence diagram login nasabah.



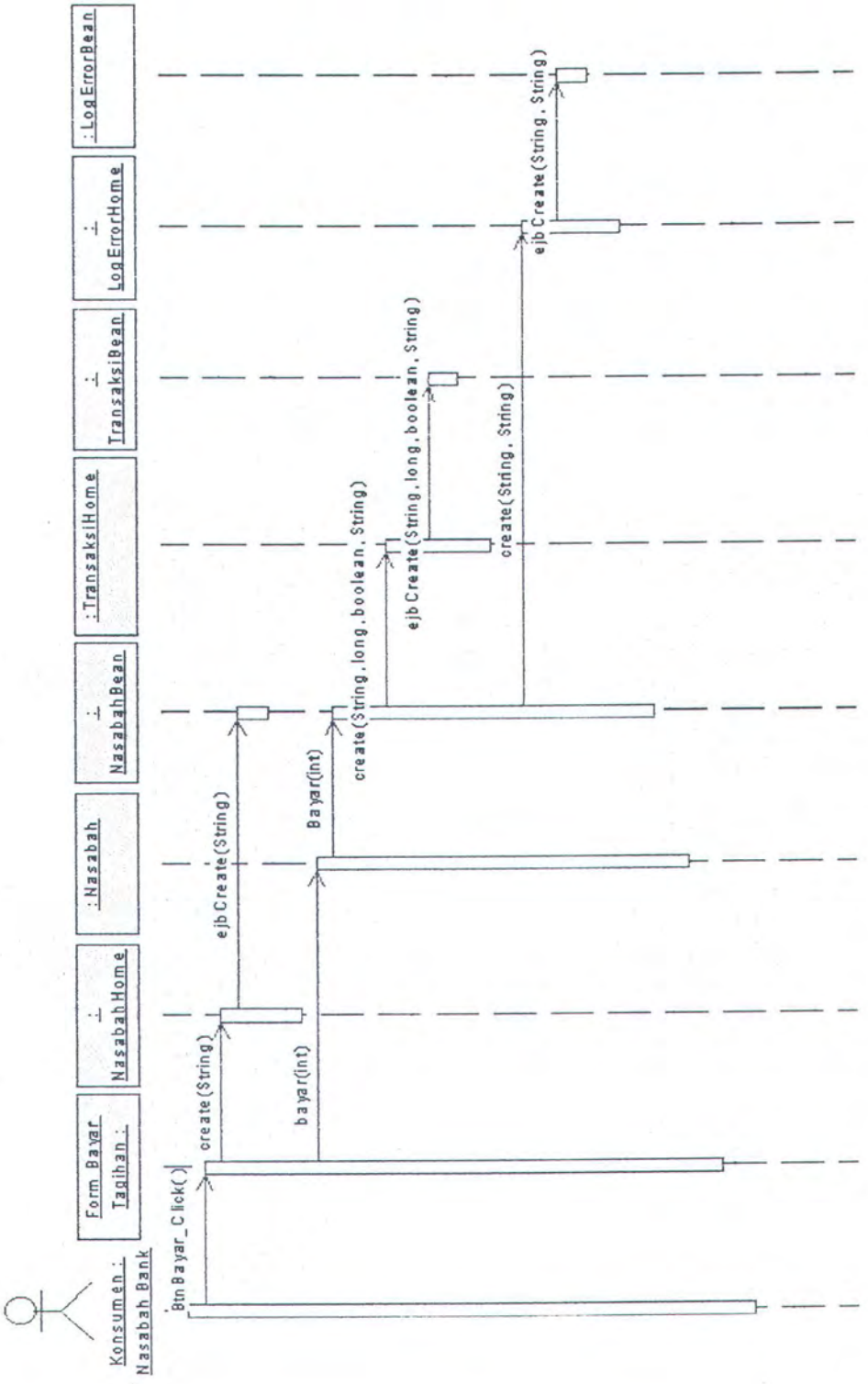
Gambar 3.6 Collaboration diagram login nasabah

3.4.2.2.2 Kondisi Persyaratan

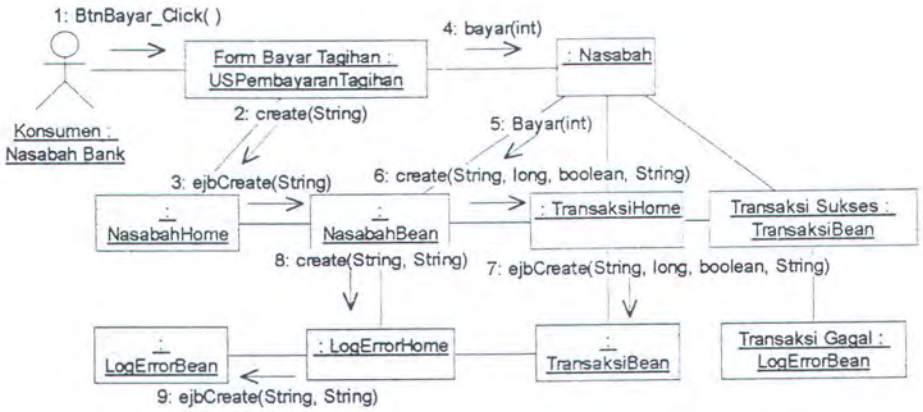
Pada use case pembayaran harus terpenuhi otentikasi dan otorisasi nomer rekening nasabah seperti yang digambarkan pada diagram login nasabah diatas.

3.4.2.3 Use Case Administrasi

Data-data mengenai staf administrasi dan ecommerce digunakan dalam melakukan pengawasan operasional sistem pembayaran online. Staf administrasi dibagi menjadi tiga kategori yaitu root, staf bank, dan staf ecommerce. Khusus untuk use case administrasi hanya dapat diakses oleh aktor root dan staf bank. Terdapat tiga skenario dalam use case ini yaitu edit data administrator, edit data ecommerce, dan pencarian data login user.



Gambar 3.7 Sequence diagram pembayaran tagihan

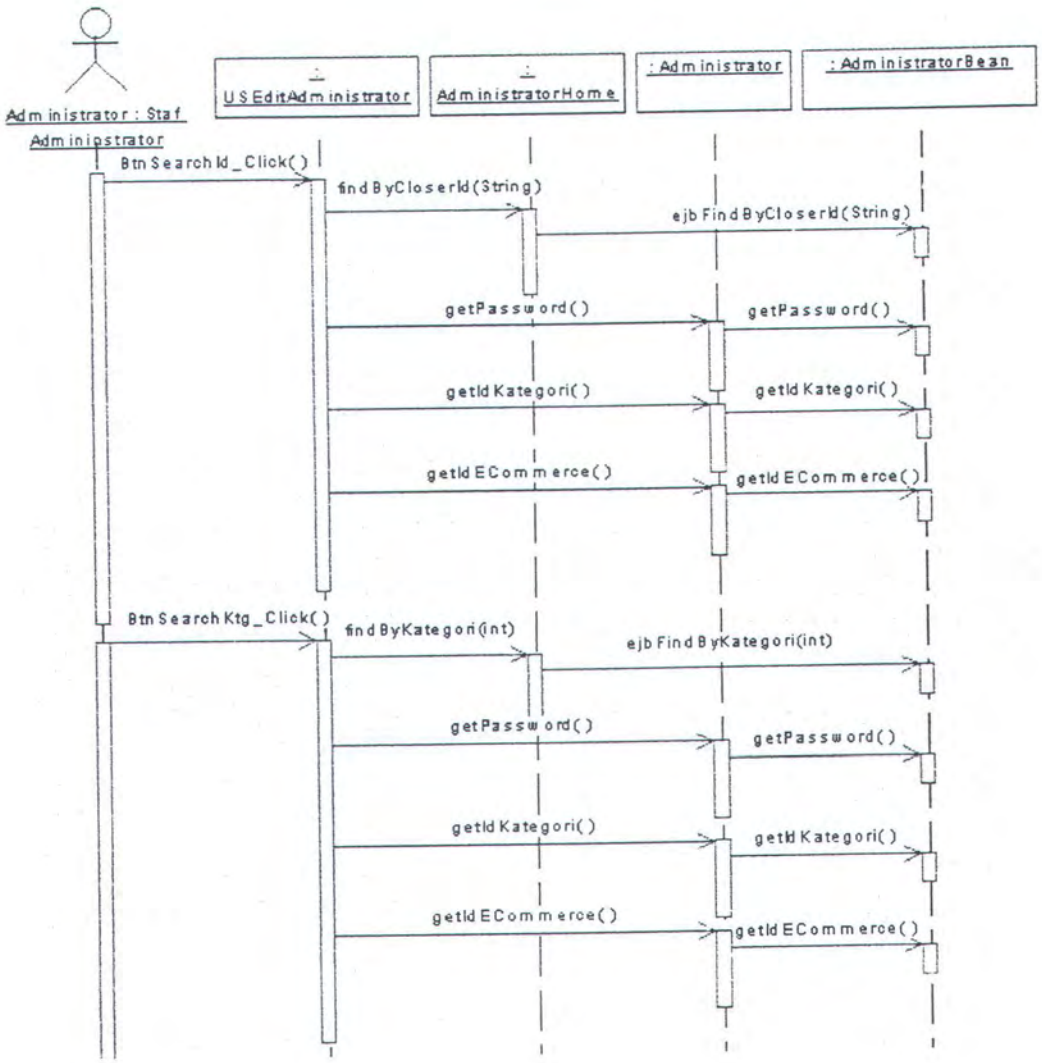


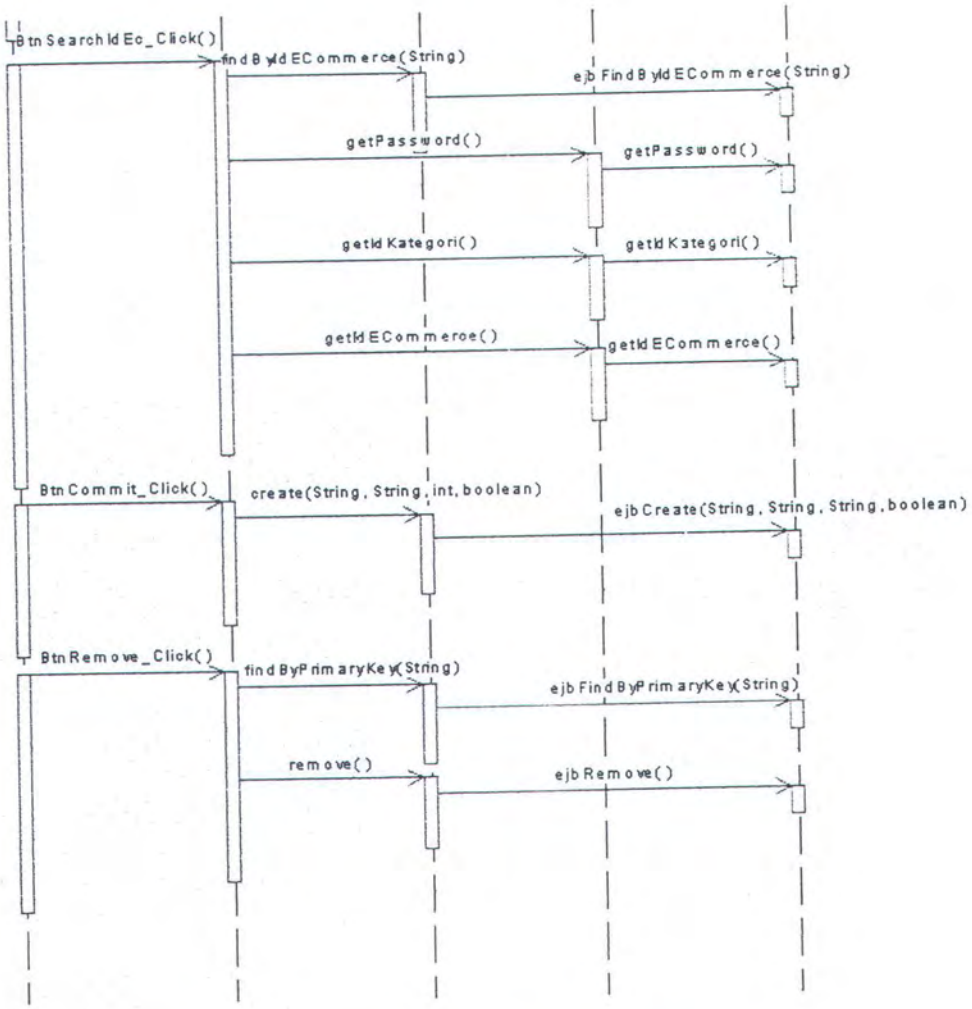
Gambar 3.8 Collaboration diagram pembayaran tagihan

3.4.2.3.1 Skenario Edit Data Administrator

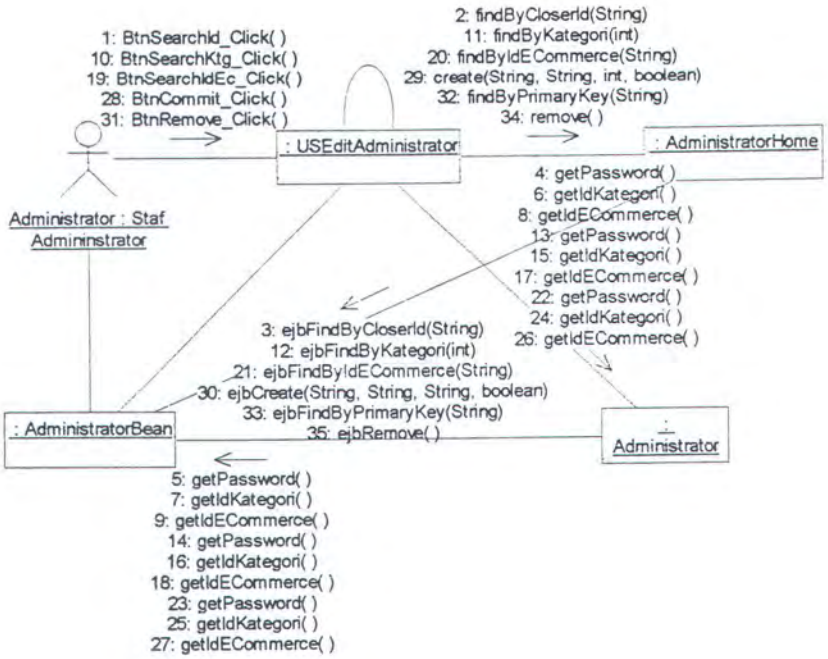
Skenario ini diawali oleh aktor staf administrator yang membuka halaman USEditAdministrator. Sebelum data dicari, kategori administrator yang berhak mengakses sistem disimpan dalam variabel di obyek Adminitrator. Kategori diperoleh melalui obyek Role.

Untuk pencarian data, aktor staf administrator memanggil method BtnSearchRec_Click() yang diteruskan dengan pemanggilan findByPrimaryKey() berdasarkan kategorinya. Apabila data yang diisikan harus disimpan maka method create dipanggil dengan parameter nrs (id), password, kategori, dan statusAktif. Sequence diagram dan collaboration diagram skenario ini terlihat pada gambar 3.9 dan gambar 3.10.


Administrator : Staf Administrator
USEditAdministrator
AdministratorHome
: Administrator
: AdministratorBean
BtnSearchId_Click()
findByCloserId(String)
ejbFindByCloserId(String)
getPassword()
getPassword()
getIdKategori()
getIdKategori()
getIdECommerce()
getIdECommerce()
BtnSearchKtg_Click()
findByKategori(int)
ejbFindByKategori(int)
getPassword()
getPassword()
getIdKategori()
getIdKategori()
getIdECommerce()
getIdECommerce()



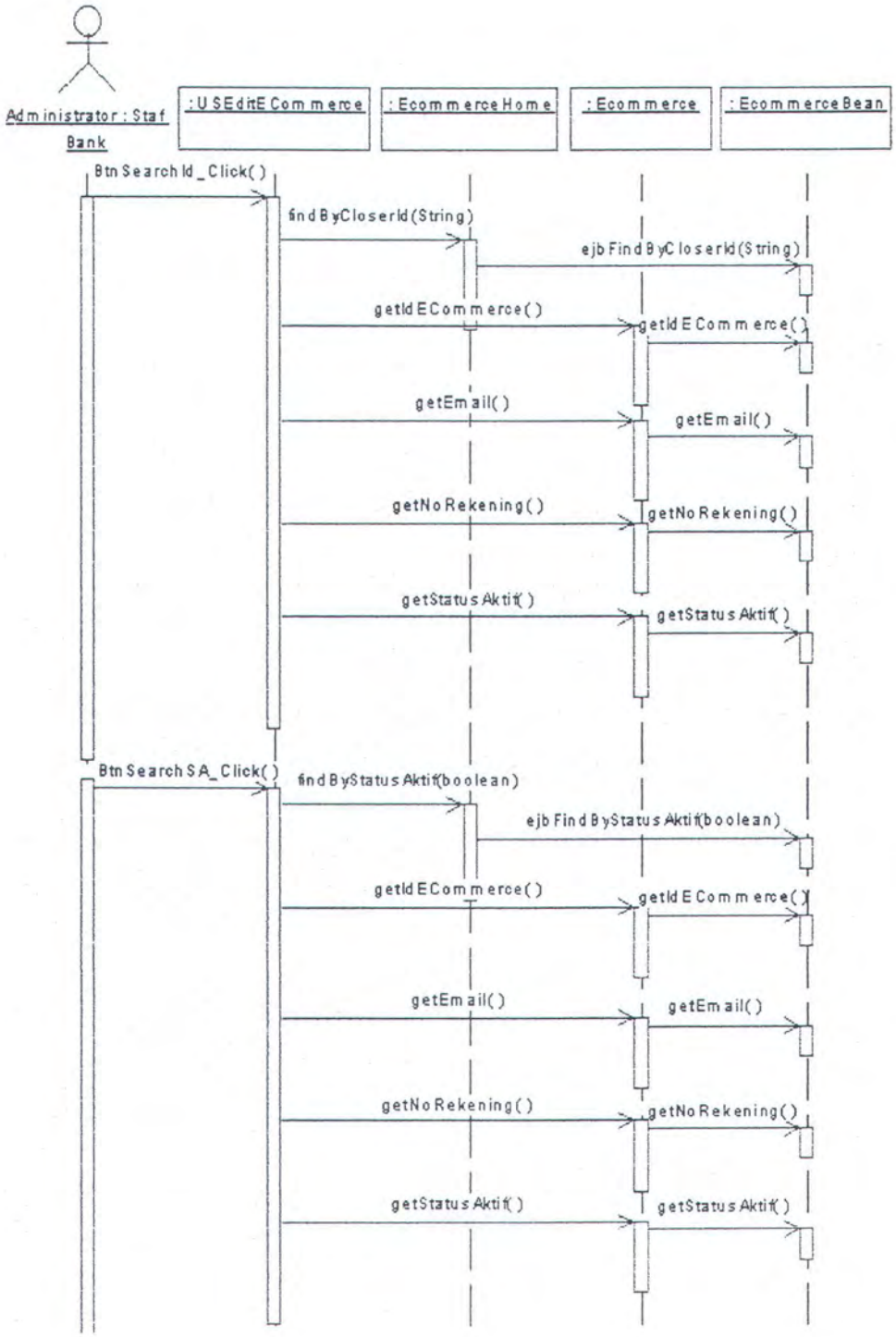
Gambar 3.9 Sequence diagram edit data administrator.

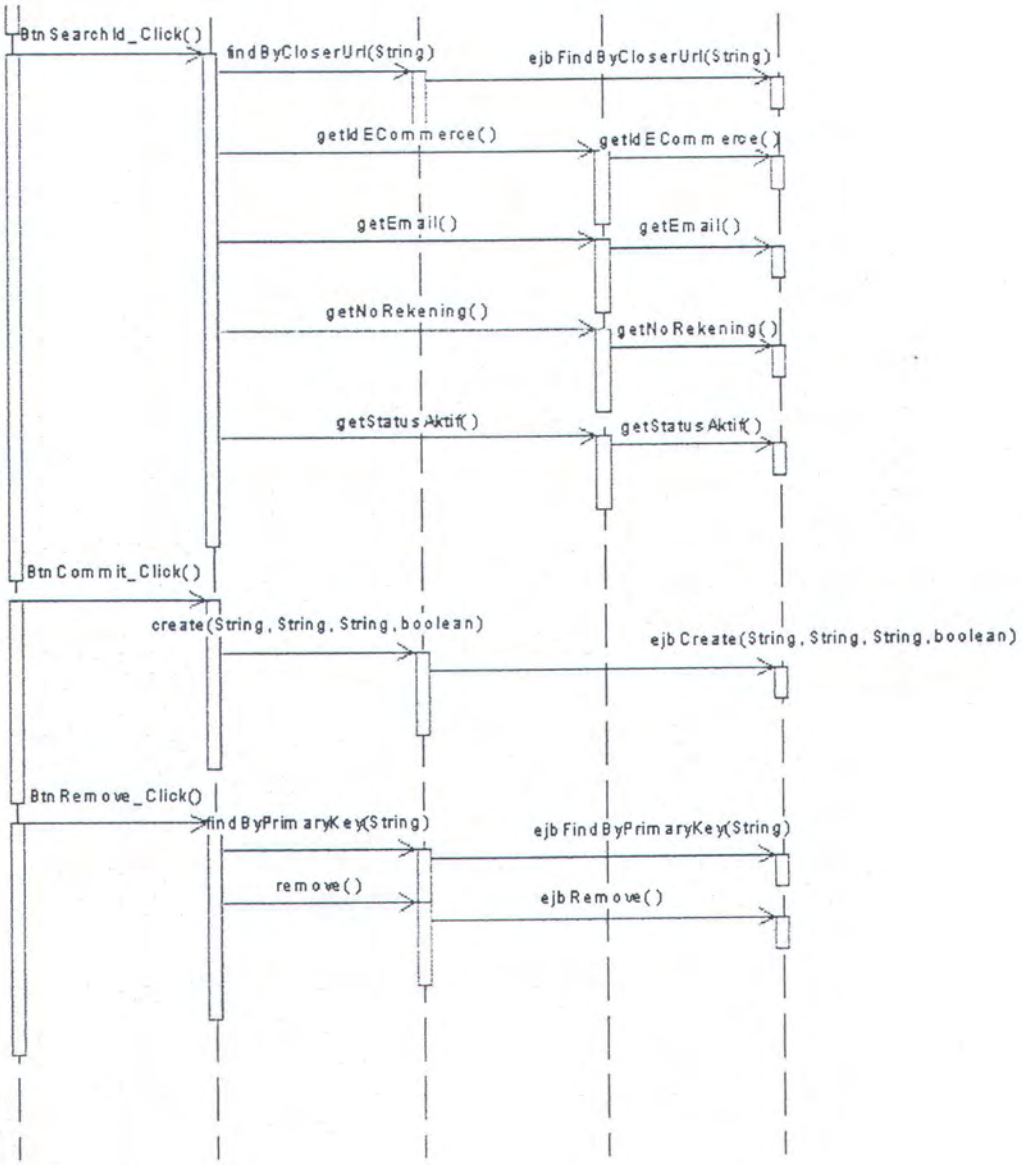


Gambar 3.10 Collaboration diagram edit data administrator.

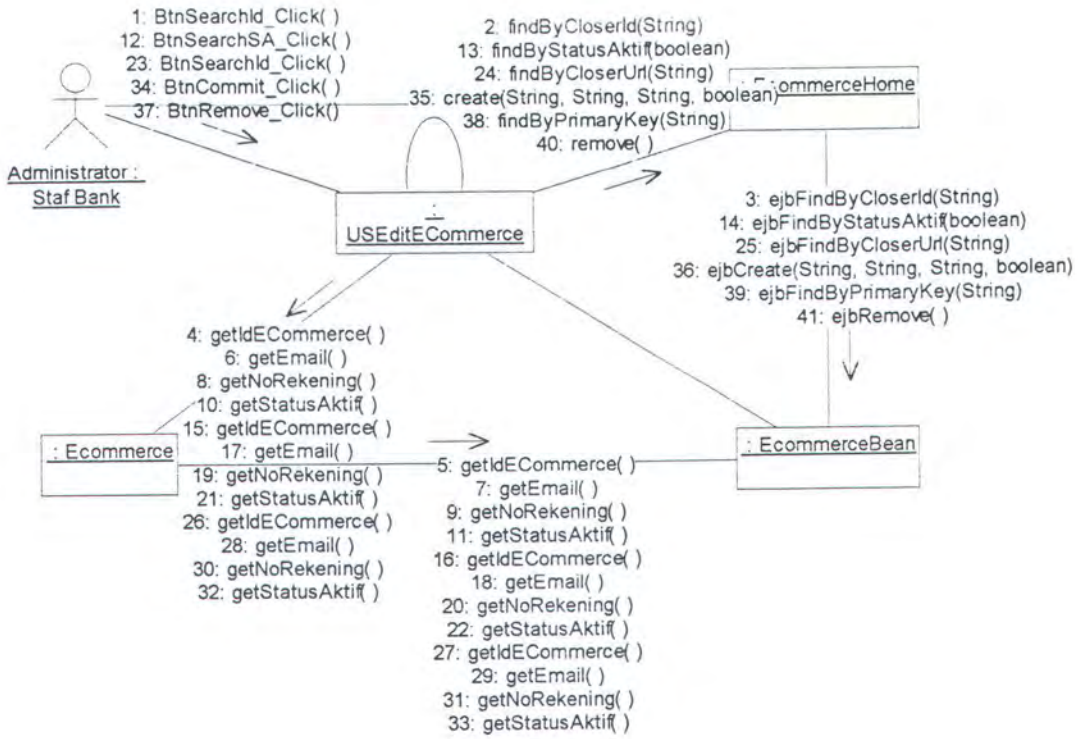
3.4.2.3.2 Skenario Edit Data Ecommerce

Skenario ini dimulai oleh aktor root atau staf bank yang membuka form USEditECommerce yang dilanjutkan dengan mengecek kategori user melalui obyek Role. Setelah data-data tersebut valid maka keseluruhan ecommerce diambil untuk ditampilkan. Apabila staf bank ingin melakukan pencarian data maka method BtnSearch_Click() dipanggil. Method ini mencari obyek ecommerce dengan primary key idECommerce. Apabila data yang diinputkan harus disampan maka dipanggil method BtnCommit_Click() yang kemudian create() obyek ecommerce baru dengan parameter idECommerce, url, email, dan statusAktif. Sequence diagram dan collaboration diagram skenario ini terlihat pada gambar 3.11 dan gambar 3.12.


Administrator : Staf Bank
: USEditECommerce
: EcommerceHome
: Ecommerce
: EcommerceBean
BtnSearchId_Click()
findByCloserId(String)
ejbFindByCloserId(String)
getIdECommerce()
getIdECommerce()
getEmail()
getEmail()
getNoRekening()
getNoRekening()
getStatusAktif()
getStatusAktif()
BtnSearchSA_Click()
findByStatusAktif(boolean)
ejbFindByStatusAktif(boolean)
getIdECommerce()
getIdECommerce()
getEmail()
getEmail()
getNoRekening()
getNoRekening()
getStatusAktif()
getStatusAktif()



Gambar 3.11 Sequence diagram edit data ecommerce.



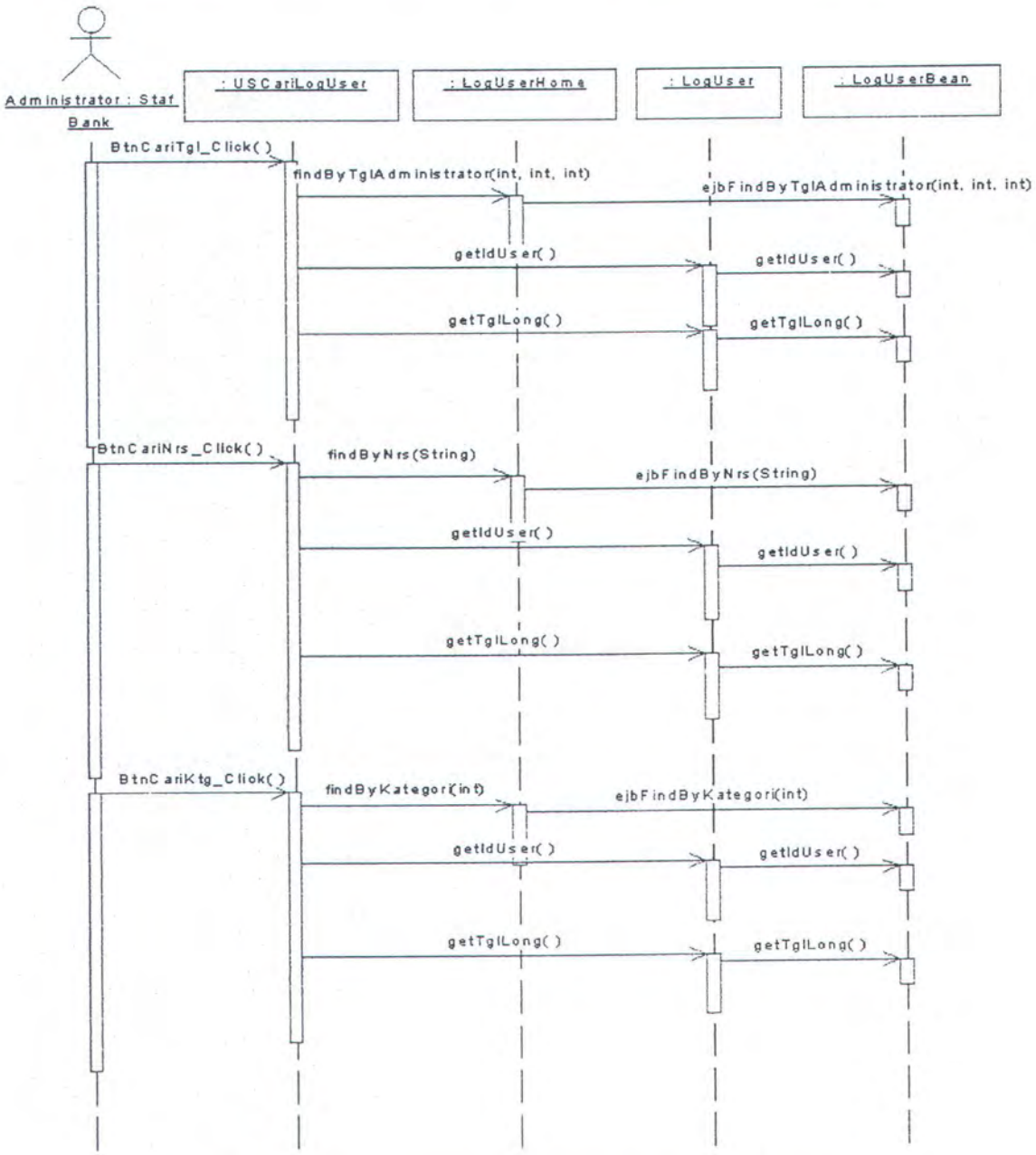
Gambar 3.12 Collaboration diagram edit data ecommerce.

3.4.2.3.3 Skenario Pencarian Data Login Administrator dan Login Nasabah

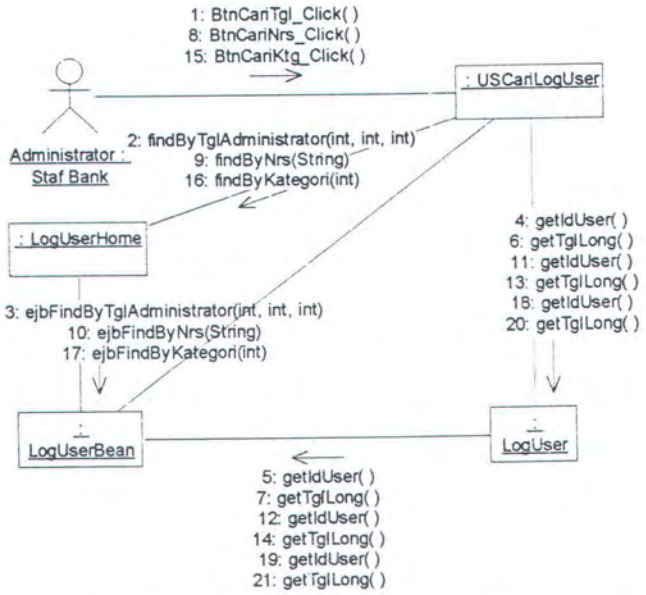
Informasi login administrator dan nasabah dapat digunakan untuk mengusut kapan saja user login kedalam sistem. Pencarian login administrator diawali oleh staf bank yang membuka form USCariLogUser yang diteruskan dengan mencari kategori user melalui obyek Role. Apabila kategori user termasuk root atau staf bank maka seluruh data pada obyek LogUser diambil. Pencarian dapat dilakukan berdasarkan dua parameter yaitu tanggal login dan id sebagai primary key user. Gambar 3.13, gambar 3.14, gambar 3.15, dan gambar 3.16 masing-masing menunjukan sequence diagram dan collaboration diagram dari skenario pencarian data login administrator serta sequence diagram dan collaboration diagram dari skenario pencarian data login nasabah.

3.4.2.3.4 Kondisi Persyaratan

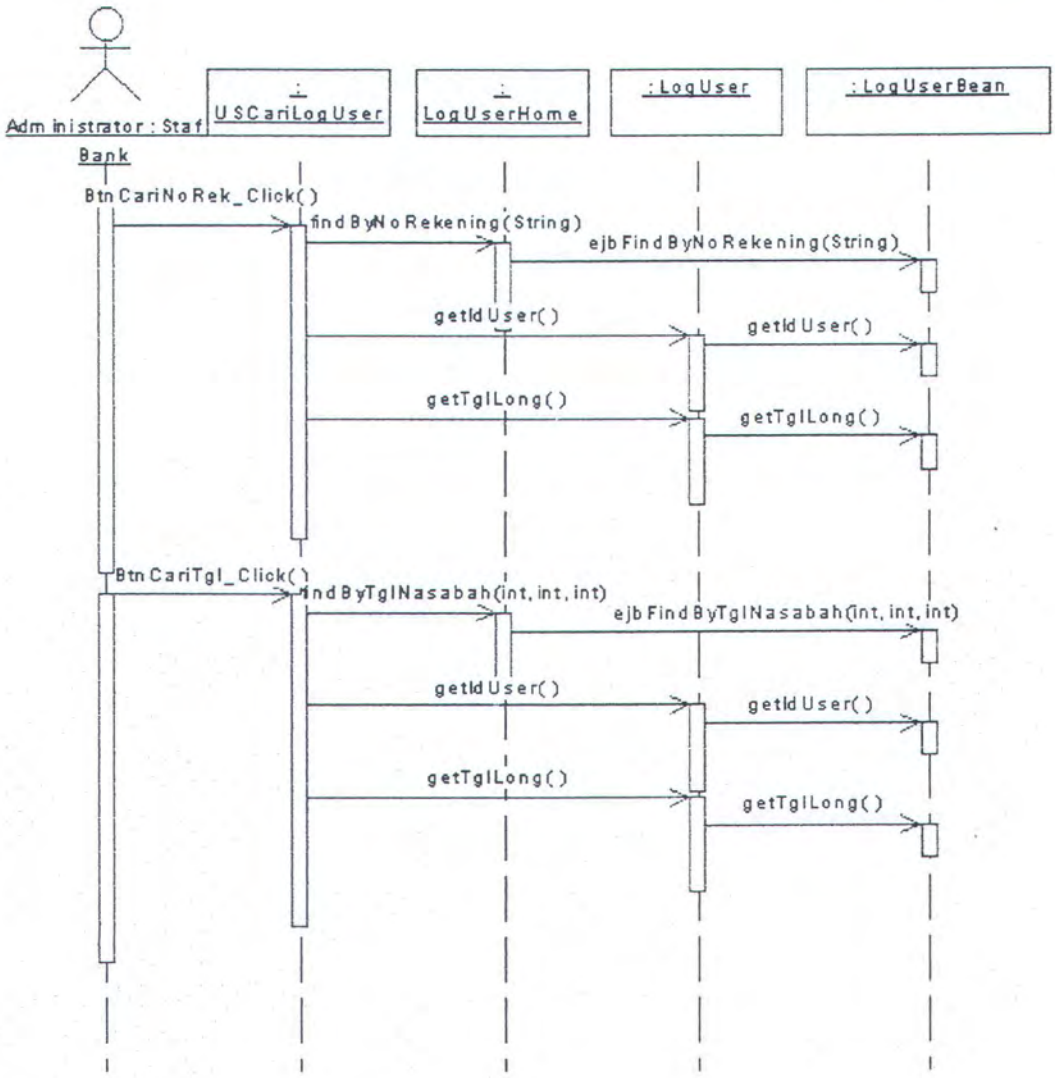
Pada use case administrasi tiap-tiap aktor staf administration yang terlibat harus memenuhi otentikasi dan otorisasi dengan cara login dahulu. Kategori staf administrator yang diberi hak akses dalam use case ini adalah root dan staf bank.



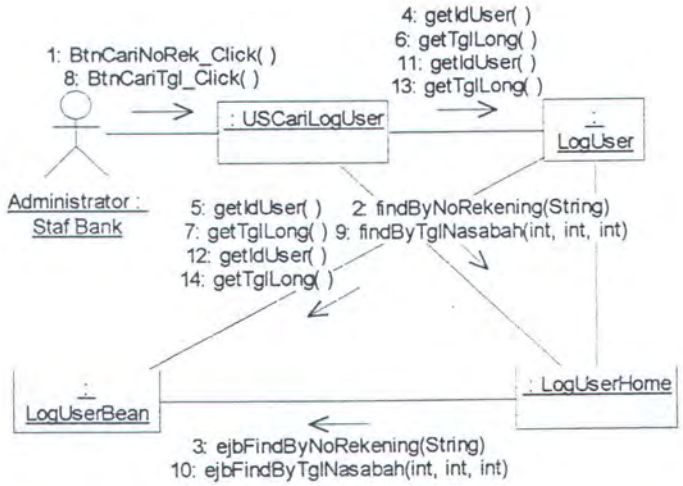
Gambar 3.13 Sequence diagram pencarian login administrator.



Gambar 3.14 Collaboration diagram pencarian login administrator.



Gambar 3.15 Sequence diagram pencarian login nasabah.



Gambar 3.16 Collaboration diagram pencarian login nasabah.

3.4.2.4 Use Case Pengawasan

Staf bank dan staf ecommerce bertugas untuk mengawasi pelaksanaan sistem pembayaran online yang meliputi pencarian transaksi nasabah dengan ecommerce, error-error yang terjadi, dan data-data login user. Dalam use case pengawasan ini terdapat dua skenario yaitu pencarian data transaksi dan pencarian data log error.

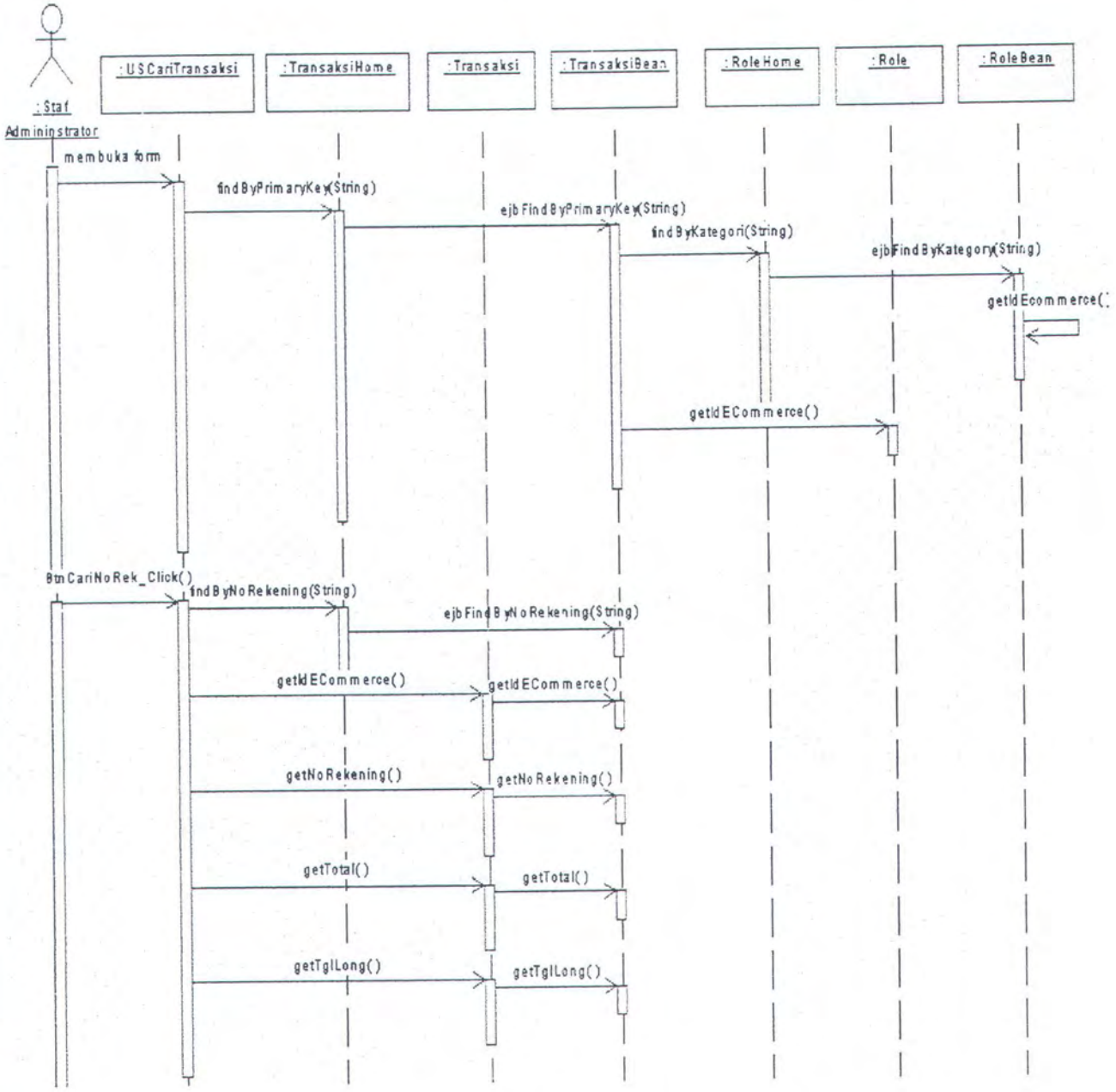
3.4.2.4.1 Skenario Pencarian Transaksi

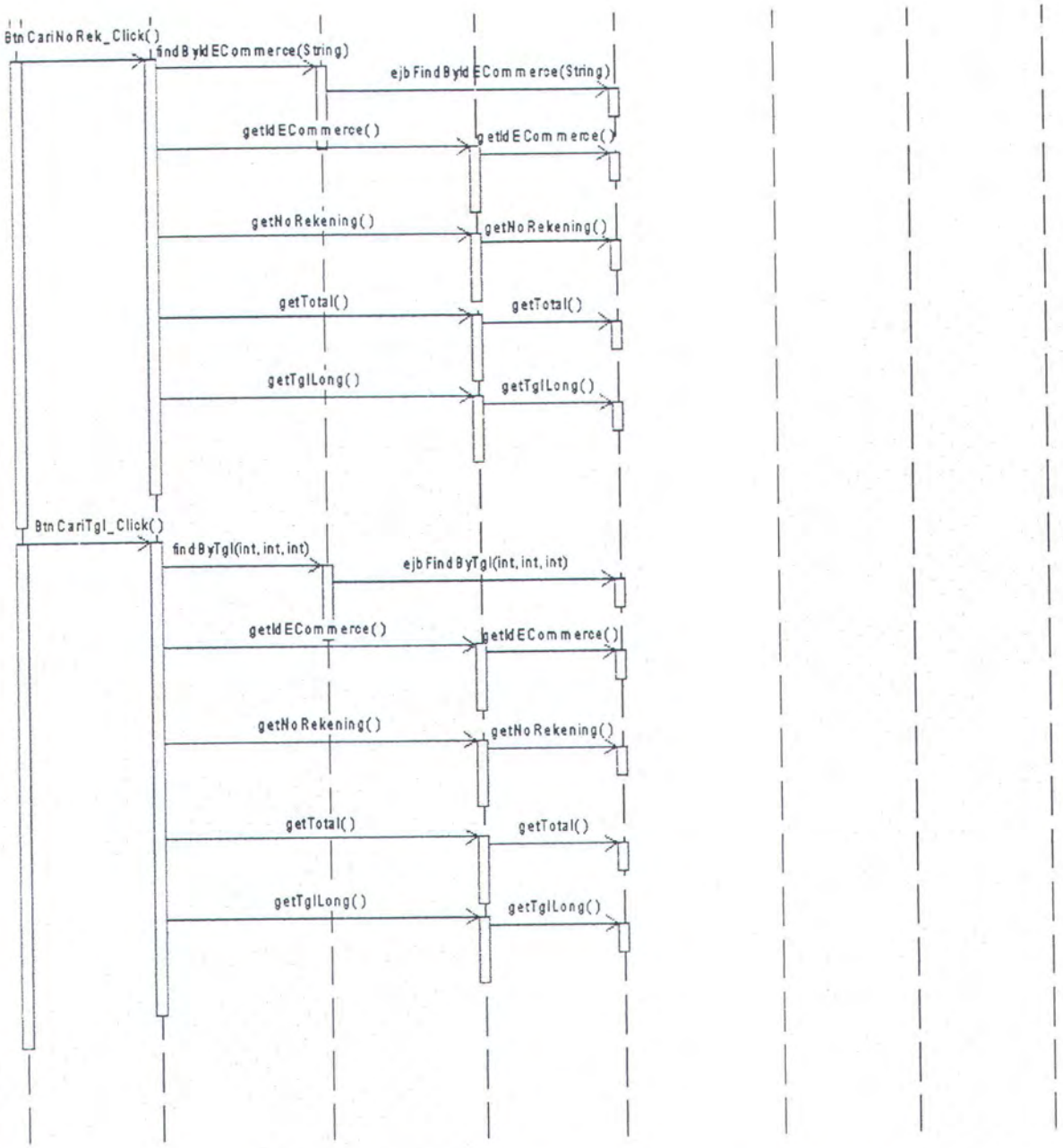
Data-data transaksi antara nasabah dan ecommerce disimpan dalam basis data agar dapat dicari untuk keperluan tertentu. Skenario ini dimulai oleh staf administrator yang membuka form USCariTransaksi yang secara otomatis mencari kategorinya pada obyek Role. Apabila termasuk kategori staf bank maka diambil keseluruhan transaksi tanpa mempedulikan idECommerce tertentu, tetapi bila kategorinya staf ecommerce maka diambil transaksi pada obyek Transaksi

dengan primary key idECommerce yang menjadi tanggungjawabnya. Terdapat dua jenis pencarian yaitu pencarian dengan parameter idECommerce dan noRekening serta pencarian berdasarkan tanggal transaksi. Sequence diagram dan collaboration diagram skenario ini terlihat pada gambar 3.15 dan 3.16.

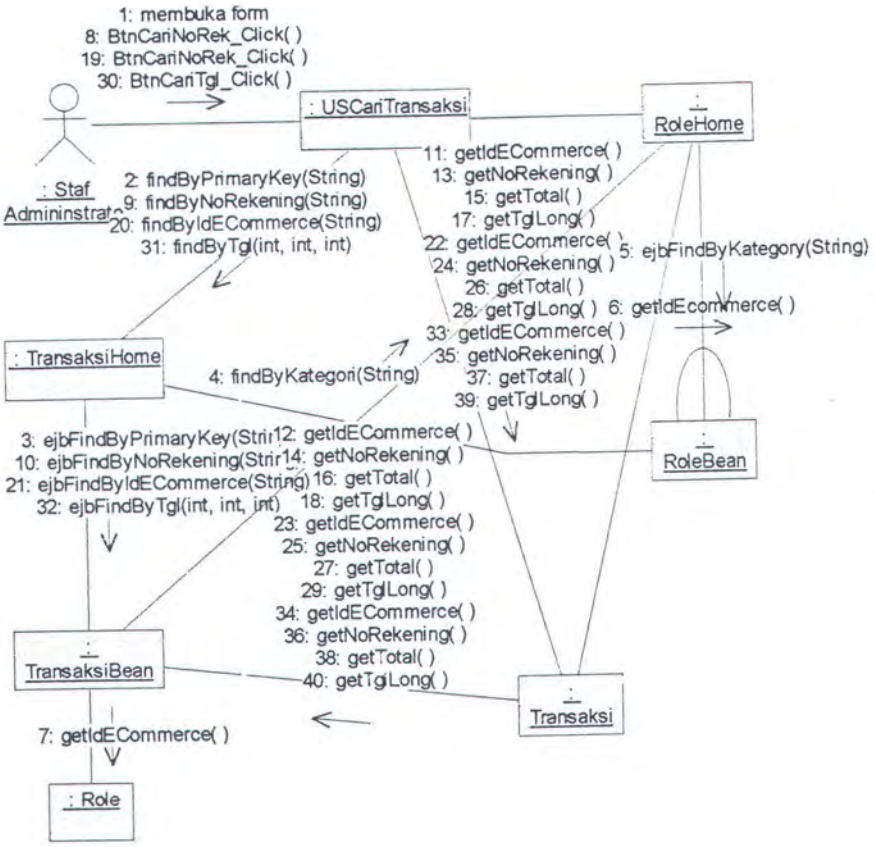
3.4.2.4.2 Skenario Pencarian Log Error

Transaksi antara nasabah dan ecommerce dapat mengalami kegagalan yang diakibatkan karena suatu kekeliruan yang disimpan dalam basis data agar dapat dicari kemudian. Skenario dimulai oleh aktor staf administrator yang membuka form USCariTransaksi yang secara otomatis mencari kategori user tersebut. Apabila kategori adalah staf bank maka diambil seluruh data dari obyek LogError tetapi apabila staf ecommerce diteruskan dengan pengambilan data dari obyek LogError berdasarkan idECommerce-nya. Terdapat dua jenis pencarian yaitu pencarian berdasarkan parameter idECommerce dan noRekening serta pencarian berdasarkan parameter tgl-nya. Sequence diagram dan collaboration diagram masing-masing ditampilkan pada gambar 3.17 dan gambar 3.18.


:Staf
Administrator
:USCariTransaksi
:TransaksiHome
:Transaksi
:TransaksiBean
:RoleHome
:Role
:RoleBean
membuka form
findByPrimaryKey(String)
ejbFindByPrimaryKey(String)
findByKategori(String)
ejbFindByKategori(String)
getIdEcommerce(
getIdECommerce()
BtnCariNoRek_Click()
findByNoRekening(String)
ejbFindByNoRekening(String)
getIdECommerce()
getIdECommerce()
getNoRekening()
getNoRekening()
getTotal()
getTotal()
getTglLong()
getTglLong()



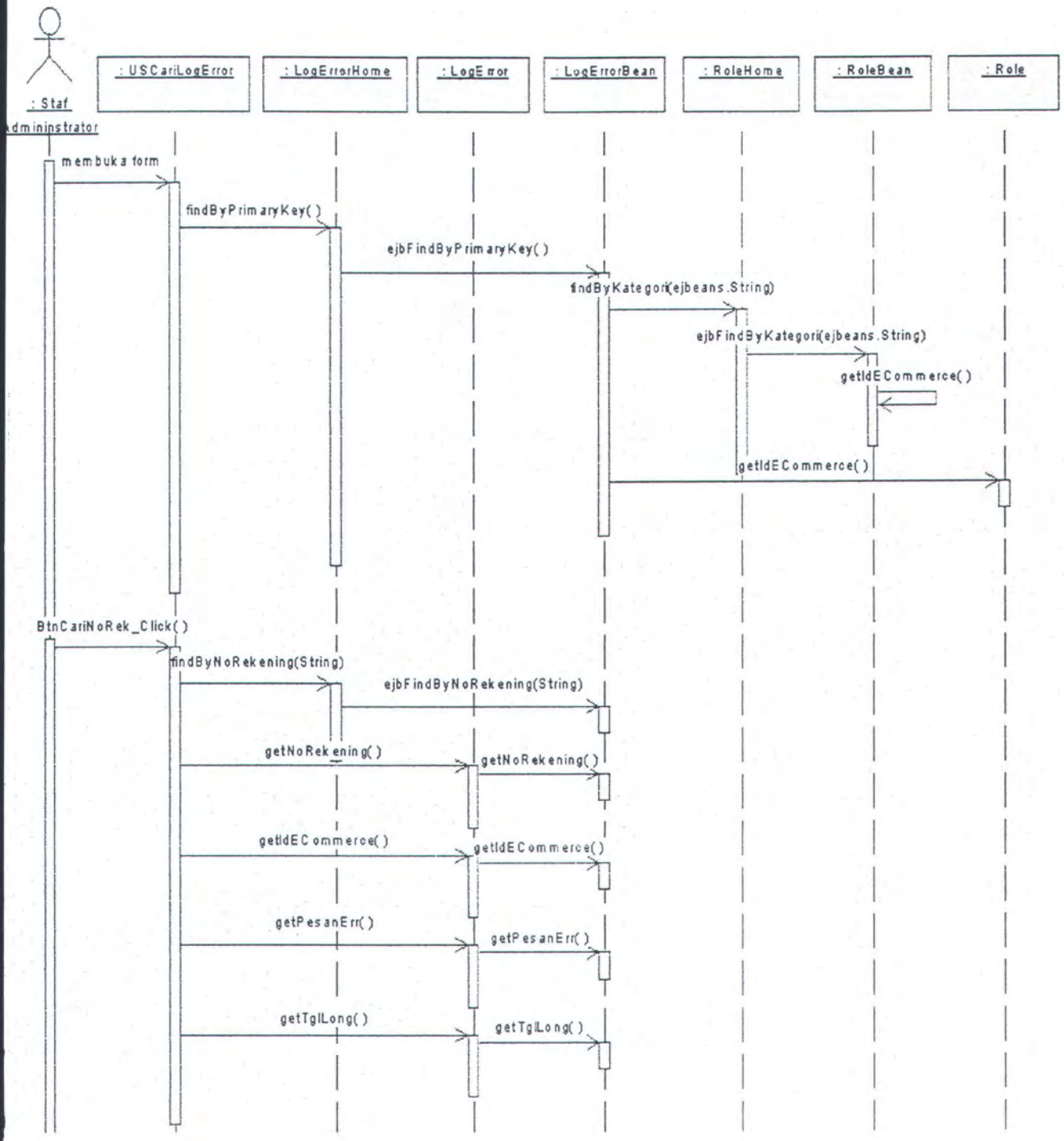
Gambar 3.15 Sequence diagram pencarian data transaksi.

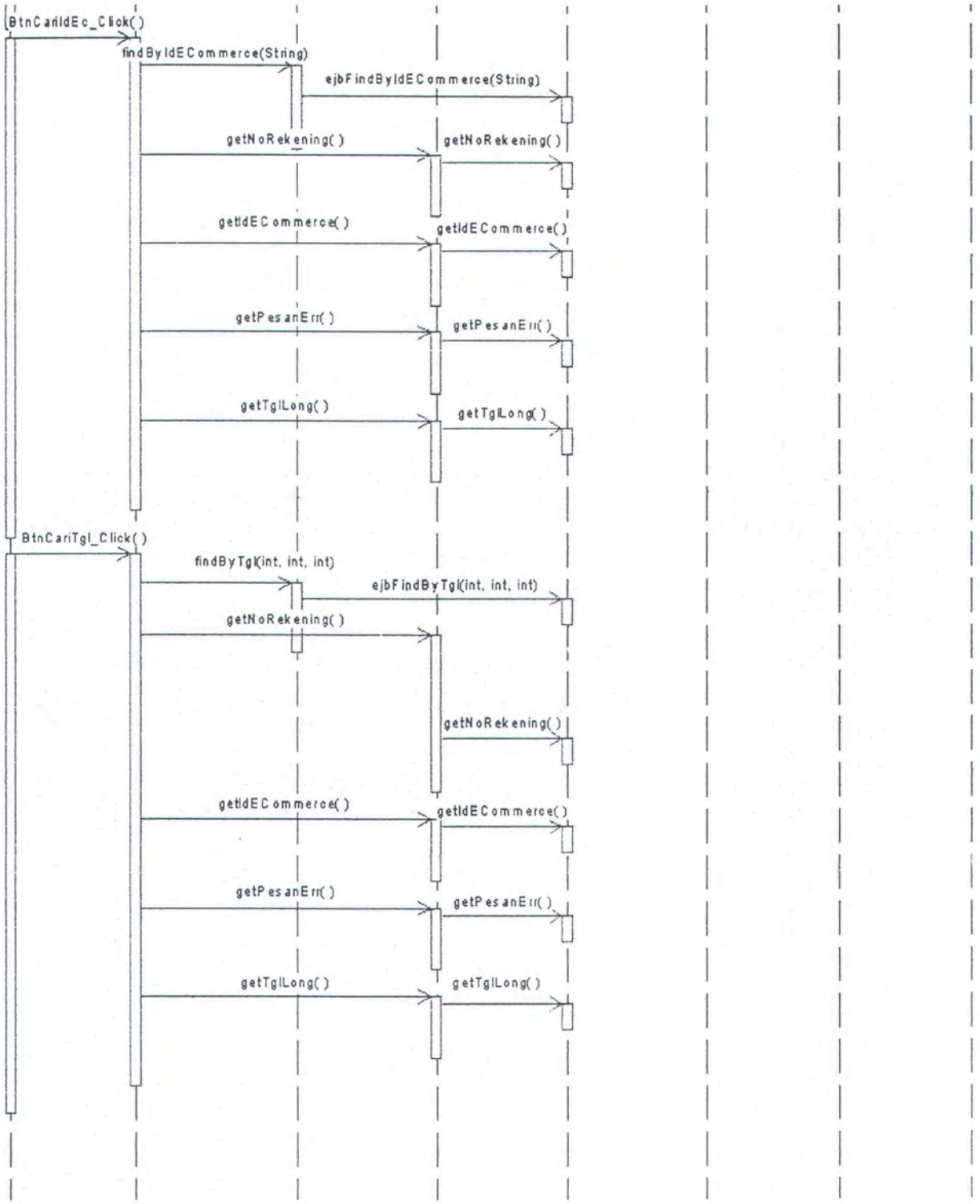


Gambar 3.16 Collaboration diagram pencarian data transaksi.

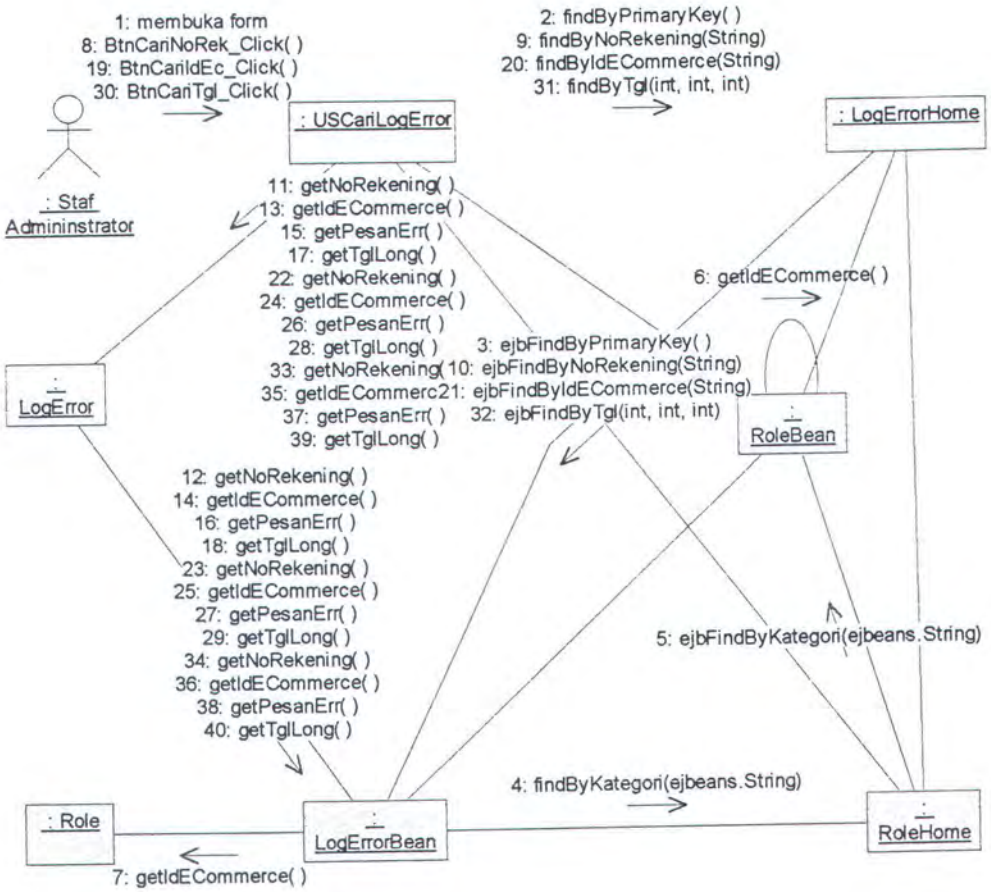
3.4.2.4.3 Kondisi Persyaratan

Pada use case pengawasan tiap-tiap aktor staf administrator yang terlibat harus memenuhi otentikasi dan otorisasi dengan cara login dahulu. Kategori staf administrator yang diberi hak untuk melakukan aktifitas dalam use case ini adalah staf bank dan staf ecommerce.


: Staf
dmininstrator
: USCariLogError
: LogErrorHome
: LogError
: LogErrorBean
: RoleHome
: RoleBean
: Role
membuka form
findByPrimaryKey( )
ejbFindByPrimaryKey( )
findByKategori(ejbeans.String)
ejbFindByKategori(ejbeans.String)
getIdECommerce( )
getIdECommerce( )
BtnCariNoRek_Click( )
findByNoRekening(String)
ejbFindByNoRekening(String)
getNoRekening( )
getNoRekening( )
getIdECommerce( )
getIdECommerce( )
getPesanErr( )
getPesanErr( )
getTglLong( )
getTglLong( )



Gambar 3.17 Sequence diagram pencarian data log error.



Gambar 3.18 Collaboration diagram pencarian data log error.

3.5 Perancangan Basis Data (Data service)

Dalam perancangan basis data sistem pembayaran online, objek yang terlibat didalam use case dan hubungan antar objek berpeluang untuk diimplementasikan kedalam basis data relasional. Perancangan basis data menggunakan class diagram dengan notasi UML untuk merepresentasikan objek dan hubungan antar objek.

Dalam perancangan basis data dalam sistem pembayaran online didapatkan objek–objek sebagai berikut:

- Nasabah (DSNasabah)

  Obyek nasabah merepresentasikan konsumen ecommerce. Antara konsumen dan ecommerce merupakan nasabah pada suatu bank yang sama. Obyek ini menyimpan informasi mengenai nasabah bank berupa nomer rekening, password, saldo, dan nama pemilik.

- Ecommerce (DSEcommerce)

  Obyek ecommerce merepresentasikan informasi mengenai ecommerce yang menjadi klien dari bank. Ecommerce menyimpan informasi mengenai id ecommerce, url, email, status aktif, dan nomer rekening bank.

- Staf Administrator (DSStafAdministrator)

  Obyek staf administrator merepresentasikan informasi mengenai user administrasi yang menyimpan data tentang nomer registrsi staf (nrs), password, kategori, dan status aktif.

- Transaksi (DSTransaksi)

  Obyek transaksi menyimpan informasi mengenai data-data transaksi antara nasabah dengan ecommerce. Data-data yang disimpan meliputi id transaksi (idTrans), tanggal transaksi (tgl), total tagihan (total), saldo awal nasabah, saldo akhir nasabah, saldo awal ecommerce, saldo akhir ecommerce, dan status transaksi (status).

- Log Error (DSLogError)

  Obyek log error menyimpan informasi mengenai transaksi-transaksi yang gagal karena suatu kekeliruan. Data-data yang disimpan meliputi id log error (idLogErr), pesan error (pesanErr), deskripsi error (descErr), dan tanggal kejadian (tgl).

- Login User (DSLogUser)

  Obyek login user menyimpan informasi mengenai aktifitas login user yang berhasil login sistem, berisi data-data yaitu id login user (idLogUser), tanggal kejadian (tgl), dan data identifikasi user.

- Login Nasabah (DSLoginNasabah)

  Obyek login nasabah mencatat data nasabah yang login. Merupakan obyek hasil relasi antara DSNasabah dan DSLogUser.

- Login Administrtor (DSLoginAdmin)

  Obyek Login adinistrator mencatat data administrator yang login. Merupakan obyek hasil relasi antara DSAdminitrator dan DSLogUser.

- Kategori Administrator (DSKategoriAdministrator)

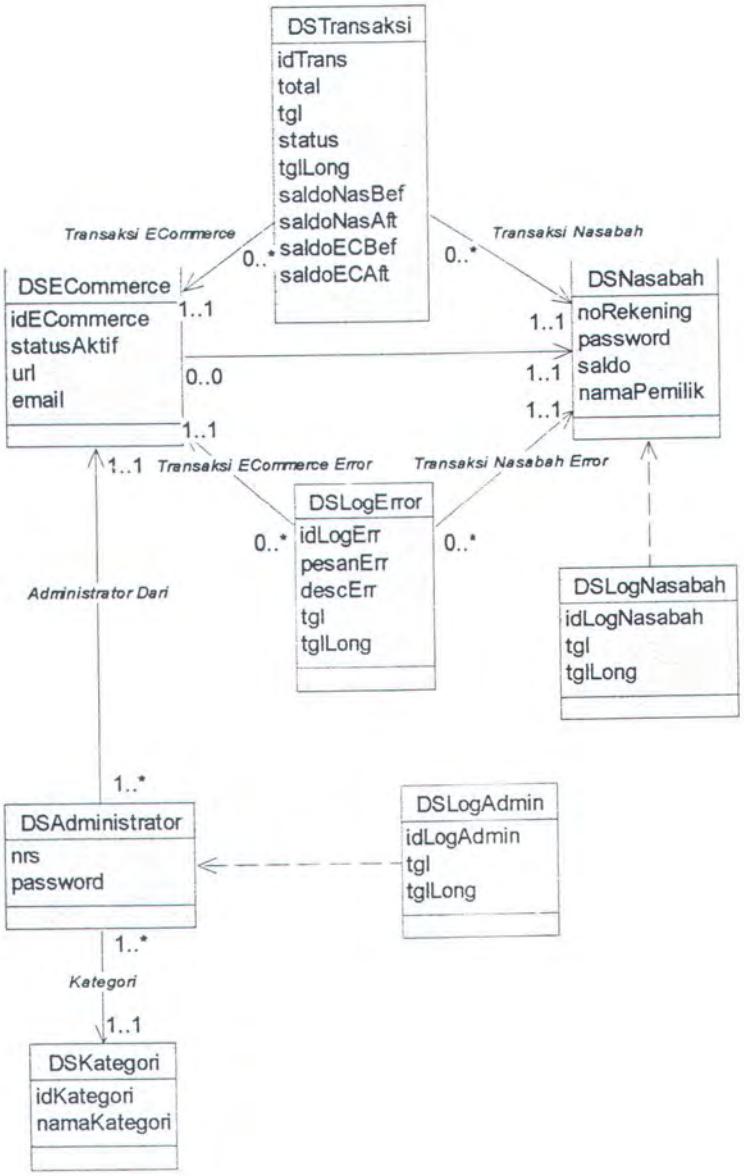
  Obyek kategori administrator berisi data kategori yang dimiliki administrator. Merupakan obyek hasil relasi antara DSAdminitrator dan DSKategori.

- Kategori (DSKategori)

  Obyek kategori menggolongkan kategori dari administrator yang dibagi menjadi tiga kategori yaitu root, staf bank, dan staf ecommerce. Kategori

ini berkaitan erat dengan pemberian *role* (hak akses sistem) kepada user administrator.

Hubungan antar objek dapat dilihat pada gambar 3-19, diagram ini digambarkan dalam class diagram sebagai suatu layer data service sebagai berikut



Gambar 3.19 Desain data service dalam class diagram basis data.

3.6 Perancangan Business Service dengan *Enterprise Java Bean*

Business Service memperlihatkan proses yang menjadi perantara antara user service dengan data service untuk melakukan proses bisnis yang diinginkan. Business service diimplementasikan menggunakan *Enterprise Java Bean* (EJB). EJB dibagi menjadi dua jenis yaitu entity bean dan session bean. Dalam sistem pembayaran online terdapat 1 class session bean dan 7 class entity bean.

Secara umum, class EJB harus mengimplementasikan antar muka *javax.ejb.EntityBean* untuk entity bean dan *javax.ejb.SessionBean* untuk session bean yang dapat dilihat pada Tabel 3.1 dan 3.2.

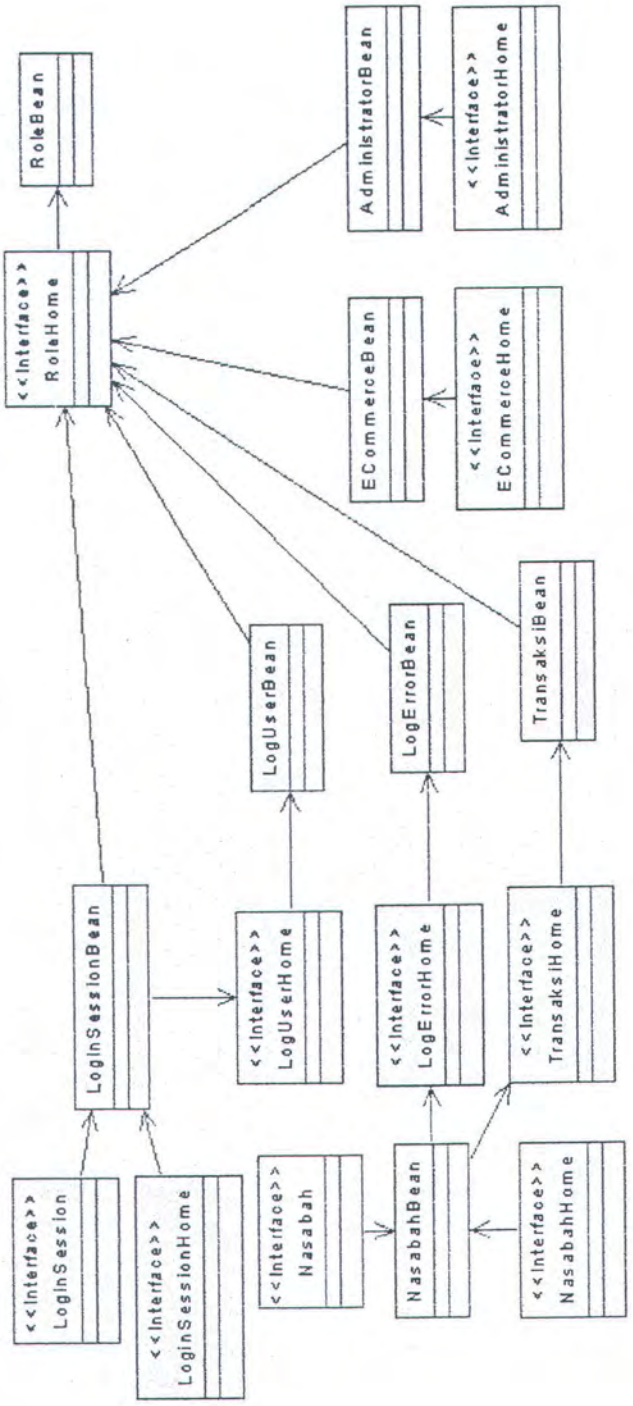
```
Public interface javax.ejb.EntityBean implements javax.ejb.EnterpriseBean {
        public abstract void setEntityContext(javax.ejb.EntityContext);
        public abstract void unsetEntityContext();
        public abstract void ejbRemove();
        public abstract void ejbActivate();
        public abstract void ejbPassivate();
        public abstract void ejbLoad();
        public abstract void ejbStore();
}
```

Tabel 3.1 Kode program interface javax.ejb.EntityBean

```
Public interface javax.ejb.EntityBean implements javax.ejb.EnterpriseBean {
        public abstract void setSessionContext(javax.ejb.EntityContext);
        public abstract void ejbRemove();
        public abstract void ejbActivate();
        public abstract void ejbPassivate();
}
```

Tabel 3.2 Kode program interface javax.ejb.SessionBean

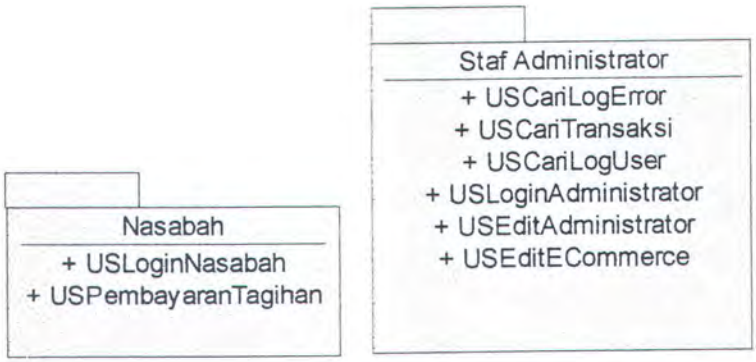
Didalam EJB terdapat class dan antar muka yaitu antar muka remote, antar muka home, class EJB, properties file masing-masing class serta deployment descriptor dan jar manifest file. Setiap obyek EJB terdiri dari 3 class yaitu Home Obyek, Remote Obyek, dan Enterprise Bean Class. Gambar 3.20 menunjukkan hubungan antara masing-masing class dan antar muka dari komponen EJB.



Gambar 3.20 Diagram relasi antara Home Obyek, Remote Obyek, dan Enterprise Bean Class yang membentuk komponen EJB.

3.7 Perancangan User Service

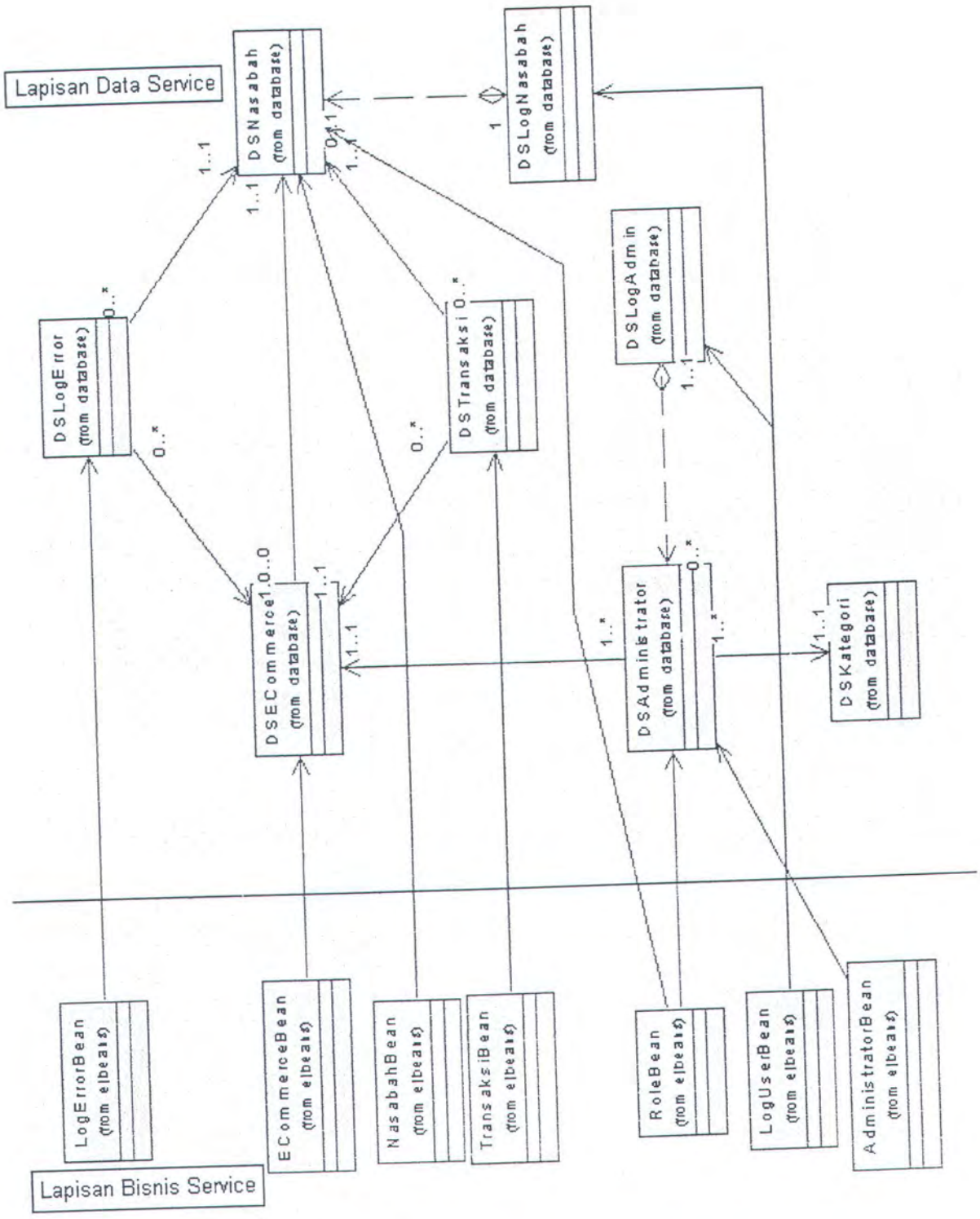
Lapisan user service merupakan bagian yang berinteraksi secara langsung dengan user (aktor) yang digolongkan menjadi dua kategori umum yaitu user service untuk nasabah dan staf administrator. Keseluruhan user service dibuat dengan menggunakan *Java Server Page (JSP)*. Agar terlihat masing-masing fungsinya, setiap obyek form pada lapisan user service ini dikelompokkan seperti yang terlihat pada gambar 3.21.



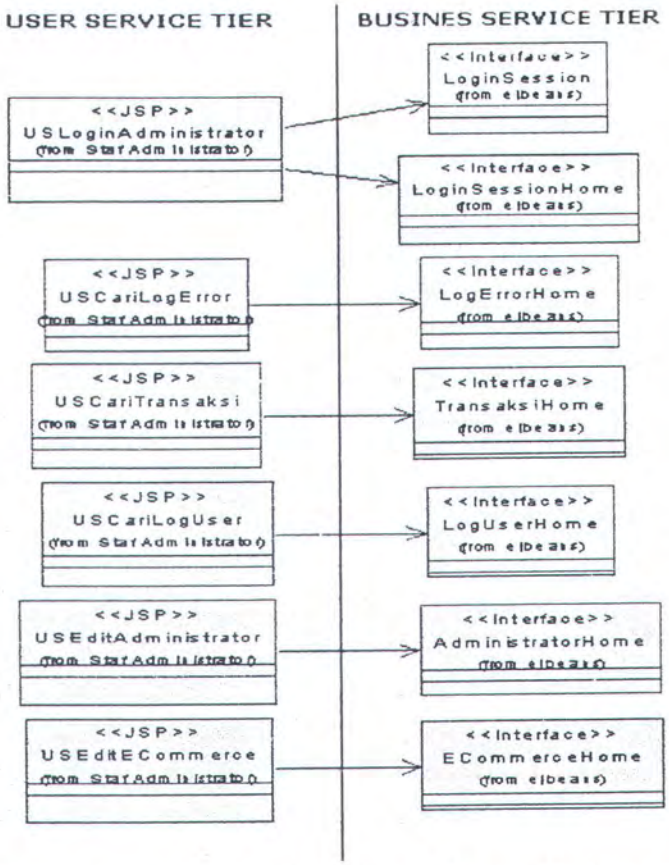
Gambar 3.21. Pengelompokan lapisan user service.

3.8 Perancangan Arsitektur Diagram 3-Tier

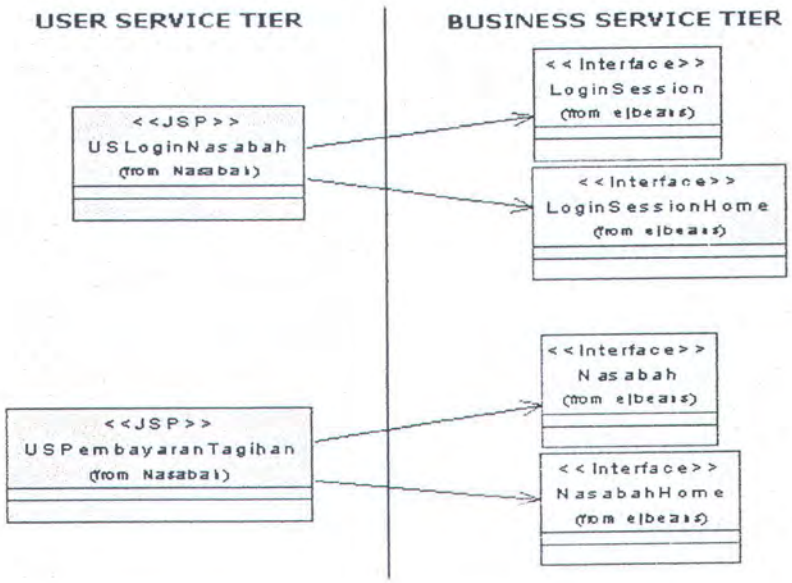
Gambar 3.22 menunjukkan diagram 3-Tier yaitu interaksi antara lapisan business service dengan lapisan data service, sementara hubungan user service dengan business service untuk administrator dan nasabah masing-masing ditunjukkan dalam gambar 3.23 dan gambar 3.24. Pembagian ini diakibatkan terlalu banyaknya relasi antara lapisan tersebut.



Gambar 3.22 Diagram relasi business lapisan service dengan lapisan data service.



Gambar 3.23 Diagram relasi lapisan servis user dengan lapisan servis bisnis administrasi.



Gambar 3.24 Diagram relasi lapisan servis user dengan lapisan servis bisnis nasabah.

Pada gambar 3.22, EJB class melaksanakan tugasnya dalam mengakses basis data didata service. Relasi antar *class* di business service sendiri telah dideskripsikan pada gambar 3.20 yaitu diagram relasi antara Home Obyek, Remote Obyek, dan *Enterprise Bean class* yang membentuk komponen EJB. Sedangkan pada gambar 3.23 dan 3.24, menunjukkan diagram relasi antara lapisan user service dengan lapisan business service berdasarkan kelompoknya masing-masing.

BAB IV

PEMBUATAN PROTOTYPE SISTEM

Perancangan sistem pembayaran online berdasarkan pendekatan UML dapat diimplementasikan dengan cara dibangkitkannya kode program dan skrip basis data dalam mempermudah mengaplikasikan proses bisnis dalam pemrograman. Dalam penerapannya, sistem ini menggunakan arsitektur sistem 3-tier yang mengintegrasikan komponen-komponen penyusunnya yaitu menyangkut susunan fisik, mesin, dan aplikasi perangkat lunak yang dibutuhkan agar sistem berjalan seperti yang diinginkan.

Perancangan berdasarkan pendekatan UML dengan *Rational Rose* dapat menghasilkan kerangka kode program untuk memudahkan pengembangannya. Kerangka ini berbentuk *class-class Enterprise Java Bean* dengan ekstensi java beserta *method-method*-nya sehingga dapat digunakan untuk mengembangkan proses bisnisnya menggunakan kode pemrograman Java. Dari skrip basis data yang dihasilkan dapat digunakan untuk membangkitkan obyek berdasarkan RDBMS Oracle.

4.1 Implementasi Skrip Basis Data

Implementasi desain data service dapat menghasilkan skrip basis data pada RDBMS tertentu yang telah disediakan *Data Definition Language (DDL)*-nya oleh *Rational Rose*. *Class-class* yang akan dibangkitkan dipilih kemudian dengan fungsi *generate* menghasilkan potongan skrip sebagai berikut

```
CREATE TABLE DSKategori(
  idKategori VARCHAR UNIQUE,
  namaKategori VARCHAR UNIQUE,
  PRIMARY KEY(idKategori),
  UNIQUE( idKategori,namaKategori));

CREATE TABLE DSTransaksi(
  idTrans int UNIQUE,
  total int NOT NULL,
  tgl datetime NOT NULL,
  status boolean NOT NULL,
  tglLong VARCHAR UNIQUE,
  idECommerce VARCHAR UNIQUE,
  FOREIGN KEY (idECommerce) REFERENCES DSECommerce,
  noRekening VARCHAR UNIQUE,
  FOREIGN KEY (noRekening) REFERENCES DSNasabah,
  PRIMARY KEY(idTrans),
  UNIQUE( idTrans,tglLong));

CREATE TABLE DSLogError(
  idLogErr VARCHAR UNIQUE,
  pesanErr VARCHAR UNIQUE,
  descErr VARCHAR UNIQUE,
  tgl VARCHAR UNIQUE,
  tglLong VARCHAR UNIQUE,
  idECommerce VARCHAR UNIQUE,
  FOREIGN KEY (idECommerce) REFERENCES DSECommerce,
  noRekening VARCHAR UNIQUE,
  FOREIGN KEY (noRekening) REFERENCES DSNasabah,
  PRIMARY KEY(idLogErr),
  UNIQUE( idLogErr,pesanErr,descErr,tgl,tglLong));

CREATE TABLE DSECommerce(
  idECommerce VARCHAR UNIQUE,
  statusAktif boolean NOT NULL,
  url VARCHAR NOT NULL,
  email VARCHAR NOT NULL,
  noRekening VARCHAR UNIQUE,
  FOREIGN KEY (noRekening) REFERENCES DSNasabah,
  PRIMARY KEY(idECommerce),
  UNIQUE( idECommerce));
```

4.2 Implementasi Komponen *Enterprise Java Bean*

4.2.1 Umum

Berdasarkan perancangan obyek *Enterprise Java Bean* (EJB) menggunakan *Rational Rose* dapat dibangkitkan kode program *class-class* EJB yang digunakan untuk mengimplementasikan proses bisnis obyek EJB. Skrip kode program yang

dihasilkan masing-masing terdiri dari 3 *obyek, obyek* tersebut adalah antar muka obyek home, antar muka obyek remote, dan obyek implementasi bean.

Potongan skrip salah satu obyek yaitu LoginSession adalah sebagai berikut

```
package ejbeans;
import java.rmi.*;
import java.util.*;
import javax.ejb.*;

public interface LoginSessionHome extends EJBHome {
  /**
    @roseuid 3B23B7AE0302
   */
  LoginSession create(String noRekening, String password, String idECommerce, int gol)
  throws CreateException,SecurityException,RemoteException;
  /**
    @roseuid 3B23B7AF0096
   */
  LoginSession create() throws CreateException,SecurityException,RemoteException;
}
```

Potongan skrip salah satu obyek remote interface LoginSession adalah sebagai berikut

```
package ejbeans;
import java.rmi.*;
import javax.ejb.*;

public interface LoginSession extends EJBObject {
  /**
    @roseuid 3B23B7AF01E0
   */
  public boolean Login() throws CreateException,SecurityException,RemoteException;
  /**
    @roseuid 3B23B7AF021C
   */
  public boolean Logout() throws CreateException,SecurityException,RemoteException;
  /**
    @roseuid 3B23B7AF024E
   */
  public String getIdECommerce()
throwCreateException,SecurityException,RemoteException;
  /**
    @roseuid 3B23B7AF024F
   */
  public int getKategori() throws CreateException,SecurityException,RemoteException;
}
```

Potongan skrip salah satu implementasi bean LoginSession adalah sebagai berikut

```
package ejbeans;
import java.rmi.*;
import java.security.*;
import java.util.*;
import javax.ejb.*;
import javax.naming.*;
import allaire.ejipt.*;
import java.rmi.server.*;
import ejbeans.*;

public class LoginSessionBean implements SessionBean {
  public int _kategori = 0;
  public int  gol = 0;
  private transient SessionContext _context;
  public RolePK rolepk = new RolePK ();
  public Role role;
  public String _noRekening = null;
  public String _password = null;
  private String _idECommerce = null;
  private String _idec = null;

  /**
    @roseuid 3B23B7A70384
   */
  public void setSessionContext(final SessionContext context) throws RemoteException {}
  /**
    @roseuid 3B23B7A8000A
   */
public void ejbCreate(final String noRekening, final String password, final String
idECommerce, final int gol) throws CreateException,SecurityException,RemoteException
{}
```

Obyek-obyek yang dibangkitkan oleh *Rational Rose* berupa kerangka (*template*) yang digunakan sebagai dasar pengembangan proses bisnis sistem yang sebenarnya.

4.2.2 Pemrograman Class EJB

Kerangka kode program yang dibangkitkan *Rational Rose* diatas dikembangkan dengan cara menambahkan kode permograman Java berdasarkan algoritma proses bisnis tertentu pada implementasi obyek bean. Berikut ini

pengembangan dari kerangka skrip untuk implementasi obyek LoginSession bean diatas.

```
public class LoginSessionBean implements SessionBean
{
  private transient SessionContext  context;
  public RolePK rolepk = new RolePK();
  public Role role;
  public String _noRekening = null;
  public String _password = null;
  private String _idECommerce = null;
  public int _kategori = 0;
  public int _gol = 0;
  private String _idec = null;

  public void setSessionContext(final SessionContext context)
     throws RemoteException
  {
     _context = context;
  }

  public void ejbCreate(final String noRekening, final String password, final String
  idECommerce, final int gol) throws CreateException, SecurityException,
  RemoteException
  {
          this._noRekening = noRekening;
          this._password = password;
          this._idECommerce = idECommerce.trim();
          this._gol = gol;
          this._idec = idECommerce.trim();

     try
     {
          role = ((RoleHome)ResourceManager.
          getLocalEJBHome("EJBonpay.RoleHome")).
          findByPrimaryKey(noRekening.trim(), password.trim(), gol);
          this._kategori = role.getKategori();
     }catch (Exception finder){
          throw new CreateException("Rekening tidak terdaftar, error:" +
          finder.toString());
     }
  }
```

4.2.3 Pengesetan Properti

EJB menggunakan properti lingkungan komponen dengan mengimplementasikan *java.util.Properties* untuk menyeting lingkungan untuk menyebarkan dan *runtime*. Properti juga digunakan untuk menyeting transaksi, keamanan akses obyek, peyimpanan basis data, dan pelayanan lainnya. Properties

dapat diakses baik oleh kontainer dan bean yang tersebar saat *runtime*. Terdapat 2 fail properti yang digunakan yaitu *deploy.properties* dan *obyek.properties*.

4.2.3.1 Deploy.Properties

Deploy.properties menunjukkan parameter-parameter lingkungan yang digunakan untuk setiap komponen saat disebar dan *runtime* yang dituliskan dalam sebuah fail bernama *deploy.properties*. Contoh fail ini adalah sebagai berikut

```
ejipt.classServer.host=10.126.12.33
ejipt.ejbJars=EJBonpay_ejb.jar
ejipt.roleHomeName=EJBonpay.RoleHome
ejipt.loginSessionHomeName=EJBonpay.LoginSessionHome
ejipt.jdbcSources=source1
source1.ejipt.sourceDriverClassName=sun.jdbc.odbc.JdbcOdbcDriver
source1.ejipt.sourceURL=jdbc:odbc:oraonpay
source1.ejipt.sourceUser=nofa
source1.ejipt.sourcePassword=nofa
```

4.2.3.2 Obyek.Properties

Obyek.Properties menunjukkan parameter-parameter lingkungan tiap-tiap *class* yang membentuk suatu paket komponen EJB. Setiap *class* pasti mempunyai parameter-parameter tersendiri termasuk pemberian hak akses (*role*) untuk masing-masing user berdasarkan kebutuhan sistem, direktori *class*, waktu *time out* obyek, jenis class, dan sebagainya. Contoh fail properties tersebut adalah

```
ejb.beanHomeName=EJBonpay.LoginSessionHome
ejb.homeInterfaceClassName=ejbeans.LoginSessionHome
ejb.remoteInterfaceClassName=ejbeans.LoginSession
ejb.enterpriseBeanClassName=ejbeans.LoginSessionBean
ejb.stateManagementType=stateful_session
ejb.sessionTimeout=600
ejb.allowedIdentities=all
ejb.runAsMode=SYSTEM_IDENTITY
```

4.3 Implementasi Klien

Dalam mengimplementasikan klien terdapat dua type klien yaitu yang berbasis Java RMI dan CORBA. Klien berbasis Java RMI menggunakan *Java*

Naming and Directory Interface (JNDI) untuk mencari obyek dan menggunakan *Java Transaction API* (JTA) untuk mengontrol transaksi sedangkan klien CORBA dengan *CORBA naming Service* (COS *Naming*) untuk mencari obyek dan menggunakan *Obyek Transaction Services* (OTS) untuk mengontrol transaksi. JNDI tahapannya sebagai berikut

1. Pencarian (look up) home obyek

   Pencarian home obyek dilakukan dengan menyimpan properti lingkungan obyek kedalam obyek *java.util.Properties*. Obyek ini menjadi parameter dalam konstruktor obyek context. Home obyek diinstanskan melalui obyek context. Contoh kode pencarian home obyek adalah sebagai berikut

```
final Properties properties = new Properties();
properties.setProperty(Context.INITIAL_CONTEXT_FACTORY, "allaire.ejipt.ContextFactory");
properties.setProperty(Context.PROVIDER_URL, "ejipt://" + ipserver + ":2323");
_context = new InitialContext(properties);
home = (LoginSessionHome) _context.lookup("EJBonpay.LoginSessionHome");
```

2. Menggunakan home obyek untuk membuat sebuah obyek EJB

   Setelah klien tereferensikan dengan sebuah home obyek maka obyek ini dapat digunakan untuk menciptakan obyek-obyek EJB seperti berikut

```
LogSessObj = home.create();
```

3. Memanggil *method* bisnis dari obyek EJB

   Setelah obyek EJB tercipta maka klien dapat memanggil method-method bisnis yang terlah tersedia seperti berikut

```
LogSessObj.Login();
```

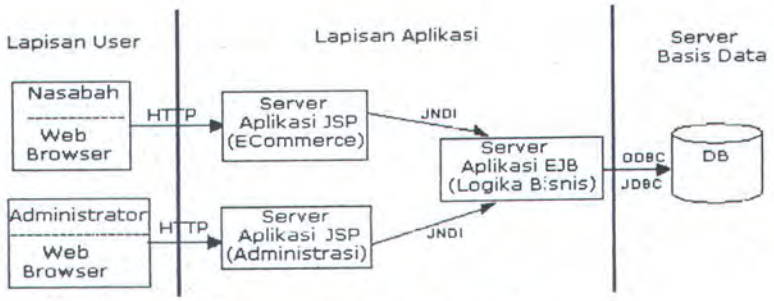
4. Menghapus obyek EJB

Method remove()dipanggil dari obyek EJB ataupun obyek home. *Method* ini digunakan kontainer untuk menghapus (*destroy*) obyek EJB pemanggilannya adalah sebagai berikut

```
LogSessObj.remove();
```

4.4 Arsitektur Sistem

Dalam penerapannya, sesuai perancangan maka dalam implementasi dibutuhkan minimal 2 server aplikasi, 1 server basis data dan 1 atau lebih komputer sebagai klien. Dalam hal ini dibutuhkan suatu jaringan TCP/IP untuk berkomunikasi antar komputer. Susunan arsitektur komputer pada implementasinya tampak pada gambar 4.1.



Gambar 4.1 Arsitektur sistem

4.4.1 Lapisan User (Klien)

Lapisan ini berhubungan secara langsung dengan user melalui web browser. Web browser me-*request* skrip *Java Server Pages* (JSP) dari server aplikasi melalui protokol HTTP. Aplikasi klien ini dibagi 3 jenis yaitu aplikasi untuk nasabah yaitu ecommerce, aplikasi administrasi staf bank, dan administrasi staf ecommerce. Untuk memperlihatkan implementasi distribusi obyek maka

digunakan 2 macam aplikasi ecommerce yang dijalankan pada 2 server aplikasi yang berbeda..

Spesifikasi klien yang digunakan sistem ini adalah

- Web Browser : Internet Explorer 4.0 keatas.
- Mendukung cookies untuk keperluan session servlet.

4.4.2 Lapisan Aplikasi

Pada lapisan ini terdapat server aplikasi yang bertugas untuk menyebarkan dan *runtime* komponen EJB. Komponen EJB dan aplikasi administrasi terletak dalam suatu server aplikasi dan aplikasi ecommerce terletak pada server aplikasi lainnya yang memanggil obyek-obyek EJB yang dibutuhkan.

- Sistem operasi : Windows 95/NT/2000/Linus/Solaris
- Server aplikasi dan server web : JRun 3.01
- Library : JDK 1.3
- Aplikasi basis data : Oracle Client (di mesin server aplikasi Jrun yang menyebarkan EJB)

4.4.3 Lapisan Basis data

Pada lapisan basis data digunakan perangkat lunak RDBMS Oracle 8i yang berjalan sistem operasi Windows NT. Koneksi EJB ke basis data dengan bantuan jembatan JDBC-ODBC.

BAB V

UJI COBA DAN EVALUASI

Dalam bab ini dibahas mengenai uji coba aplikasi sistem pembayaran online ini. Uji coba dilakukan untuk mengetahui bahwa aplikasi ini berjalan sebagaimana mestinya. Ada 2 tujuan yang di uji coba, yaitu kemampuan untuk mendistribusikan sumber daya berupa obyek-obyek dalam lingkungan multi-tier dan dapat melakukan transaksi obyek yang dapat menjamin ACID properties (*Atomicity, Consistency, Isolation*, dan *Durability*). Uji coba dilakukan dalam sistem jaringan lokal (intranet) dan digunakan beberapa komputer yang masing masing bertindak sebagai klien, server aplikasi EJB dan JSP serta server basis data.

5.1 Lingkungan Uji Coba

Uji coba dilakukan dalam suatu jaringan lokal (intranet) dengan menggunakan 4 buah komputer yang terbagi 1 komputer bertindak sebagai server basis data, 1 komputer sebagai server aplikasi EJB dan aplikasi administrasi, 2 komputer sebagai server aplikasi ecommerce, serta komputer-komputer yang bertindak sebagai klien nasabah. Berikut ini adalah sistem operasi beserta perangkat lunak pendukung yang dipakai dalam uji coba:

- Server basis data

  Sistem operasi : Windows NT

  Perangkat lunak : Oracle 8i server

- Server aplikasi EJB dan aplikasi admnistrasi web

  Sistem operasi : Windows NT

  Perangkat lunak : JRun 3.01, Oracle Client, Java Development Kit 1.3

- Server aplikasi ecommerce

  Sistem operasi : Windows NT

  Perangkat lunak : JRun 3.01, Oracle Client, Java Development Kit 1.3

- Klien (browser web)

  Sistem operasi : Windows NT

  Perangkat lunak : Internet Explorer 5.0

5.2 Pelaksanaan Uji Coba

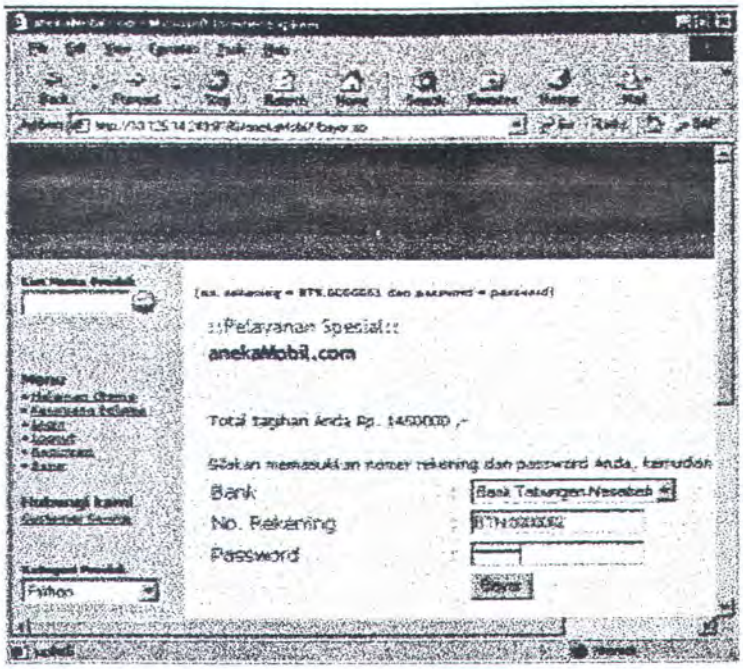
Setelah semua sarana dan perangkat lunak pendukung sudah siap maka dilakukan uji coba prototipe yang telah dibuat. Sebelum pelaksanaan uji coba dilakukan maka pertama kali perlu dilakukan mengaktifkan semua server dan pengecekan keaktifan web server. Setelah terpenuhi maka pada aplikasi klien dapat dilakukan uji coba.

5.2.1 Aplikasi Pembayaran Online Ecommerce

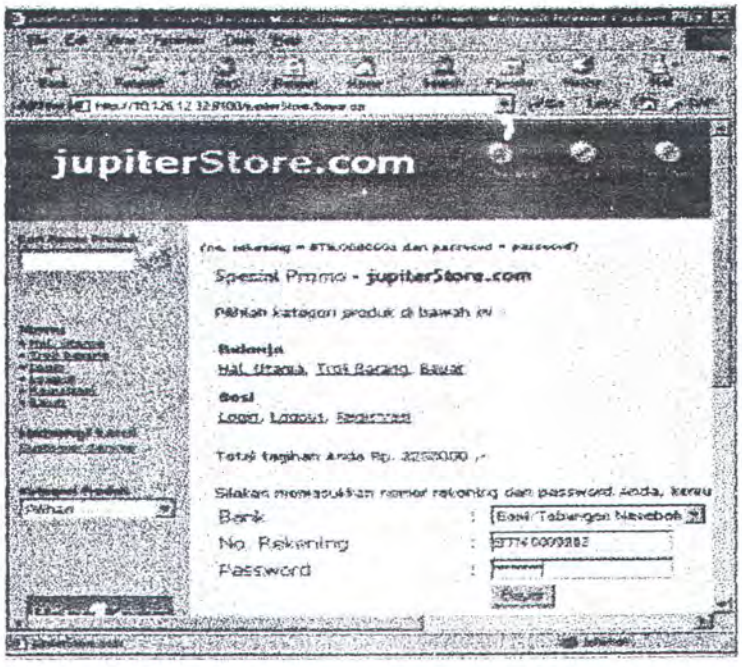
Proses pembayaran nasabah dilakukan melalui halaman ecommerce yang dilakukan setelah selesai berbelanja dan memasukkan seluruh daftar pembeliannya di keranjang belanja. Kemudian menuju halaman pembayaran dengan diawali login untuk mengecek otentikasi dan otorisasi nasabah terhadap rekeningnya dibank.

Gambar 5.1 Login rekening nasabah di halaman ecommerce

Setelah validasi selesai, nasabah dapat menekan tombol bayar untuk melakukan pembayaran tagihan. Pembayaran yang dilakukan nasabah merupakan transfer saldo yang terjadi antara rekening nasabah dengan ecommerce yang harus memenuhi beberapa krteria, jika kriteria tersebut tidak terpenuhi maka terjadi kekeliruan proses sehingga proses digagalkan dan menyimpan status tersebut kedalam basis data log error sebaliknya status tersebut disimpan kedalam basis data log transaksi. Konkurensi terjadi ketika terdapat dua atau lebih nasabah yang melakukan pembayaran pada situs ecommerce menggunakan nomer rekening yang sama. Skenario ini diujicobakan ketika dua user masing-masing dengan nomer rekening yang sama yaitu BTN.0000002 membayar melalui dua situs ecommerce seperti yang terlihat pada gambar 5.2 dan gambar 5.3.



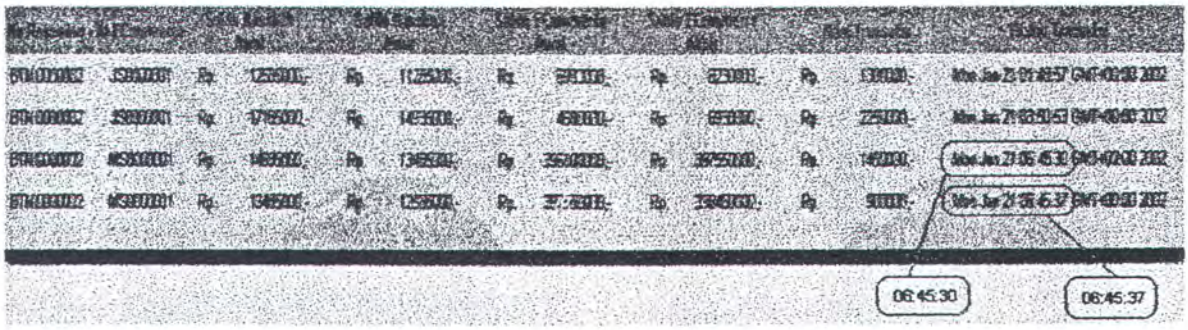
Gambar 5.2 Pembayaran tagihan nasabah melalui anekaMobil.com



Gambar 5.3 Pembayaran tagihan nasabah melalui jupiterStore.com

Hasil kedua transaksi ini dapat dilihat pada halaman administrasi root atau bank pada gambar 5.4 dimana dalam menyimpan ke basis data di tabel transaksi

terdapat perbedaan waktu simpan dengan selisih beberapa milidetik. Hal ini menandakan bahwa ketika suatu rekod basis data sedang diupdate maka operasi tersebut terjadi secara atomic dan rekod tersebut diisolasi dari proses update rekod yang lainnya, sehingga menghasilkan data yang konsisten. Apabila terjadi kerusakan yang bersifat fisik maka tahapan proses yang disimpan diulangi lagi jika kerusakan tersebut berakhir dan sistem berjalan secara normal lagi sehingga hal ini memenuhi unsur durabilitas basis data.



Gambar 5.4 Halaman pencarian data transaksi pembayaran

5.2.2 Aplikasi Administrasi Root Dan Staf Bank

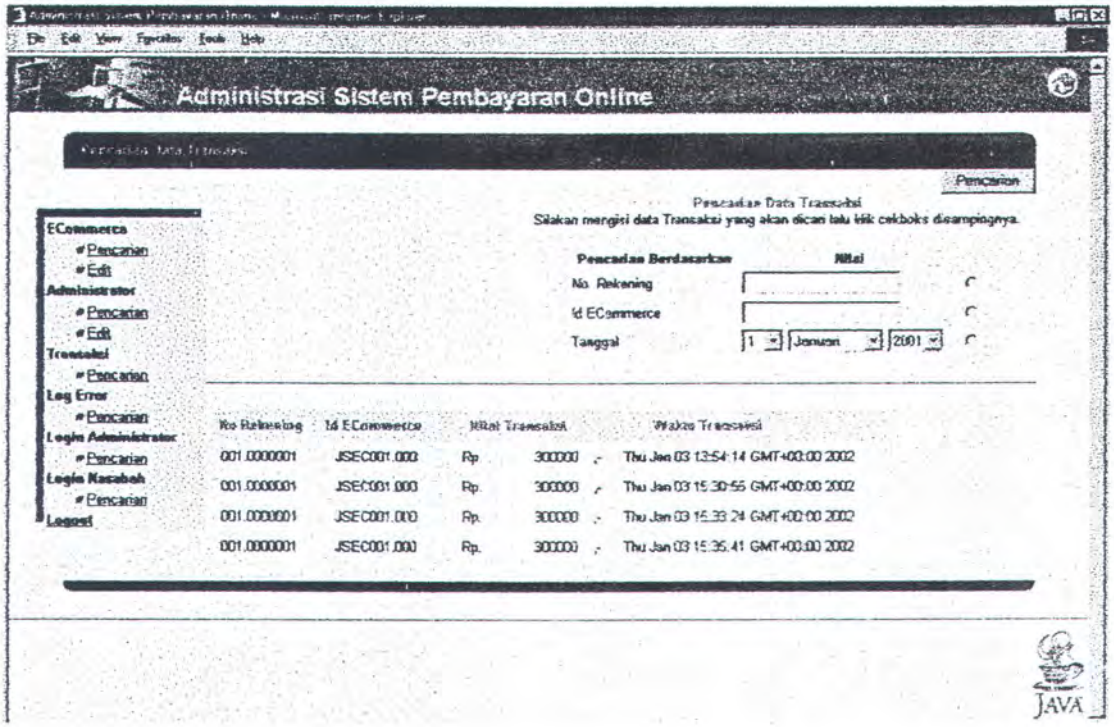
Petugas bank dapat melakukan administrasi data-data baik administrator maupun ecommerce termasuk juga melakukan pelacakan transaksi, kekeliruan, dan login user. Fitur-fitur ini dapat dimonitor oleh administrator bank dan root berkaitan dengan pelaksanaan aktifitas operasional sistem pembayaran.

Untuk mengetahui administrator yang berhak maka haruslah dilakukan login dan mencatatnya kedalam basis data. Nrs dan password yang benar akan mengantarkan administrator bank untuk masuk ke sistem, sebaliknya terjadi penolakan dan kegagalan login. Login yang valid akan mengantarkan admnistrator untuk melakukan aktifitas-aktifitas kehalaman administrasi. Halaman administrasi memiliki fitur-fitur penelusuran aktifitas sistem, seperti pencarian

data log error, log transaksi, dan data login user serta fitur-fitur yang bersifat administrasi seperti pengubahan data ecommerce dan administrasi..

5.2.3 Aplikasi Administrasi Staf ECommerce

Petugas ecommerce dapat melakukan aktifitas administrasi sesuai dengan hak akses yang dimiliki. Sebuah ecommerce memiliki beberapa staf yang berhak mengakses ecommerce yang sesuai. Jika staf mengakses ecommerce yang tidak sesuai terjadi penolakan dan kegagalan login. Operasi-operasi yang dilakukan staf ecommerce ini berupa pencarian data log error dan transaksi yang spesifik.

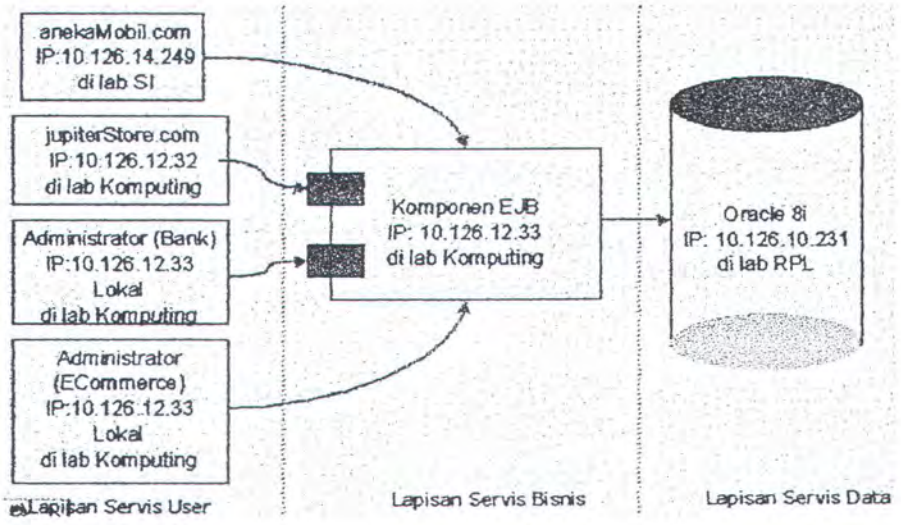


Gambar 5.5 Halaman pencarian data transaksi oleh staf ecommerce.

5.3 Distribusi Obyek Enterprise Bean

Pada sistem pembayaran online ini, server aplikasi bank menyebarkan obyek-obyek enterprise bean. Server aplikasi ini bertindak sebagai pengatur sumber daya yang berupa obyek-obyek enterprise bean yang tersebar sehingga

dapat dipanggil oleh aplikasi klien yang berjalan pada server aplikasi yang terletak pada komputer lainnya dan diantaranya terpisahkan secara fisik oleh jaringan. Arsitektur ini terlihat pada gambar 5.6. Situs ecommerce anekaMobil.com dijalankan pada mesin dengan IP 10.126.14.249, sedangkan jupiterStore.com dijalankan pada mesin dengan IP 10.126.12.32, keduanya memanggil obyek-obyek sistem pembayaran online yang disebarkan oleh bank sebagai *resource manager* yang terletak pada mesin dengan IP 10.126.12.33. *Resource manager* mengakses basis data secara remote yang terletak pada mesin ber-IP 10.126.10.231. Berdasarkan karakteristik sistem tersebut maka dapat dikatakan bahwa sistem pembayaran online ini terdistribusi.



Gambar 5.6 Arsitektur uji coba sistem pembayaran online dalam lingkungan intranet

Sistem pembayaran online ini tergolong dalam model sistem terdistribusi berbasis obyek karena antara aplikasi ecommerce meminta layanan dari penyedia obyek tersebar yang berada pada bank melalui suatu lapisan antar muka tersendiri.

Ecommerce sebagai klien mengirim pesan ke obyek enterprise bean yang menginterprestasikannya untuk memutuskan servis yang harus dilaksanakan.

BAB VI

PENUTUP

Bab penutup menjelaskan tentang kesimpulan dan kemungkinan pengembangan lebih lanjut dari aplikasi sistem pembayaran online.

6.1 KESIMPULAN

Beberapa kesimpulan dari tugas akhir ini adalah sebagai berikut :

1. Penggunaan komponen Enterprise Java Bean mendukung basis data dalam menangani transaksi dalam memenuhi properti ACID (*Atomicity, consistency, Isolation, Durability*) sehingga dapat diterapkan pada aplikasi yang digunakan oleh banyak user yang membutuhkan *sharing* data atau pada aplikasi yang sangat memungkinkan menderita kerusakan jaringan atau mesin.
2. Sumber daya (obyek) yang disebarkan oleh pengatur sumber daya (*resource manager*) dapat diakses oleh klien yang dipisahkan jaringan dengan cara berkomunikasi sehingga memenuhi kriteria sistem terdistribusi.
3. Apabila terdapat dua atau lebih operasi yang akan mengupdate basis data untuk menjaga konsistensi data maka penyimpanan ke dalam basis data dilakukan dengan menggunakan sistem *locking* (penguncian). Ketika suatu operasi sedang mengupdate suatu rekord basis data rekord tersebut di *lock* sehingga operasi yang lain yang akan mengupdate rekord yang sama menunggu sampai operasi pertama selesai mengupdate dan melepaskan

*lock*nya. Hal ini dapat dibuktikan berdasarkan hasil uji coba yang menunjukkan adanya selisih waktu antara dua operasi yang saling konkuren.

6.2 SARAN

Kemungkinan pengembangan lebih lanjut dari perangkat lunak yang telah dibuat

1. Agar didapatkan sistem yang optimal dan dapat digunakan untuk sistem yang berskala lebih besar maka layanan keamanan pengiriman data perlu diperhatikan misalnya seperti enkripsi data dan pemanfaatan SSL.
2. Sistem dan aplikasi yang dibuat dalam tugas akhir ini dalam bentuk protipe bisa dikembangkan menjadi lebih kompleks yang dapat memenuhi fitur-fitur tertentu yang lebih spesifik.

DAFTAR PUSTAKA

[Anuff 96] Ed Anuff, *The Java Sourcebook*, John Wiley & Sons, inc, 1996.

[Echols 88] John M. Echols, Hasan Shadily, *Indonesian English Dictionary*, PT. Gramedia Jakarta, 1988.

[FSTConsortium98] Financial Services Technology Consortium, *Bank Internet Payment System Spesification version 1.0*, August 24, 1998.

[Goodwill 99] James Goodwill, *Developing Java Servlet*, SAMS Publishing, 1999.

[Mahmoud 99] Qusay H. Mahmoud, *Distributed Programming with Java*, 1999.

[Roman 99] Ed Roman, *Mastering Enterprise Java Beans and the Java™ 2 Platform Enterprise Edition*, John Wiley & Sons, Inc, 1999.