PENGEMBANGAN PERANGKAT LUNAK DIAGNOSA MESIN MOBIL BERDASARKAN STANDAR OBD-II

TESIS

Karya tulis sebagai salah satu syarat
untuk memperoleh gelar Magister dari
Institut Teknologi Bandung

Oleh

ALEX XANDRA ALBERT SIM

NIM : 23511051

(Program Studi Magister Informatika)

INSTITUT TEKNOLOGI BANDUNG

2013

PENGEMBANGAN PERANGKAT LUNAK DIAGNOSA MESIN MOBIL BERDASARKAN STANDAR OBD-II

Oleh

ALEX XANDRA ALBERT SIM

NIM : 23511051

(Program Studi Magister Informatika)

Institut Teknologi Bandung

Menyetujui,

Tanggal ……..

Pembimbing

_____

(Prof. Dr. Ing. Ir. Benhard Sitohang)

ABSTRAK

PENGEMBANGAN PERANGKAT LUNAK DIAGNOSA MESIN MOBIL BERDASARKAN STANDAR OBD-II

Oleh

Alex Xandra Albert Sim

NIM: 23511051

(Program Studi Magister Informatika)

OBD-II merupakan sebuah standar yang dikembangkan Uni Eropa agar manufaktur mobil tidak memiliki monopoli pada bisnis reparasi kendaraan. OBD-II memungkinkan pihak ketiga untuk berkomunikasi dengan ECU. Tesis ini ditulis untuk eksplorasi dan dokumentasi ECU dan OBD-II, sehingga implementasi perangkat lunak diagnostik dapat dilakukan dengan mudah. Perangkat lunak diagnostik juga dibangun sebagai contoh implementasi.

Tesis dimulai dengan penjelasan latar belakang masalah, dan dilanjutkan dengan tujuan dan ruang lingkup tesis. Selanjutnya, dokumentasi mengenai ECU (komponen, cara kerja) dan OBD-II diberikan. Tesis ini kemudian memberikan penjelasan mengenai analisa dan perancangan dari perangkat lunak yang dikembangkan, yang dilengkapi dengan detil implementasi dan hasil pengembangan. Pengembangan dilakukan menggunakan metode *waterfall*, dan berhasil dengan baik karena spesifikasi OBD-II yang telah matang dan tidak

berubah. Selanjutnya, tesis juga mendiskusikan pengujian yang dilakukan terhadap perangkat lunak. Hasilnya: komunikasi dengan ECU berjalan tanpa masalah, tetapi terdapat masalah dalam penampilan informasi, khususnya pada modul grafik. Informasi tidak dapat ditampilkan dengan akurat, karena terdapat jeda waktu pengiriman satu perintah ke perintah berikutnya. Hal ini menyebabkan adanya jeda antara perintah pada sumbu x dan sumbu y, sehingga hasil kalkulasi nilai pada sumbu y baru dilakukan setelah nilai pada sumbu x berubah. Untuk menanggulanginya, perangkat lunak menampilkan jeda waktu antara perintah sumbu x dan sumbu y kepada pengguna, agar pengguna tidak salah menginterpretasikan informasi yang diberikan.

Kesimpulan yang didapatkan yaitu bahwa OBD-II dapat dengan mudah diimplementasikan, dan memberikan dampak positif bagi manufaktur, pengguna, dan bengkel. Implementasi standar OBD-II dengan metode *waterfall* sangat optimal, tetapi komunikasi dengan serial port tidak optimal untuk mendapatkan informasi *real-time*, karena jeda waktu antar perintah yang telah dijelaskan sebelumnya.

Kata Kunci: mesin, ECU, OBD-II, sistem diagnosa

ABSTRACT

DEVELOPING A CAR ENGINE DIAGNOSTIC SOFTWARE BASED ON OBD-II STANDARD

By

Alex Xandra Albert Sim

NIM: 23511051

(Program Studi Magister Informatika)

OBD-II is a standard developed by the European Union so that car manufacturers don't have monopoly over the car repair business. OBD-II facilitates third party developers to communicate with the ECU. This thesis is written to explore and document OBD-II and ECU, so that people who want to implement a diagnostic software can do so easily. As a proof of concept, a diagnostic software that implements the OBD-II was built.

This thesis starts with an explanation on the problem statement, and proceeds with the thesis objectives and boundaries. Next, this thesis provides documentation for ECU (components, how it works) and OBD-II. It then explains the analysis and design of the diagnostic software built, complete with the implementation details and result. The software was built using the waterfall method, and it works very

well because OBD-II specification is mature and not changing. The thesis then proceed to discuss the testings done with the software. Overall, the software can communicate with ECU well, via OBD-II. There is, however, a problem in displaying some information on the software, particularly on the graph module. The information can't be displayed accurately, because there is a delay on sending one command to another. This leads to a delay for the commands on x axis and y axis, and resulting in value of y axis being calculated after the the value of x axis changed. The software displays the delay between both commands to the user, so s/he can interpret the information correctly.

It is concluded that OBD-II is easy enough to implement, and gives more benefits for manufacturers, customers, and car repair shops. Implementing OBD-II using the waterfall method was found to be optimal, but communicating via serial port for real-time data is not optimal, because of delays from one command to another.

Keyword: engine, ECU, OBD-II, diagnostic system

PEDOMAN PENGGUNAAN TESIS

Tesis S2 yang tidak dipublikasikan terdaftar dan tersedia di Perpustakaan Institut Teknologi Bandung, dan terbuka untuk umum dengan ketentuan bahwa hak cipta ada pada pengarang dengan mengikuti aturan HaKI yang berlaku di Institut Teknologi Bandung. Referensi kepustakaan diperkenankan dicatat, tetapi pengutipan atau peringkasan hanya dapat dilakukan seizin pengarang dan harus disertai dengan kebiasaan ilmiah untuk menyebutkan sumbernya.

Memperbanyak atau menerbitkan sebagian atau seluruh tesis haruslah seizin Direktur Program Pascasarjana, Institut Teknologi Bandung.

KATA PENGANTAR

Pertama-tama, penulis ingin memanjatkan segala puji dan syukur kepada Tuhan Yang Maha Esa atas rahmat, karunia, dan berbagai kemudahan yang diberikan selama penulisan tesis yang berjudul "Pengembangan Perangkat Lunak Diagnosa Mesin Mobil Berdasarkan Standar OBD-II" ini. Selama menempuh pendidikan Magister dan menyelesaikan tesis, penulis juga memperoleh dukungan dan bantuan dari banyak pihak. Oleh karena itu, penulis ingin mengucapkan banyak terima kasih kepada:

1. Bapak Prof. Dr. Ing. Ir. Benhard Sitohang, selaku dosen pembimbing selama penulis mengerjakan tesis, yang dengan penuh kesabaran memberikan bimbingan, arahan, nasihat, serta embelajaran selama penulisan tesis. Penulis mendapatkan sangat banyak ilmu, melebihi apa yang terdapat di dalam tesis ini.
2. Bapak Bayu Hendrajaya, ST.,MT., atas ketersediaannya menjadi penguji pada sidang tesis, dan memberikan masukan untuk perbaikan tesis.
3. Bapak Adi Mulyanto, M.T., atas ketersediaannya menjadi penguji pada sidang tesis, dan memberikan masukan untuk perbaikan tesis, serta selaku wali akademik yang telah memberikan bimbingan, arahan, dan segenap perhatian selama masa perkuliahan.
4. Seluruh staf pengajar Magister Informatika dan Rekayasa Perangkat Lunak ITB atas ilmu dan didikan selama masa perkuliahan. Bapak Ade Taryat dan seluruh pegawai Tata Usaha yang sangat membantu kelancaran administrasi perkuliahan dan tesis.
5. Seluruh teman-teman S2 ITB, khususnya IF dan RPL 2011 untuk kebersamaan yang menyenangkan selama masa perkuliahan.
6. Seluruh pihak lainnya yang tidak dapat disebutkan satu per satu.

Telah banyak pengalaman, ilmu, dan pengetahuan yang penulis dapatkan selama menyelesaikan studi di ITB. Memiliki kesempatan untuk menempuh pendidikan di ITB merupakan sebuah kehormatan yang besar bagi penulis.

Penulis juga menyadari bahwa tesis ini masih jauh dari sempurna. Karenanya, masukan maupun kritik untuk perbaikan di masa mendatang sangat diharapkan oleh penulis.

Bandung, 16 Januari 2013,

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LISTING KODE

Bab I Pendahuluan

Bab ini menguraikan latar belakang, rumusan masalah, ruang lingkup dan batasan masalah, tujuan, metodologi, serta sistematikan pembahasan dari tesis ini.

I.1 Latar Belakang Masalah

Untuk mendapatkan performa yang optimal, pada umumnya sistem kendaraan (mobil) kerap kali menggunakan unit kontrol khusus yang dikenal dengan Engine Control Unit (ECU). Selain berperan sebagai unit kontrol, ECU bahkan seringkali berperan sebagai “otak” dari sebuah mobil - melakukan optimasi performa sesuai keadaan dan kontrol terhadap banyak bagian dari mobil.

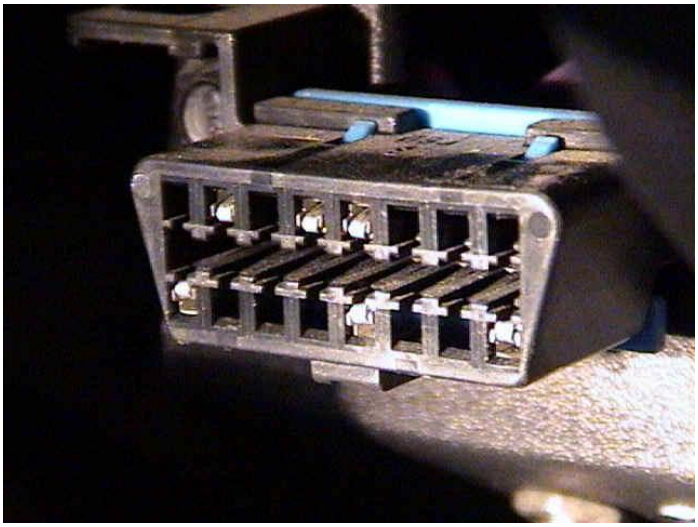
Gambar I-1 ECU Mercedes Benz CLK W208

(Sumber: (GT-Shop, 2011))

ECU memberikan manfaat yang sangat besar kepada pengguna maupun manufaktur mobil. Pengguna mendapatkan performa mesin yang optimal, sementara manufaktur mendapatkan kontrol penuh terhadap mesin dan keseluruhan mobil. Sayangnya, penggunaan ECU juga memberikan dampak

negatif, terutama kepada bengkel mobil independen yang tidak bekerja sama dengan manufaktur: perbaikan dan diagnosa kerusakan dalam mobil tidak lagi dapat dilakukan dengan manual, tetapi harus melalui pembacaan data digital ECU. Hal ini tentu menyebabkan kesulitan bagi bengkel karena perbaikan maupun deteksi kerusakan tidak lagi dapat dilakukan dengan cara klasik (melihat kerusakan di dalam mesin / badan mobil) melainkan harus menggunakan sistem komputer (yang tidak diberikan oleh manufaktur kepada bengkel umum / independen).

Kesulitan bengkel untuk bersaing dengan manufaktur mobil ini telah ditanggapi oleh Uni Eropa dengan menerbitkan regulasi mengenai praktek anti-kompetisi dalam pasar bebas (European Union, 2010). Dengan regulasi ini, manufaktur mobil tidak boleh melakukan penguncian terhadap ECU yang digunakan oleh mobil yang dibangun. Tujuan akhir dari regulasi ini adalah untuk memberikan jalan bagi bengkel agar tidak bergantung pada manufaktur mobil dalam melakukan reparasi mobil.



Gambar I-2 Port Koneksi OBD-II

(Sumber: (Movin on GPS, 2012))

Regulasi dari Uni Eropa tersebut kemudian diikuti dengan standar diagnosa EOBD (European On-Board Diagnostic), yang lebih dikenal dengan nama OBD-

II (Gambar I-2) di Amerika Serikat dan negara lainnya. OBD-II harus diterapkan pada seluruh kendaraan yang beredar di Eropa dan Amerika, sehingga seluruh kendaraan di Eropa dan Amerika dapat dipastikan memiliki sistem ini. Dengan adanya OBD-II, maka bengkel-bengkel independen tetap dapat melakukan diagnosa dan perbaikan mobil, tanpa harus bergantung kepada manufaktur.

Sayangnya, bahkan dengan adanya regulasi dari Uni Eropa tersebut, manufaktur mobil tetap menambahkan banyak fitur-fitur yang di luar standar OBD-II. Hal ini dianggap sebagai sebuah *competitive advantage* dari sebuah manufaktur, dan akibatnya setiap manufaktur memiliki tambahan fungsi ECU yang berbeda-beda. Hal ini menyebabkan standar OBD-II hanya mampu mendiagnosa bagian-bagian terpenting dalam mobil, tetapi tidak keseluruhan mobil.

Tesis ini dibuat dengan tujuan eksplorasi teknologi ECU dan perangkat diagnosanya, baik yang sesuai dengan standar OBD-II maupun fitur spesifik manufaktur. Eksplorasi teknologi ECU dilakukan dalam hal cara kerja ECU serta cara komunikasi antara ECU dengan perangkat diagnosa. Cara kerja ECU penting dipahami untuk mendapatkan cetak biru dari hal-hal yang harus ditangani ECU, sementara komunikasi dengan ECU penting untuk dapat melakukan optimasi dan diagnosa dari informasi yang didapatkan melalui cetak biru tersebut. Perangkat diagnosa dibangun sebagai bukti keberhasilan mempelajari tekologi tersebut. Untuk tambahan fitur non-standar OBD-II, maka pembahasan teknologi ECU dibatasi pada satu mobil sebagai contoh kasus.

I.2 Perumusan Masalah

Rumusan masalah pada tesis ini yaitu eksplorasi cara kerja ECU serta pengembangan sebuah perangkat lunak diagnosa yang dapat berkomunikasi dengan ECU. Perangkat lunak yang berkomunikasi dengan ECU merupakan perangkat lunak diagnosa ECU, yang dapat melakukan pembacaan status kendaraan serta melakukan pengaturan berbagai parameter dalam kendaraan.

I.3 Tujuan Tesis

Tujuan dari tesis ini adalah:

1. Meneliti dan mendokumentasikan cara kerja (termasuk arsitektur dan fungsionalitas) ECU, termasuk komunikasi antara ECU dengan perangkat diagnosa ECU pada umumnya.
2. Mengembangkan perangkat lunak diagnosa ECU yang dapat digunakan untuk melakukan diagnosa terhadap ECU, sesuai dengan standar OBD-II.
3. Menambahkan pembacaan fitur-fitur ECU yang tidak termasuk dalam standar OBD-II untuk dapat mengerti cara kerja keseluruhan ECU.

I.4 Ruang Lingkup dan Batasan Masalah

Tesis ini mencakup kajian cara kerja dari ECU, termasuk komunikasi antara ECU dengan perangkat diagnosa. Selain itu, pengembangan perangkat lunak diagnosa juga dilakukan, dengan target pencapaian minimal kompatibilitas dengan standar OBD-II.

I.5 Metodologi Penelitian

Tahapan-tahapan penelitian yang akan dilalui adalah sebagai berikut:

1. Studi literatur

   Pada studi literatur, berbagai literatur mengenai ECU pada berbagai mobil dikumpulkan untuk dikaji lebih lanjut. Berdasarkan seluruh literatur yang didapatkan, dokumentasi mengenai ECU, termasuk arsitektur, cara kerja, serta cara komunikasi ECU dengan perangkat lunak diagnostik didokumentasikan. Selain itu, tahap ini juga melakukan dokumentasi mengenai OBD-II sesuai dengan literatur yang didapatkan.

   Hasil dari studi literatur ialah bab 2 tesis ini, yaitu tinjauan pustaka yang membahas secara rinci mengenai ECU dan OBD-II.

2. Pengembangan Perangkat Lunak

   Setelah mendapatkan informasi yang cukup dan mengetahui cara kerja ECU dan OBD-II dengan baik, maka pengembangan perangkat lunak diagnosa akan mulai dilakukan. Metodologi pengembangan yang digunakan akan disesuaikan dengan informasi mengenai ECU dan OBD-II yang didapatkan.

   Hasil dari tahapan ini ialah dokumentasi analisis dan perancangan sistem, yang dapat dibaca pada bab 3 tesis ini.

3. Dokumentasi Perangkat Lunak yang Dikembangkan

   Pada tahap dokumentasi, berbagai tantangan dan perbedaan antara perancangan dan implementasi yang ditemui selama pengembangan akan dicatat, lengkap dengan alasan atau penjelasan mengenai perbedaan dan tantangan tersebut. Bagaimana perangkat lunak tersebut memenuhi tujuan awal tesis (bagian Tujuan Tesis) dan apakah perangkat lunak memiliki karakteristik performa yang baik akan didokumentasikan juga.

   Tujuan utama dari dokumentasi perangkat lunak yang dikembangkan ini ialah untuk memberikan wawasan dan pengetahuan (*insight*) kepada pembaca yang ingin mengembangkan perangkat lunak sejenis. Hasil dari tahap ini dapat dibaca pada bab 4 tesis ini.

4. Kesimpulan dan Saran

   Setelah semua tahapan dijalankan, maka akan dilakukan penarikan kesimpulan dari pengerjaan tesis ini. Kesimpulan yang diambil didasarkan pada pembahasan yang dilakukan pada bagian Tujuan Tesis, untuk menelaah lebih jauh dari apa yang telah dilakukan pada tahap dokumentasi sebelumnya.

   Saran untuk penelitian lanjutan tesis ini juga akan diberikan, sesuai dengan penemuan-penemuan yang didapatkan selama pengerjaan tesis. Saran yang diberikan diharapkan menjadi potensi untuk penelitian lanjutan dalam topik yang sama, ataupun bagaimana meningkatkan metode atau implementasi penelitian yang dilakukan pada tesis ini.

I.6 Sistematika Penulisan

Penulisan laporan tesis ini terdiri dari lima bab dengan perincian sebagai berikut:

Bab I. Pendahuluan, menguraikan latar belakang, rumusan masalah, ruang lingkup dan batasan masalah, metodologi penelitian, serta sistematika penulisan tesis.

Bab II. Tinjauan Pustaka, yang berisi dasar-dasar teori yang mendukung tesis ini. Pembahasan akan dilakukan terhadap dua hal utama: ECU dan pengembangan perangkat lunak.

Bab III. Analisis dan Perancangan, berisi dokumentasi dari analisis dan perancangan sistem yang akan dibangun. Menjelaskan rancangan abstrak dari perangkat lunak yang dikembangkan serta alasan digunakannya rancangan tersebut.

Bab IV. Konstruksi, Hasil dan Evaluasi, berisi penjelasan bagaimana sistem dibangun, tantangan yang dihadapi dalam pengembangan sistem, serta keberhasilan dan kegagalan dalam implementasi sistem. Hasil dari pengembangan juga dijelaskan pada bagian ini, dalam bentuk penjelasan rinci dari sistem yang dibangun. Hasil pengembangan tersebut kemudian dievaluasi, untuk memastikan sistem telah memenuhi tujuan yang dijelaskan pada bagian I.3.

Bab V. Kesimpulan dan Saran, berisi kesimpulan yang dapat diambil dari pelaksanaan tesis ini serta saran pengembangan untuk penelitian lanjutan.

Bab II Tinjauan Pustaka

Bab ini akan memberikan pengertian dan penjelasan mengenai komponen-komponen yang ada dalam ECU, cara kerja ECU, serta OBD-II.

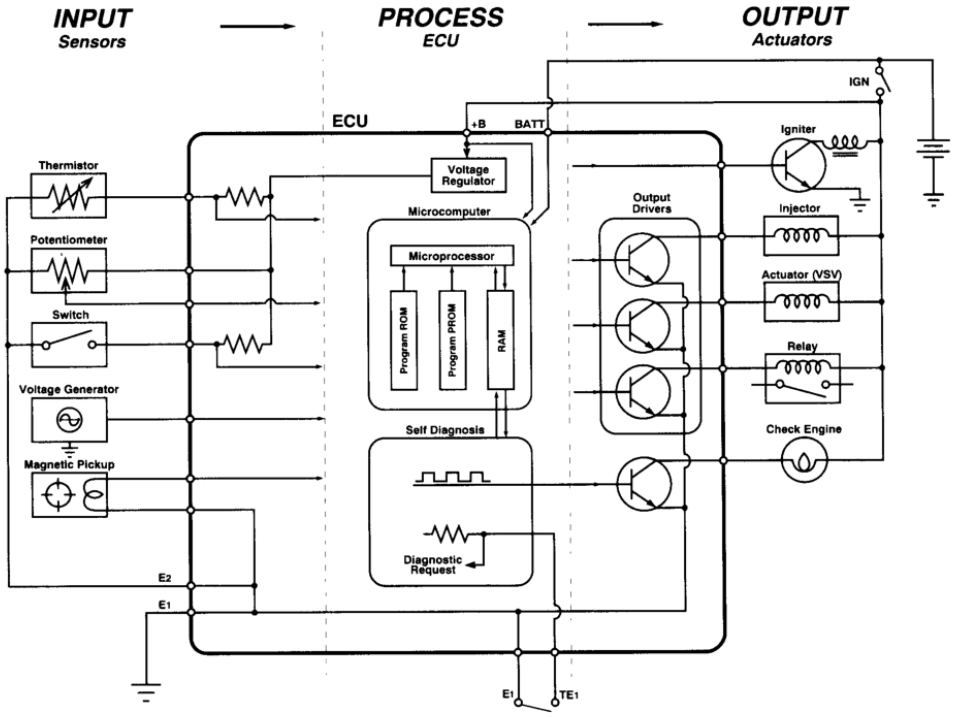
II.1 Engine Control Unit (ECU)

Engine Control Unit (ECU) adalah jantung dari sistem manajemen sebuah kendaraan. ECU merupakan sebuah komputer yang mengendalikan segala hal dalam mesin, mulai dari penguncian kendaraan ketika mesin mati sampai dengan kontrol waktu yang tepat untuk api pertama saat mesin dinyalakan (O'Connor, 2009). Awalnya, ECU hanya berfungsi dalam pengendalian karburator untuk memberikan kombinasi bahan bakar dan udara yang tepat untuk pembakaran optimal. ECU dikembangkan untuk optimasi mesin serta pengurangan emisi mesin (Ritcher, 2006).

Semakin berkembangnya teknologi dalam kendaraan, semakin bertambah juga fungsi-fungsi yang dapat ditangani oleh ECU. ECU modern bahkan menggunakan mikroprosesor untuk melakukan tugasnya, dengan efek menjadikan arsitektur ECU sama seperti arsitektur komputer pada umumnya. ECU modern mengandung komponen perangkat keras (mikroprosesor, memori, ROM) dan komponen perangkat lunak (*firmware*) yang menjadikan ECU dapat diprogram untuk melakukan apa saja. Kemampuan kontrol ECU secara bebas ini menjadikan fungsi ECU berkembang dari kontrol emisi menjadi kontrol keseluruhan mobil. Adapun hal-hal yang ditangani oleh ECU termasuk, tetapi tidak terbatas pada: kontrol rasio udara dan bahan bakar, kontrol tempo pengapian, kontrol transmisi, sistem anti pencuri (kunci pintu), pengaturan kursi, dan lain-lain (Autologic Software, 2012).

II.1.1 Komponen ECU

ECU dibentuk oleh banyak komponen yang berbeda-beda, tergantung kepada fitur yang dimiliki oleh ECU tersebut. Setiap manufaktur membangun ECU dengan cara yang berbeda-beda, sehingga detil dari komponen ECU akan berbeda dari satu manufaktur ke manufaktur lain. Begitupun, secara garis besar dapat dilihat bahwa setiap ECU memiliki komponen-komponen sebagai berikut:



Gambar II-1 Komponen ECU

(Sumber: (Toyota Motor Sales, 2005))

1. Komponen Sensor

   Merupakan komponen yang bertanggung jawab dalam memberikan data dari kondisi-kondisi yang dimonitor oleh ECU. Misalnya, komponen sensor oksigen bertugas untuk membaca kadar oksigen di udara dan memberikan hasil pembacaan tersebut kepada komponen prosesor untuk diproses lebih lanjut.

2. Komponen Prosesor

   Komponen yang melakukan kalkulasi dan pengambilan keputusan berdasarkan data yang diberikan oleh komponen sensor. Bagian ini umumnya merupakan komponen mikroprosesor sederhana serta perangkat lunaknya.

3. Komponen Komunikasi (Diagnosis)

   Komponen ini bertugas untuk memberikan data dari komponen sensor yang telah diproses kepada sistem diagnosa di luar ECU. Komunikasi sistem luar kepada ECU dilakukan melalui komponen ini.

4. Komponen Aktuator (Keluaran)

   Hasil dari pengolahan data dari komponen prosesor dikirimkan kepada komponen aktuator untuk mengendalikan fungsi tertentu pada kendaraan, dan sekaligus memberikan informasi kepada pengendara, melalui antarmuka yang telah disediakan. Umumnya komponen ini memberikan informasi melalui *dashboard* kendaraan (lihat Gambar II-2).

Gambar II-2 Pesan Kesalahan ECU

(Sumber: (Elec_Intro, 2010))

Secara sederhana, alur data dari komponen-komponen yang telah dijelaskan sebelumnya dapat dilihat pada Gambar II-1. Cara kerja dari ECU secara rinci dibahas pada bagian Cara Kerja ECU.

II.1.2 Cara Kerja ECU

Secara garis besar, alur kerja dari sebuah ECU dapat disederhanakan menjadi tiga langkah utama:

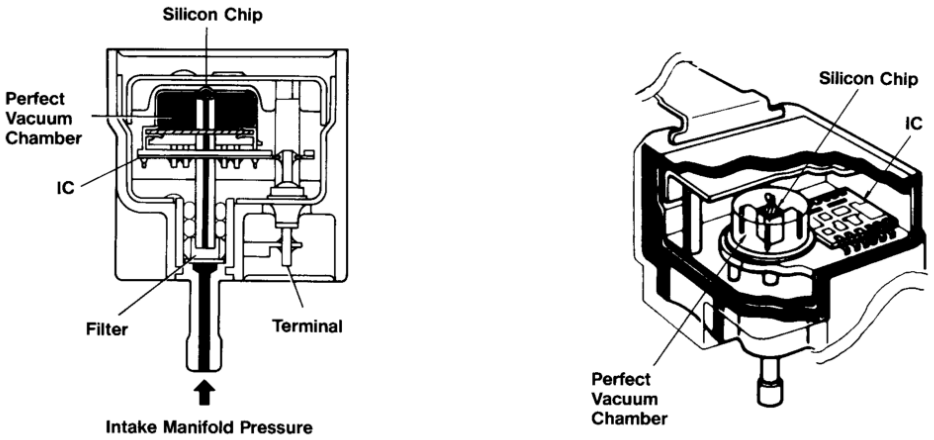
1. **Input** – Pengambilan data lingkungan sekitar kendaraan oleh sensor-sensor yang dipasang dalam kendaraan.
2. **Proses** – Analisa dan kalkulasi data hasil masukan oleh mikrokomputer yang mana hasil kalkulasi akan menjadi dasar pengambilan keputusan dalam operasional mesin.
3. **Output** – Keluaran dari hasil proses berupa perintah kepada bagian tertentu mesin untuk melakukan sesuatu, sesuai hasil dari kalkulasi pada langkah sebelumnya.

II.1.2.1 Langkah Input ECU

Sebagai sumber informasi dari ECU adalah sensor masukan yang digunakan ECU sebagaimana manusia menggunakan panca indranya. Manusia mengambil keputusan berdasarkan oleh apa yang dirasakan oleh panca indranya: ketika manusia menyentuh air dan indra perasa mendeteksi panas yang luar biasa, maka manusia kemungkinan besar tidak akan meminum (atau melompat masuk ke) air tersebut. Sensor memberikan ECU kemampuan yang sama. Sensor temperatur mesin misalnya, memberikan informasi panas mesin kepada ECU agar ECU dapat menghidupkan kipas mesin ketika mesin terlalu panas.

Setiap manufaktur (dan mungkin jenis mesin / mobil) memiliki jenis, jumlah, dan cara kerja sensor yang berbeda-beda. Untuk memberikan gambaran, tulisan ini akan membahas satu contoh sensor yang digunakan oleh mesin umumnya serta cara kerjanya. Sensor yang akan dibahas adalah sensor vakum untuk mendeteksi muatan (beban) mesin.

Sensor vakum umumnya terletak dekat dengan bagian *intake manifold*, yang menyalurkan campuran udara dan bahan bakar ke dalam silinder mesin. Sensor vakum mendeteksi jumlah udara dalam *intake manifold* dengan memantau perubahan tekanan udara di dalamnya.



Gambar II-3 Sensor Vakum pada Mesin

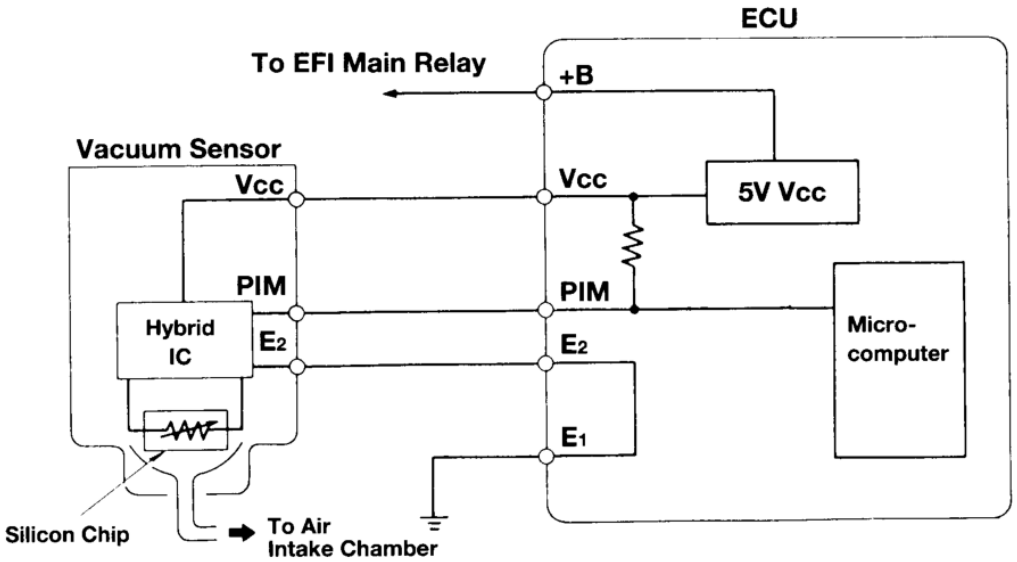
(Sumber: (Toyota Motor Sales, 2005))

Sensor vakum terdiri dari sebuah chip silikon piezoresistif dan sebuah Integrated Circuit (IC). Piezoresistif merupakan sifat sebuah silikon di mana jumlah hambatan elektrik (resistan) dapat berubah-ubah sesuai dengan tekanan yang diberikan pada silikon tersebut. Chip silikon tersebut sebagian dimasukkan ke dalam ruang vakum dan sebagian lagi diberi tekanan dari *manifold*. Ketika tekanan udara dalam *intake manifold* berubah, maka chip tersebut akan melentur dan mengalami perubahan resistan. Resistan yang terus berganti tersebut kemudian menyebabkan perubahan pada signal tegangan (listrik) pada terminal PIM (Pressure Intake Manifold), yang membaca nilai tekanan pada *intake manifold*.

Hasil pembacaan dari PIM ini kemudian dikirimkan kepada ECU untuk diproses lebih lanjut.

II.1.2.2 Langkah Proses ECU

Proses analisa dan kalkulasi terhadap data yang diberikan oleh sesnor dalam ECU dilakukan dengan sangat sederhana. Perhitungan umumnya dilakukan hanya dengan membaca dan membandingkan nilai terhadap tabel perhitungan yang sudah ada. Hal ini dilakukan karena keterbatasan kemampuan komputasi yang dimiliki sistem serta banyaknya sensor yang harus terus dipantau. Setiap sensor yang dipantau berhubungan dengan ECU melalui cara yang berbeda-beda. Melanjutkan contoh mengenai sensor vakum, Gambar II-4 menunjukkan cara sensor vakum berkomunikasi dengan ECU.



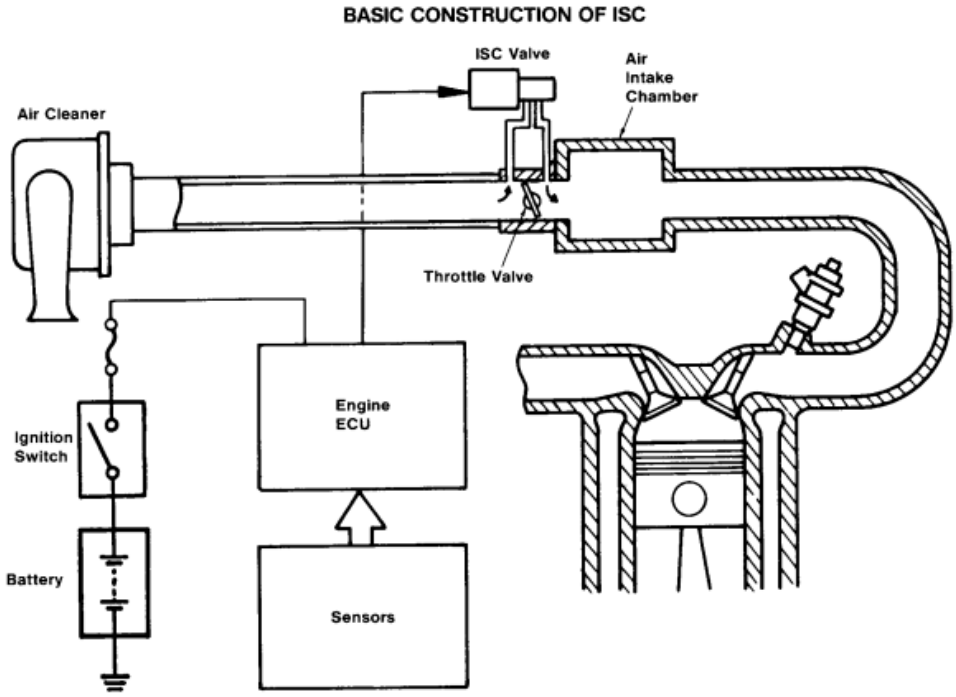
Gambar II-4 Diagram Pengkabelan Hubungan Sensor Vakum dengan ECU

(Sumber: (Toyota Motor Sales, 2005))

Sensor vakum terhubung dengan ECU melalui tiga terminal elektrik, yaitu Vcc (Voltage Constant Control), PIM (Pressure Intake Manifold), dan E2 (Earth Ground). Ketika tekanan dalam *intake manifold* meningkat, signal tegangan yang diberikan oleh PIM juga ikut meningkat. Melalui perubahan signal dan data sensor-sensor lainnya maka mikrokomputer melakukan kalkulasi terhadap nilai muatan (beban) mesin.

II.1.2.3 Langkah Output ECU

Hasil proses dari ECU dijadikan sebuah perintah kepada bagian tertentu mesin melalui komponen aktuator. Aktuator merupakan komponen elektromekanikal seperti motor, solenoida, atau *relay*. Secara sederhana, aktuator bekerja dengan melakukan menggerakkan motor (atau komponen mekanik lainnya) sesuai dengan energi yang diberikan (tegangan, tekanan, panas, dll). Karena kompleksitas dari aktuator pada sensor vakum, maka ilustrasi yang diberikan untuk langkah output akan menggunakan bagian mesin yang lain, yaitu Idle Speed Control (ISC) pada mesin. Bagian ISC berada tepat sebelum *intake manifold* di dalam mesin, yang bertujuan untuk mengontrol kecepatan putaran mesin dengan mengatur jumlah udara yang diberikan ke bagian lain mesin. Gambar II-5 menunjukkan rancangan sederhana dari sebuah ISC.



Gambar II-5 Rancangan ISC Sederhana

(Sumber: (Toyota Motor Sales, 2005))

Jenis aktuator yang digunakan oleh ISC yaitu *stepper motor*. Motor ini digunakan untuk mengendalikan jumlah udara yang melewati katup mesin (*throttle valve* pada Gambar II-5). Ketika mendapatkan informasi dari sensor, ECU melakukan kalkulasi terhadap jumlah udara yang harus diberikan ke mesin, dan kemudian menggerakkan katup tersebut melalui ISC Valve (ISCV). *Stepper motor* terpasang di dalam ISCV, di mana motor ini menggerakkan katup mesin ke luar atau ke dalam. Bukaan inilah yang menjadi pengatur dari jumlah udara yang dapat lewat ke dalam mesin.

II.2 Onboard Diagnostics (OBD)

Seluruh mobil yang dipasarkan di Negara yang menerapkan regulasi OBD-II pada masa kini harus mengimplementasikan teknologi OBD-II, sesuai dengan aturan yang berlaku. Standar ini dikembangkan untuk memastikan tidak adanya praktik anti kompetisi dari manufaktur mobil.

Sebagai aturan, OBD mengatur 3 kategori standar:

1. Standar komunikasi dengan ECU,
2. Standar perintah (permintaan informasi) – PID, dan
3. Standar kode kesalahan – DTC.

Teknologi OBD memberikan keuntungan bagi bengkel (dan pengendara) mobil dengan memberikan informasi-informasi mengenai berbagai hal yang dikerjakan oleh ECU. Informasi yang diberikan diharapkan dapat membantu bengkel untuk mengindentifikasi permasalahan dalam mobil dan melakukan diagnosa serta perbaikan seperlunya.

II.2.1 Protokol Komunikasi OBD-II

Untuk berkomunikasi dengan perangkat diagnosa, OBD-II memanfaatkan beberapa protokol komunikasi. Terdapat 5 protokol utama yang digunakan oleh manufaktur mobil pada saat ini, yaitu:

1. J1850 PWM
2. J1850 VPW
3. ISO 9141-2
4. ISO 14230 (KWP2000)
5. ISO 15765-4 - Controller Area Network (CAN)

ISO 9141-2 dan ISO 14230 adalah sama dari sudut pandang signal elektrik (Volkswagen Group of America, 2000). Kedua standar ini umum digunakan di Eropa. Misalnya Pugeot dan MG/Rover menggunakan ISO 14230 sedangkan beberapa mobil MG lain menggunakan ISO 9141, yang pada dasarnya adalah sama. Hal ini terjadi dikarenakan oleh pergantian ECU untuk unit yang berbeda versi (O'Connor, 2009).

II.2.2 Mode Operasi OBD-II dan Parameter IDs (PIDs)

Parameter ID (PID) merupakan sebuah kode atau perintah yang disyaratkan OBD. Komunikasi terhadap ECU melalui OBD-II dilakukan dengan mengirimkan PID, sesuai dengan jenis informasi yang diinginkan, dan ECU akan memberikan respon dalam bentuk serangkaian *byte*. Rangkaian *byte* yang diberikan oleh ECU berupa format heksadesimal.

Standar OBD-II tidak mengharuskan manufaktur unuk mengimplementasikan seluruh PID yang ada. OBD-II bahkan tidak memberikan standar implementasi **minimal** yang harus diimplementasikan manufaktur untuk beberapa kategori permintaan yang ada, misalnya untuk Mode 1 dan Mode 2. Tetapi umumnya manufaktur mengimplementasikan standar-standar yang umum seperti kecepatan kendaraan dan RPM mesin.

Karena ada banyak kategori permintaan, standar OBD-II membagi PID ke dalam beberapa bagian, yang diberi nama Mode. Dokumen spesifikasi J1979 yang diterbitkan oleh SAE (Society of Automotive Engineers) mencatat 9 Mode PID sebagai berikut:

- **Mode 1:** menampilkan data *real-time* dari status kendaraan yang sedang berjalan. Misalnya hasil dari pembacaan sensor RPM mesin.
- **Mode 2:** memberikan data *snapshot* dari seluruh sensor dalam Mode 1 ketika terjadi kesalahan mesin. Data *snapshot* ini dikenal dengan nama *freeze frame*.
- **Mode 3:** ECU memberikan daftar Diagnostic Trouble Code (DTC) yang disimpan.
- **Mode 4:** mengirimkan perintah kepada ECU untuk menghapus seluruh DTC yang ada dalam memori serta mematikan Malfunction Indicator Lamp (MIL, Gambar II-2) jika lampu hidup.
- **Mode 5:** memberikan hasil pengujian dari sensor oksigen.
- **Mode 6:** memberikan hasil pengujian sensor lain-lain.
- **Mode 7:** memberikan data DTC yang tertunda (belum ditampilkan).
- **Mode 8:** mengendalikan operasional dari sistem *on-board*.
- **Mode 9:** memberikan data Vehicles Identification Number (VIN)

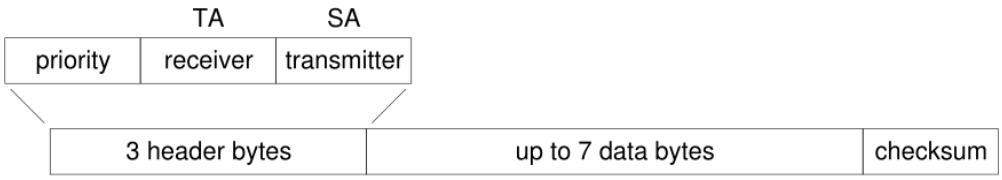
Selain 9 mode di atas, juga terdapat mode-mode lain yang digunakan oleh manufaktur. Kode-kode tersebut bersifat spesifik manufaktur, yang berarti mode yang ada berbeda-beda artinya untuk setiap manufaktur.

Pengiriman perintah kepada ECU melalui OBD-II harus menyertakan mode dan PID. Misalnya, jika ingin mengirimkan permintaan RPM mesin, maka perintah yang harus dikirimkan adalah `01 0C`, di mana `01` merupakan mode (mode 1) dan `0C` merupakan perintah (permintaan data RPM Mesin). Untuk mendapatkan informasi kesalahan pada RPM mesin (yang berada pada mode 2), maka perintah yang dikirimkan adalah `02 0C`.

II.2.3 Format Pesan OBD-II

Sistem OBD dirancang sebagai sistem yang fleksibel dan dapat menjadi sarana komunikasi beberapa perangkat sekaligus. Dalam pengiriman pesan antar beberapa perangkat, adalah sangat penting untuk menambahkan informasi mengenai jenis informasi yang dikirimkan, perangkat yang menerima informasi, serta perangkat yang mengirimkan informasi. Tingkat kepentingan dari informasi yang dikirimkan juga harus dipertimbangkan – posisi poros mesin jelas lebih penting untuk mesin yang sedang berjalan daripada permintaan informasi VIN. Karenanya, pesan yang dikirimkan juga menyimpan informasi prioritas pesan.

Informasi mengenai prioritas, penerima pesan, dan pengirim pesan biasanya diperlukan oleh penerima pesan bahkan sebelum jenis pesan yang dikirimkan diketahui. Untuk memastikan informasi ini didapatkan terlebih dahulu, standar OBD menaruh informasi ini di bagian depan (*head*) dari pesan. Data yang berada di depan ini dikenal dengan nama *header byte*.



Gambar II-6 Format Pesan OBD

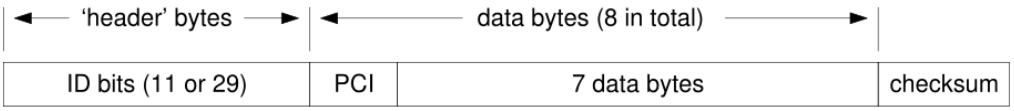
(Sumber: (O'Connor, 2009))

Gambar II-6 memperlihatkan format pesan yang digunakan oleh standar J1850, ISO 9141-2, dan ISO 14230. Format pesan ini menggunakan *header byte* untuk memberikan data prioritas, penerima, dan pengirim informasi. Sebagai catatan tambahan, banyak dokumentasi yang merujuk penerima sebagai TA dan pengirim sebagai SA.

Hal lain yang harus diperhatikan dalam pengiriman data ialah kesalahan dan korupsi data yang dapat terjadi dalam transmisi. Kesalahan seperti ini dapat

menyebabkan kesalahan interpretasi informasi yang dikirimkan. Untuk melakukan pengecekan kesalahan, protokol komunikasi umumnya memberikan metode verifikasi data yang diterima. Metode verifikasi ini dikenal dengan nama *checksum*. Untuk menangani kesalahan, informasi yang dikirimkan oleh OBD juga dilengkapi dengan *byte checksum* di bagian akhir data tersebut.

Empat standar dari OBD-II (J1850 PWM, J1850 VPW, ISO 9141-2, dan ISO 14230) menggunakan format pesan seperti yang ditampilkan pada Gambar II-6. Standar ISO 15765-4 (CAN) menggunakan standar yang berbeda, seperti yang ditampilkan pada Gambar II-7.



Gambar II-7 Format Pesan CAN OBD

(Sumber: (O'Connor, 2009))

Perbedaan utama dari kedua format tersebut terdapat pada struktur dari *header byte*. *Header byte* yang digunakan oleh CAN (disebut ID bits) teridir dari 11 atau 29 bit.

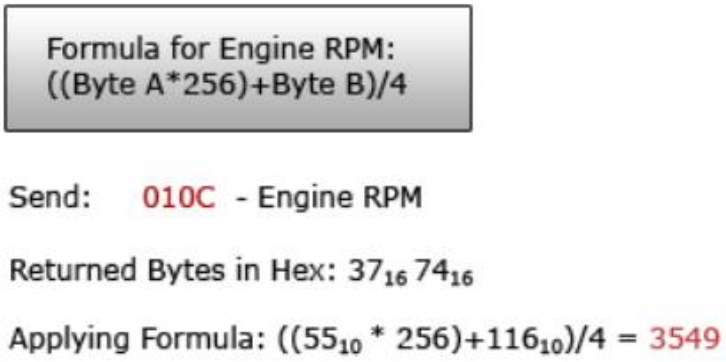
II.2.4 Pembacaan Respon OBD-II

Data yang dikembalikan oleh ECU diberikan dalam bentuk serangkaian *byte* yang harus diinterpretasikan lagi untuk mendapatkan arti dari data tersebut. Respon yang diberikan dapat berupa nilai yang sederhana ataupun berupa rangkaian bit yang harus dikalkulasi lagi untuk mendapatkan makna sebenarnya dari respon tersebut.

Kalkulasi terhadap rangkaian bit hasil respon OBD-II tidak memiliki rumus yang tetap. Metode kalkulasi yang digunakan berbeda-beda, sesuai dengan Mode dan PID yang diminta. Kerumitan rumus yang digunakan memiliki sebaran yang

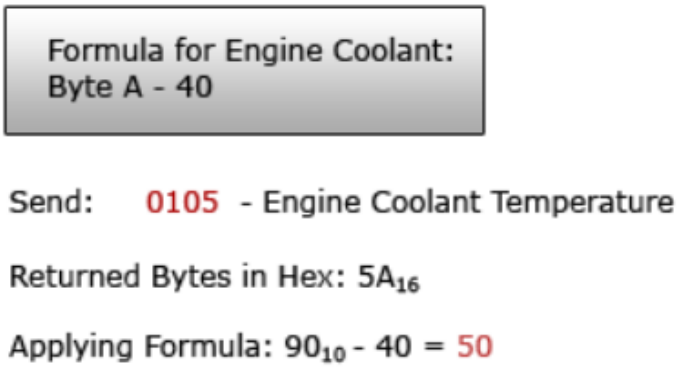
sangat luas, dari rumus sederhana seperti `x / 4` sampai rumus yang rumit seperti `jika byte A bernilai X, maka byte B berarti Y`. Untungnya, setiap PID memiliki standar perhitungan yang harus diikuti manufaktur, sehingga tidak terdapat permasalahan perbedaan rumus antar manufaktur.

Sebagai contoh, Gambar II-8 dan Gambar II-9 memperlihatkan kalkulasi yang harus dilakukan untuk permintaan RPM dan temperatur mesin.



Gambar II-8 Kalkulasi Respons untuk Perintah Permintaan RPM Mesin

(Sumber: (O'Connor, 2009))



Gambar II-9 Kalkulasi untuk Perintah Permintaan Temperatur Mesin

(Sumber: (O'Connor, 2009)

II.2.5 OBD-II Drive Cycle

Pembacaan nilai respon OBD-II seperti yang dijelaskan pada bagian II.2.4 hanya dapat dilakukan setelah sensor-sensor yang terdapat dalam ECU memiliki nilai hasil pembacaan. Pembacaan biasanya dilakukan ketika kendaraan dijalankan oleh pengguna secara normal, tetapi pembacaan pada saat penggunaan normal belum tentu dapat mencakup pembacaan keseluruhan sensor.

Untuk mendapatkan hasil pembacaan dari seluruh sensor, dibutuhkan langkah-langkah khusus yang harus dilakukan pengendara selama menggunakan kendaraannya. Langkah-langkah ini dikenal dengan nama "Drive Cycle" (DC). DC dari setiap manufaktur berbeda-beda, tetapi secara umum langkah-langkah berikut dapat dilakukan untuk menjalankan seluruh sensor yang harus diterapkan kendaraan untuk memenuhi standar OBD-II (Lyberty.com, 2005):

1. **Cold start**: menghidupkan mesin dalam keadaan di mana pendingin mesin memiliki suhu di bawah 50 ^{o}C dan tidak memiliki perbedaan suhu lebih dari 6 ^{o}C dengan lingkungan sekitar. Setelah mesin hidup, kunci diletakkan pada posisi netral.
2. **Idle**: mesin dihidupkan selama 2,5 menit dengan AC dan *rear defroster* (*wiper* belakang) diaktifkan.
3. **Accelerate**: matikan AC dan beban lainnya, kemudian mulai menekan pedal gas dan menahan pedal pada keadaan setengah penuh, sampai kendaraan mencapai kecepatan sekitar 88 km/h (kilometer per jam).
4. **Steady Speed**: kecepatan 88 km/h dipertahankan selama tiga menit.
5. **Decelerate**: pedal gas dilepas, tanpa menekan rem, sampai kendaraan melambat di kecepatan sekitar 32 km/h.
6. **Accelerate**: pedal gas ditekan kembali, kali ini dalam keadaan ¾ penuh, sampai kendaraan mencapai kecepatan 88 km/h kembali.
7. **Steady Speed**: kecepatan kendaraan dipertahankan, pada 88 km/h.
8. **Decelerate**: sama dengan langkah e, tetapi sampai kendaraan berhenti.

II.2.6 Pembacaan Pesan Kesalahan (DTC)

Diagnostic Trouble Code (DTC) merupakan standar format pesan kesalahan yang ditentukan oleh OBD. DTC dibuat untuk mempermudah pengendara dan bengkel dalam diagnosa kerusakan dan reparasi kendaraan. Kode DTC memberikan informasi mengenai kesalahan-kesalahan yang terdapat di dalam mobil.

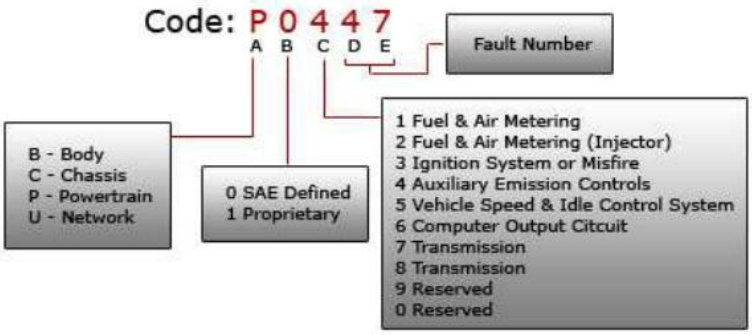
Terdapat empat jenis DTC yang didefinisikan oleh standar SAE, yang dapat dilhat pada Tabel II-1.

Tabel II-1 Jenis DTC Sesuai Standar SAE

Kode	Singkatan	Digit Pertama
Powertrain Code	P Code	0 – 3
Chassis Code	C Code	4 – 7
Body Code	B Code	8 – B
Network Code	U Code	C – F

Kode-kode tersebut mengidentifikasikan letak dari kesalahan sistem. Misalnya, kode *powertrain* berarti terdapat kesalahan dalam mesin.

DTC terdiri dari 5 angka dalam format heksadesimal. Angka pertama mengidentifiksaikan jenis kode (*powertrain*, *chassis*, dll). 4 angka selanjutnya memberikan informasi kesalahan lainnya. Angka kedua mengidentifikasikan apakah DTC yang diberikan merupakan kode standar SAE atau kode spesifik manufaktur, dan angka ketiga dan keempat mengidentifikasikan sistem yang mengalami kealahan. Gambar II-10 memperlihatkan contoh salah satu format DTC.



Gambar II-10 Contoh Format DTC

(Sumber: (O'Connor, 2009))

Arti pesan kesalahan yang ditampilkan oleh Gambar II-10 yaitu “terdapat kesalahan pada kontrol emisi (kode 4 pada kolom C), di bagian sirkuit ventilasi (kode 47 pada kolom D dan E)”. Seluruh standar PID dan DTC dapat dibaca pada dokumen standar OBD-II (SAE International, 2002).

Bab III Analisis dan Perancangan Sistem

Bab ini menjelaskan mengenai analisis dan perancangan dari sistem yang dikembangkan.

III.1 Analisis Kebutuhan Sistem

Bagian ini akan menjelaskan mengenai kebutuhan yang harus dipenuhi oleh sistem. Kebutuhan yang terdokumentasi merupakan kebutuhan fungsional, dengan kebutuhan non-fungsional disebutkan pada penjelasan detil sistem.

III.1.1 Deskripsi Kebutuhan Sistem

Sistem yang dikembangkan ialah perangkat lunak diagnosa mesin mobil. Sebagai perangkat diagnosa, sistem harus dapat berkomunikasi dengan ECU dan mengerti format pesan yang dikirimkan oleh ECU maupun yang harus diberikan ke ECU. Komunikasi antara sistem dan ECU dapat dilakukan dengan menggunakan perangkat lunak atau perangkat keras (dukungan terhadap salah satu saja sudah cukup). Untuk mempermudah pengguna, sistem diharapkan dapat berjalan pada *Personal Computer* (PC) standar, tanpa memerlukan perangkat lunak atau perangkat keras khusus, dengan pengecualian perangkat komunikasi antara PC dengan ECU.

Sistem yang dikembangkan berkomunikasi dengan ECU melalui perangkat keras, yaitu chip ELM 327. ELM 327 merupakan sebuah chip yang dirancang untuk menerjemahkan format pesan OBD-II menjadi format teks yang dapat dibaca oleh perangkat lunak yang ingin berkomunikasi dengan ECU. ELM 327 mendukung seluruh protokol komunikasi OBD-II, dan berkomunikasi dengan perangkat lunak lain melalui port Serial. Dengan menggunakan ELM 327, sistem yang dikembangkan dapat langsung mengirimkan perintah kepada ECU melalui port Serial, dan kemudian menerima respon dari ECU. Perlu diperhatikan juga bahwa ELM 327 hanya melakukan terjemahan format protokol komunikasi OBD-II,

sehingga sistem yang dikembangkan tetap harus melakukan pembacaan respon OBD-II, sesuai dengan yang dijelaskan pada bagian II.2.4.

Sistem yang dibangun harus dapat membaca informasi diagnosa yang didukung oleh OBD-II. Pembacaan informasi tertutup spesifik vendor merupakan hal yang bersifat tambahan. Untuk membantu pengguna dalam mencari informasi diagnosa spesifik vendor, sistem harus menyediakan antarmuka khusus di mana pengguna dapat mengirimkan pesan kepada ECU secara langsung, misalnya dalam bentuk terminal sederhana.

Karena diagnosa baru dapat dilakukan dengan optimal setelah seluruh sensor mesin diaktivasi (dengan melakukan *Drive Cycle (DC)*, yang dapat dibaca pada bagian II.2.5) maka sistem harus dapat digunakan ketika kendaraan sedang berjalan maupun berhenti (mesin kendaraan tidak mati). Jika digunakan dalam keadaan sedang berjalan, sistem harus memiliki antarmuka khusus untuk pengendara, yang hanya memberikan informasi-informasi yang berguna tentang DC. Antarmuka khusus pengendara ini harus sederhana dan mudah dilihat sehingga pengendara tidak terdistraksi dalam berkendara.

Gambar III-1 Hubungan Antara Sistem ke ECU Melalui ELM327

Secara singkat, berikut adalah daftar kebutuhan sistem yang harus digunakan:

1. Berjalan pada perangkat komputer (PC) standar.
2. Dapat melakukan komunikasi dengan ECU, baik melalui perangkat lunak maupun perangkat keras. Dalam tesis ini, sistem berkomunikasi dengan ECU melalui perangkat keras, yaitu chip ELM327 (lihat Gambar III-1).

3. Dapat mengerti pesan-pesan yang dikirimkan oleh ECU, dan dapat mengirimkan pesan dalam format yang dimengerti oleh ECU.
4. Dapat memberikan informasi diagnosa mesin, minimal informasi yang didukung oleh OBD-II.
5. Memiliki antarmuka yang mudah digunakan ketika sedang berkendara. Antarmuka ini tidak harus mendukung seluruh informasi diagnosa OBD-II, tetapi harus memberikan informasi yang berguna untuk menjalankan keseluruhan *Drive Cycle*.
6. Memiliki antarmuka khusus yang memberikan fasilitas kepada pengguna untuk mengirimkan pesan ke ECU secara langsung. Antarmuka ini berguna untuk membantu pengguna mendapatkan informasi diagnosa spesifik vendor.

III.1.2 Sekenario Penggunaan Sistem

Terdapat dua sekenario utama dalam menggunakan sistem, yaitu penggunaan pada saat kendaraan sedang berjalan dan penggunaan pada saat kendaraan sedang berhenti. Bagian ini akan mejelaskan sekenario penggunaan masing-masing dengan detil.

III.1.2.1 Penggunaan Saat Kendaraan Berjalan

Tujuan utama untuk penggunaan sistem pada saat kendaraan berjalan ialah untuk mendapatkan informasi diagnosa yang lengkap dari setiap sensor. Adapun langkah-langkah untuk mencapai tujuan tersebut adalah sebagai berikut:

1. Pengguna menghubungkan PC dengan ECU melalui perangkat ELM327.
2. Pengguna memberikan perintah ke sistem untuk memulai komunikasi dengan ECU.
3. Pengguna memilih antarmuka khusus berkendara.
4. Tombol khusus untuk mulai mencatat informasi dari sensor ditekan oleh pengguna. Tombol ini juga berada dalam antarmuka khusus berkendara.
5. Kendaraan dijalankan, sesuai, dengan langkah-langkah lengkap untuk memenuhi *Drive Cycle*, yang tertulis pada bagian II.2.5.

Selesai berkendara, pengguna dapat menjalankan mode kendaraan berhenti, jika masih ingin membandingkan informasi diagnosa dalam keadaan berjalan dan berhenti. Jika ingin mendapatkan informasi spesifik vendor, pengguna dapat masuk ke dalam mode terminal untuk langsung mengirimkan perintah ke ECU.

III.1.2.2 Penggunaan Saat Kendaraan Berhenti

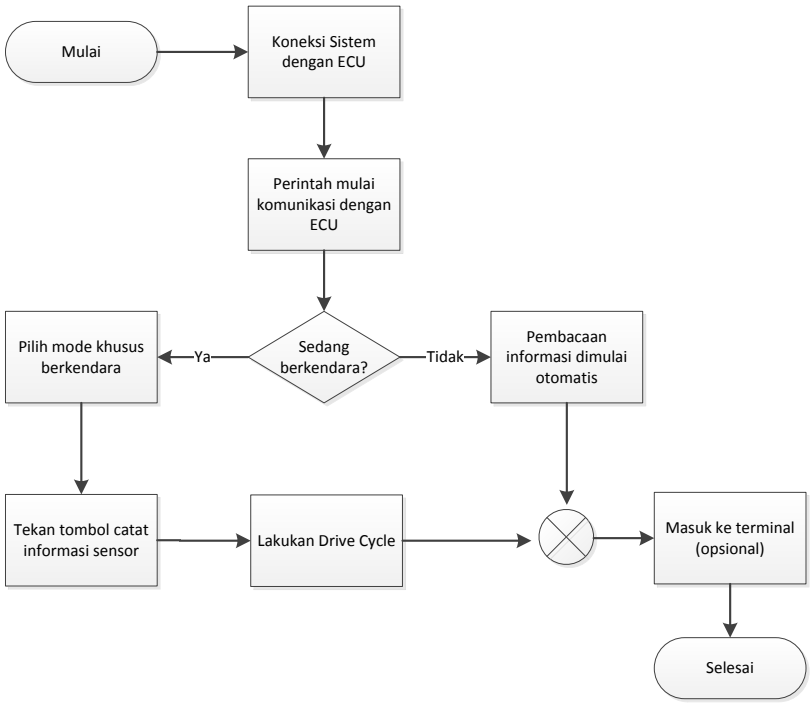
Menggunakan sistem pada saat kendaraan berhenti dapat memberikan **sebagian** informasi diagnosa kepada pengguna. Informasi yang diberikan tentunya tidak sebanyak yang didapatkan jika pengguna melakukan *Drive Cycle* terlebih dahulu. Adapun sekenario penggunaan pada saat kendaraan berhenti adalah sebagai berikut:

1. Pengguna menghubungkan PC dengan ECU melalui perangkat ELM327.
2. Pengguna memberikan perintah ke sistem untuk memulai komunikasi dengan ECU.
3. Sistem akan mulai melakukan pembacaan secara otomatis, dan pengguna dapat langsung mendapatkan informasi hasil diagnosa ECU.

Sama seperti pada sekenario kendaraan berjalan, pengguna dapat langsung memberikan perintah kepada ECU melalui mode terminal yang disediakan sistem.

III.1.3 Diagram Alir Sekenario Penggunaan Sistem

Untuk mempermudah pemahaman, Gambar III-2 menunjukkan diagram alir dari kedua sekenario sistem yang telah dijabarkan sebelumnya.



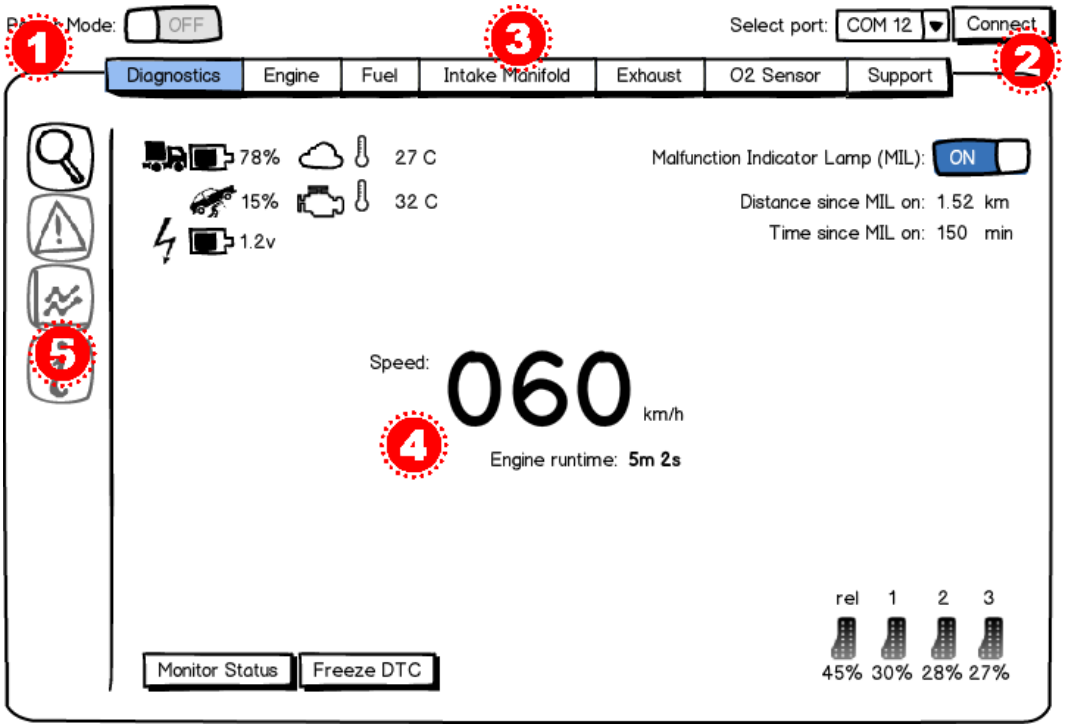
Gambar III-2 Diagram Alir Sekenario Penggunaan Sistem

III.1.4 Spesifikasi Detil Sistem

Bagian ini menjelaskan fitur-fitur yang ada dalam sistem, serta antarmuka yang ada, serta detil dari setiap antarmuka tersebut. Antarmuka dirancang sesuai dengan sekenario penggunaan yang telah dijelaskan pada bagian III.1.2. Jika terdapat informasi tambahan yang didapatkan pada bagian lain, maka akan terdapat catatan khusus. Jika terdapat kebutuhan non-fungsional juga akan diberikan catatan khusus.

III.1.4.1 Spesifikasi Antarmuka Utama Sistem

Gambar III-3 memperlihatkan rancangan antarmuka utama dari sistem. Antarmuka sistem dirancang dengan tujuan utama kemudahan navigasi. Penyusunan arsitektur informasi juga dilakukan untuk mempermudah pengguna dalam mencerna informasi diagnosa sistem yang sangat banyak.



Gambar III-3 Rancangan Antarmuka Utama Sistem

Adapun penjelasan dari setiap bagian antarmuka (sesuai dengan penomoran yang diberikan pada Gambar III-3) adalah sebagai berikut:

1. Tombol untuk berpindah ke mode terminal (disebut "**Power Mode**"). Tombol ini merupakan sebuah tombol *toggle (on / off)*, yang ikut berubah bentuk sesuai dengan status sistem.
2. Pemilihan port koneksi ke ECU, serta tombol untuk mulai berkomunikasi. Port yang ditampilkan merupakan seluruh port COM yang terkoneksi ke komputer, karena ELM327 mengidentifikasikan diri sebagai port COM kepada sistem operasi.
3. Pemilihan informasi yang ingin ditampilkan kepada pengguna. Pilihan diberikan dalam bentuk *tab*, di mana pengguna dapat memilih informasi yang ingin dilihat dengan melakukan klik pada *tab* yang diinginkan Terdapat enam informasi utama, yaitu:

a. **Diagnostics**, yaitu informasi diagnosa umum pada mesin. Terdiri dari tiga menu, yaitu informasi umum, DTC, dan grafik antara dua variabel dalam ECU.
b. **Engine**, yaitu informasi yang berhubungan dengan mesin.
c. **Fuel**, yaitu informasi yang berhubungan dengan sistem bahan bakar. Memiliki dua menu, yaitu informasi dasar dan informasi tekanan bahan bakar.
d. **Intake Manifold**, yaitu informasi yang berhubungan dengan intake manifold mesin.
e. **Exhaust**, berisi informasi sistem pembuangan mesin.
f. **O_2 Sensor**, memberikan informasi mengenai sensor oksigen mesin. Terdiri dari dua menu, yaitu informasi dasar dan tegangan sensor oksigen.

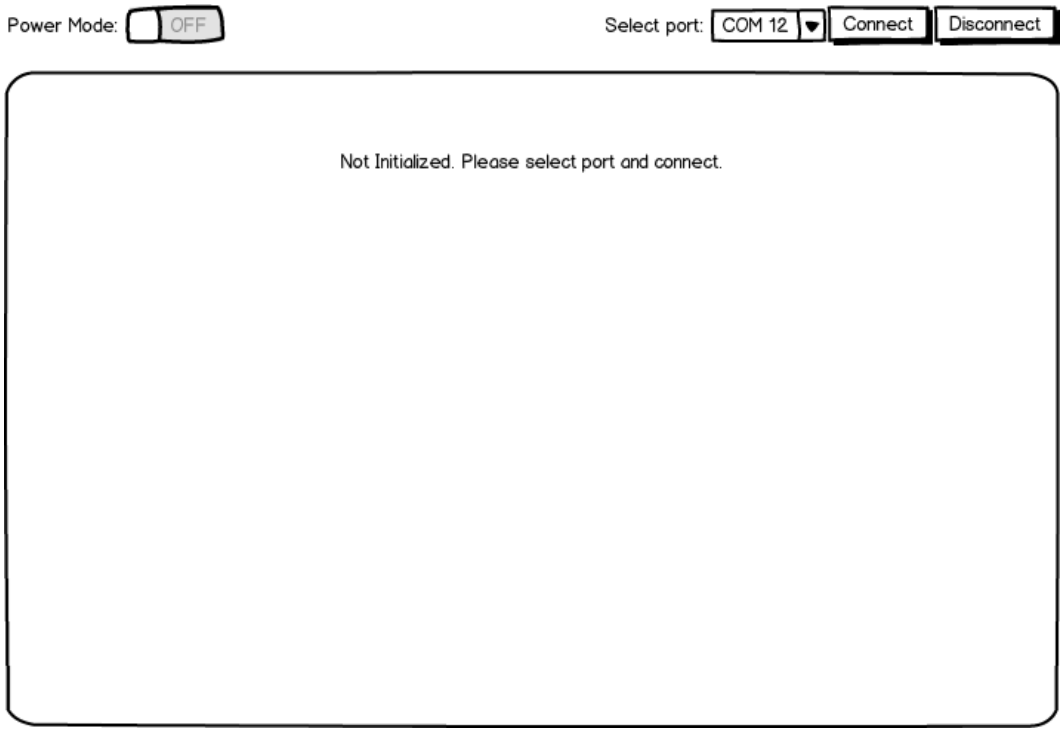
4. Panel utama. Menampilkan informasi pada bagian 3 (dan subbagian pada nomor 5) yang dipilih oleh pengguna.
5. Panel kontrol. Untuk memberikan pilihan submenu dari bagian 3 yang akan ditampilkan oleh sistem kepada pengguna. Panel ini merupakan kumpulan tombol, yang ketika ditekan akan mengubah isi panel utama (nomor 4) menjadi informasi yang disediakan oleh panel kontrol.

   Panel kontrol juga memberikan petunjuk visual kepada pengguna: ketika pengguna memilih sebuah submenu, maka tombol-tombol lain selain tombol submenu tersebut akan menjadi lebih terang (sehingga terkesan transparan oleh mata pengguna).

Untuk memperjelas fungsi-fungsi yang ada pada bagian tiga, maka subbab selanjutnya akan menjelaskan tiap-tiap bagian secara detil.

III.1.4.2 Spesifikasi Antarmuka Awal

Ketika pengguna pertama kali menjalankan sistem, pengguna akan dihadapkan dengan tampilan awal sistem, yang berupa tampilan kosong. Tampilan awal dapat dilihat pada Gambar III-4.



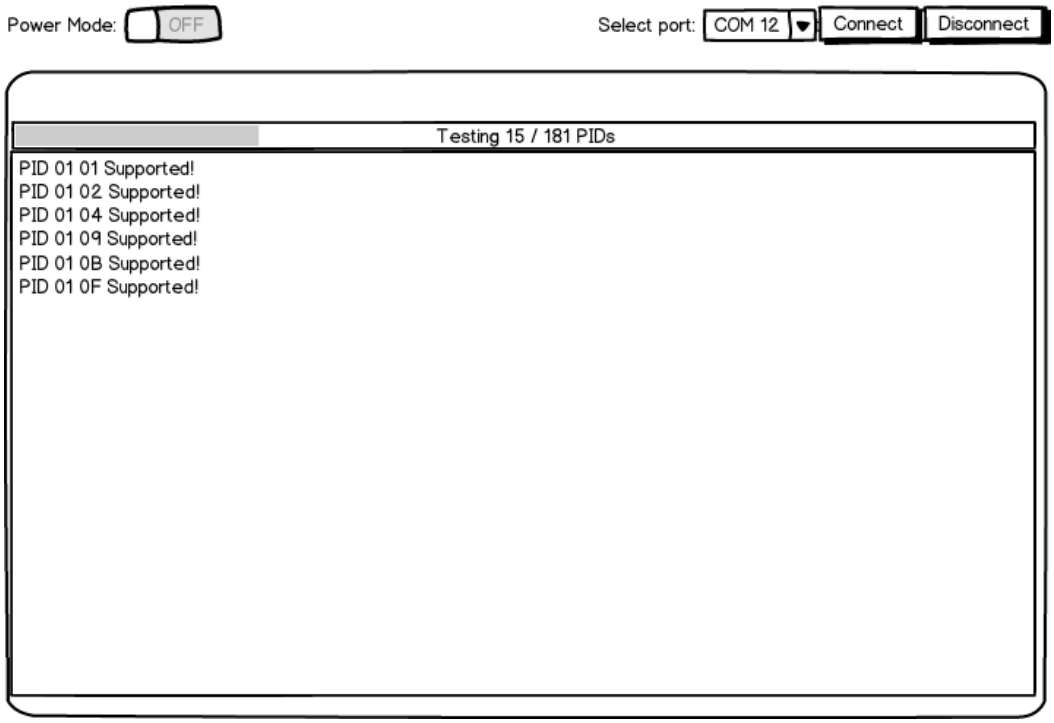
Gambar III-4 Rancangan Antarmuka Awal Sistem

Tampilan awal dibuat kosong untuk memastikan pengguna memilih dan mengkoneksikan diri ke ECU terlebih dahulu, sebelum mulai dapat melihat data-data dari ECU. Setelah memilih port yang tepat, pengguna dapat menekan tombol "Connect" untuk memulai koneksi sistem.

Ketika awal sistem terkoneksi, sistem akan memeriksa keberadaan PID pada ECU terlebih dahulu. Jika pengguna telah menggunakan sistem sebelumnya, sistem akan secara otomatis menyimpan data PID yang didukung. Jika sistem menemukan data PID yang didukung tersebut, maka sistem akan menanyakan apakah pengguna ingin menggunakan data sebelumnya. Jika pengguna ingin menggunakan data sebelumnya, sistem akan langsung membawa pengguna ke halaman awal "**Standard Mode**".

Pemeriksaan PID dilakukan dengan cara mengirimkan perintah PID satu per satu, dan membaca balasan dari ECU. Apabila ECU memberikan respon yang benar, PID akan dicatat oleh sistem sebagai PID yang didukung oleh ECU, dan sebaliknya jika respon ECU tidak sesuai standar, PID akan dicatat sebagai PID

yang tidak didukung. Gambar III-5 menunjukkan rancangan awal antarmuka untuk pengujian sistem.

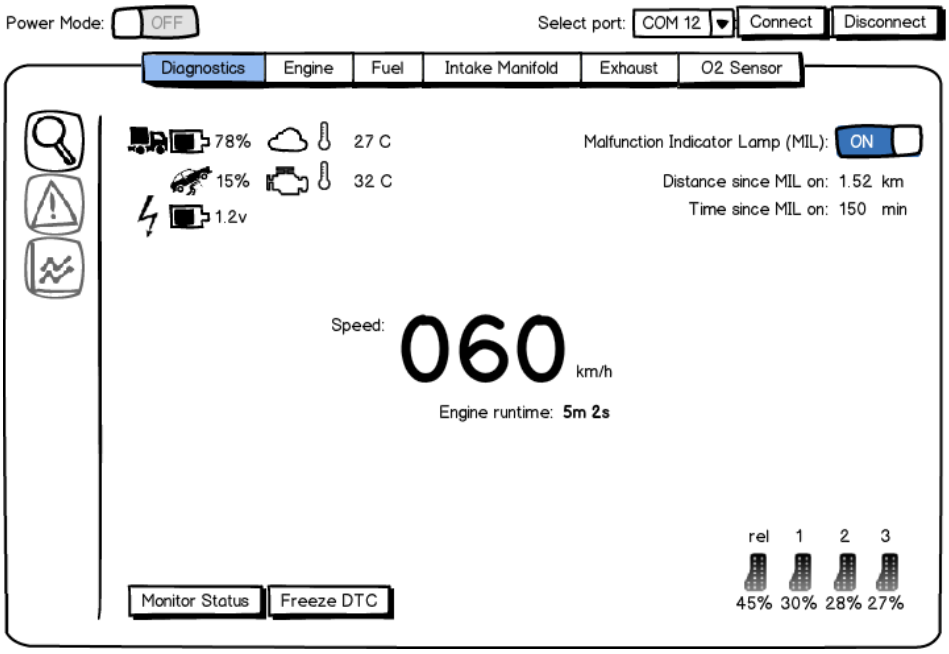


Gambar III-5 Rancangan Antarmuka Pengecekan PID

Setelah pengecekan seluruh PID yang terdapat dalam basis data sistem selesai, pengguna baru dapat menggunakan fitur-fitur lain yang disediakan oleh sistem. Cara untuk mulai menggunakan fitur-fitur yang ada yaitu dengan masuk ke dalam "**Power Mode**" dengan menekan tombol "Power Mode" yang ada pada bagian kiri atas antarmuka.

III.1.4.3 Spesifikasi Antarmuka Menu "Diagnostics"

Bagian ini akan menjelaskan spesifikasi antarmuka menu "**Diagnostics**". Diagnosa merupakan menu standar yang pertama kali ditampilkan oleh sistem ketika sistem dijalankan. Rancangan antarmuka utama dapat dilihat pada Gambar III-6.



Gambar III-6 Rancangan Antarmuka Utama Menu “Diagnostics”

Tampilan utama pada menu diagnosa juga merupakan tampilan yang dioptimasi untuk digunakan pada saat berkendara, karenanya antarmuka bagian ini dirancang untuk mudah dicerna secara visual. Informasi pada halaman ini dirancang untuk dapat diserap hanya dengan sapuan mata singkat.

Adapun informasi-informasi yang diberikan pada menu ini adalah informasi yang diperlukan pada saat berkendara untuk menjalankan *Drive Cycle* (alternatif lain untuk menjalanakn *Drive Cycle* adalah submenu grafik - bagian III.1.4.3.2), yaitu:

1. Sisa baterai mobil,
2. Tegangan baterai mobil,
3. Beban mesin,
4. Temperatur mesin,
5. Temperatur udara sekitar kendaraan,
6. Kecepatan kendaraan,
7. Waktu berjalannya kendaraan,
8. Posisi pedal (gas, rem, maupun perseneleng), dan

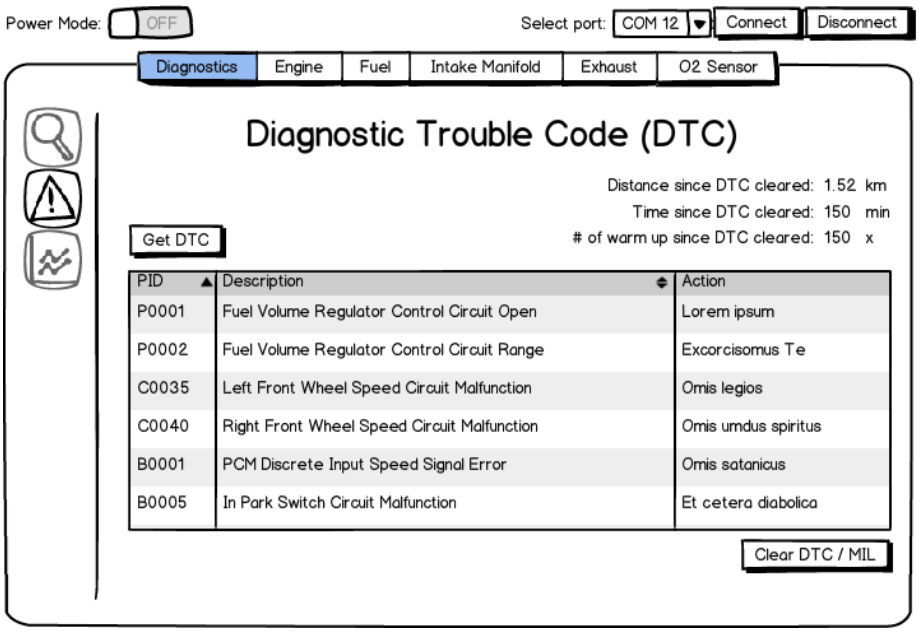
9. Informasi lampu indikator kerusakan (status: hidup atau mati, waktu, dan jarak berkendara sejak lampu hidup).

Selain itu, antarmuka menu juga memberikan dua tombol aksi kepada pengguna, yaitu "**Monitor Status**" dan "**Freeze DTC**". "**Monitor Status**" digunakan untuk memberikan perintah kepada ECU untuk mulai membaca sensor ECU, sementara "**Freeze DTC**" digunakan untuk mengambil informasi kesalahan yang tersimpan ketika pembacaan sensor dilakukan. Selain kedua tombol ini tidak terdapat komponen lain yang dapat memberikan respon kepada pengguna.

Selain tampilan utama, menu "**Diagnostics**" memiliki dua submenu lagi, yaitu informasi lain-lain dan DTC.

III.1.4.3.1 Spesifikasi Antarmuka Submenu DTC

Submenu DTC merupakan tempat untuk melihat informasi mengenai kode-kode kesalahan yang tersimpan dalam ECU. Rancangan antarmuka dari submenu ini dapat dilihat pada Gambar III-7.



Gambar III-7 Rancangan Antarmuka Submenu DTC

Terdapat dua informasi utama mengenai DTC yang disediakan oleh submenu ini, yaitu:

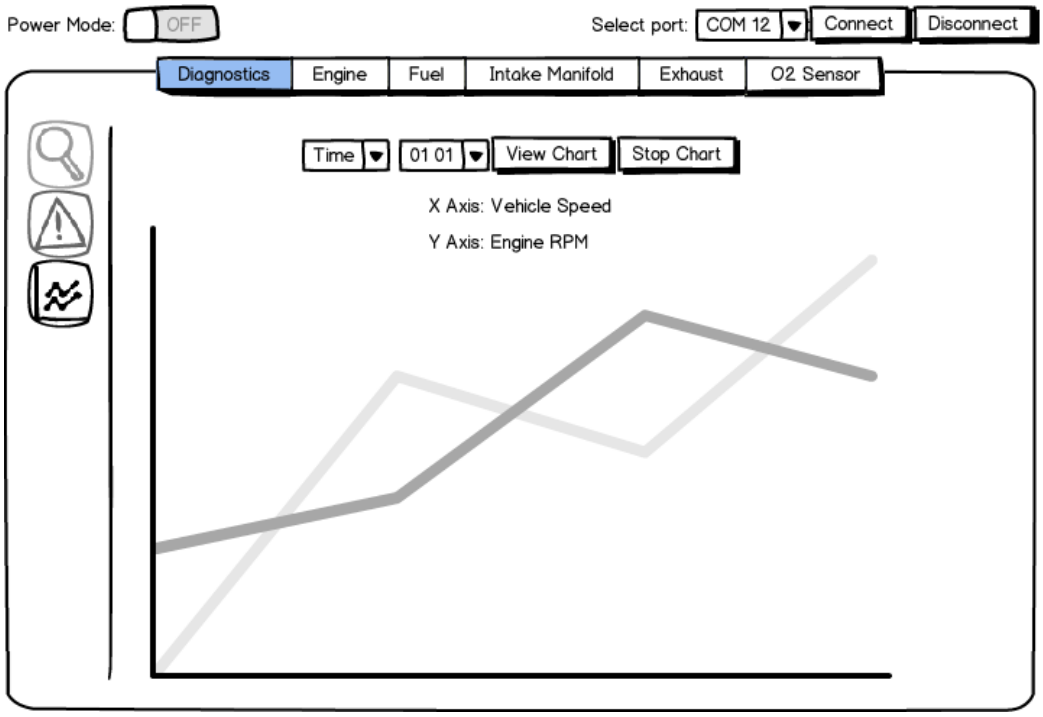
1. Berbagai pesan kesalahan yang tersimpan dalam ECU ditampilkan dalam tabel yang berada di tengah tampilan. Ketika submenu pertama kali dibuka, informasi ini tidak akan langsung ditampilkan (tabel kosong pada awal pembukaan menu). Tabel baru akan terisi setelah pengguna memberikan perintah melalui tombol "**Get DTC**".
2. Informasi jarak, waktu, dan jumlah pemanasan mesin sejak DTC terakhir kali dibersihkan.

Submenu DTC juga memberikan dua tombol aksi yang dapat digunakan oleh pengguna untuk memberikan perintah ke ECU, yaitu "**Get DTC**" dan "**Clear DTC / MIL**".

Seperti yang telah dijelaskan sebelumnya, tombol "**Get DTC**" akan mengambil kode kesalahan dari ECU dan menampilkannya dalam tabel yang disediakan. "**Clear DTC / MIL**" memberikan perintah kepada ECU untuk menghapus kode kesalahan yang tersimpan, serta mematikan lampu indikator kesalahan pada kendaraan. Perlu diingat bahwa tombol ini hanya akan menghapus kode, **tanpa memperbaiki kesalahan**. Idealnya tombol ini hanya digunakan setelah masalah pada mesin diselesaikan.

III.1.4.3.2 Spesifikasi Antarmuka Submenu Grafik

Submenu ini merupakan alternatif dari antarmuka menu "**Diagnostic**" standar, untuk pengguna yang ingin melihat pergerakan informasi mesin secara *real-time*. Gambar III-8 memperlihatkan rancangan antarmuka submenu grafik.



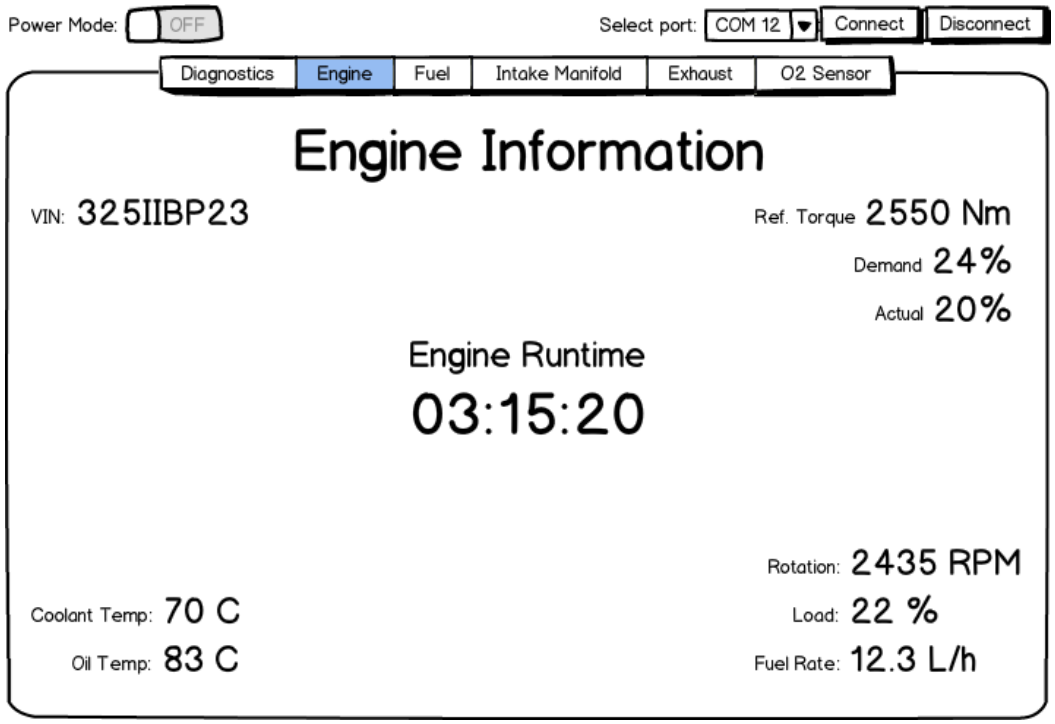
Gambar III-8 Rancangan Antarmuka Submenu Grafik

Informasi grafik yang ingin dilihat oleh pengguna, baik pada sumbu x maupun sumbu y dapat dipilih secara bebas. Waktu *update* dari data yang diberikan mesin adalah tergantung kepada kecepatan yang dapat diberikan oleh ECU.

III.1.4.4 Spesifikasi Antarmuka Menu "Engine"

Menu mesin memberikan informasi mengenai mesin secara lengkap. Karena informasi yang diberikan tidak dirancang untuk digunakan saat berkendara, lebih banyak informasi dapat dimasukkan dalam satu layar. Hal ini disebabkan oleh penggunaan teks sebagai media utama penampilan informasi. Banyaknya informasi yang dapat ditampilkan juga menyebabkan menu ini tidak memiliki submenu.

Rancangan antarmuka menu "**Engine**" dapat dilihat pada Gambar III-9.



Gambar III-9 Rancangan Antarmuka Menu "Engine"

Adapun informasi yang diberikan oleh menu ini adalah sebagai berikut:

1. nomor VIN mesin,
2. waktu berjalannya mesin,
3. temperatur pendingin mesin,
4. temperatur oli,
5. kecepatan rotasi mesin (rpm),
6. beban mesin,
7. laju penggunaan bahan bakar,
8. torsi standar mesin,
9. torsi yang diminta oleh pengendara, dan
10. torsi yang diberikan oleh mesin.

Menu "**Engine**" tidak menyediakan aksi kepada pengguna, sehingga menu ini bersifat pasif. Nilai-nilai yang ditampilkan kepada pengguna akan diperbaharui paling lambat setiap satu detik.

III.1.4.5 Spesifikasi Antarmuka Menu "Fuel"

Seperti namanya, menu ini memberikan informasi mengenai bahan bakar yang terdapat dalam kendaraan. Terdapat dua submenu pada menu "**Fuel**", yaitu:

1. Submenu informasi dasar yang menampilkan informasi mengenai bahan bakar dan penggunaannya. Submenu ini merupakan submenu yang pertama kali ditampilkan ketika pengguna memilih menu "**Fuel**".
2. Submenu tekanan yang memberikan informasi mengenai tekanan bahan bakar yang terdapat dalam kendaraan.

Kedua submenu di atas bersifat pasif, yang berarti submenu hanya menampilkan informasi kepada pengguna, tanpa menyediakan perintah-perintah tambahan yang dapat digunakan oleh pengguna. Detil dari kedua submenu tersebut akan dibahas pada bagian berikutnya.

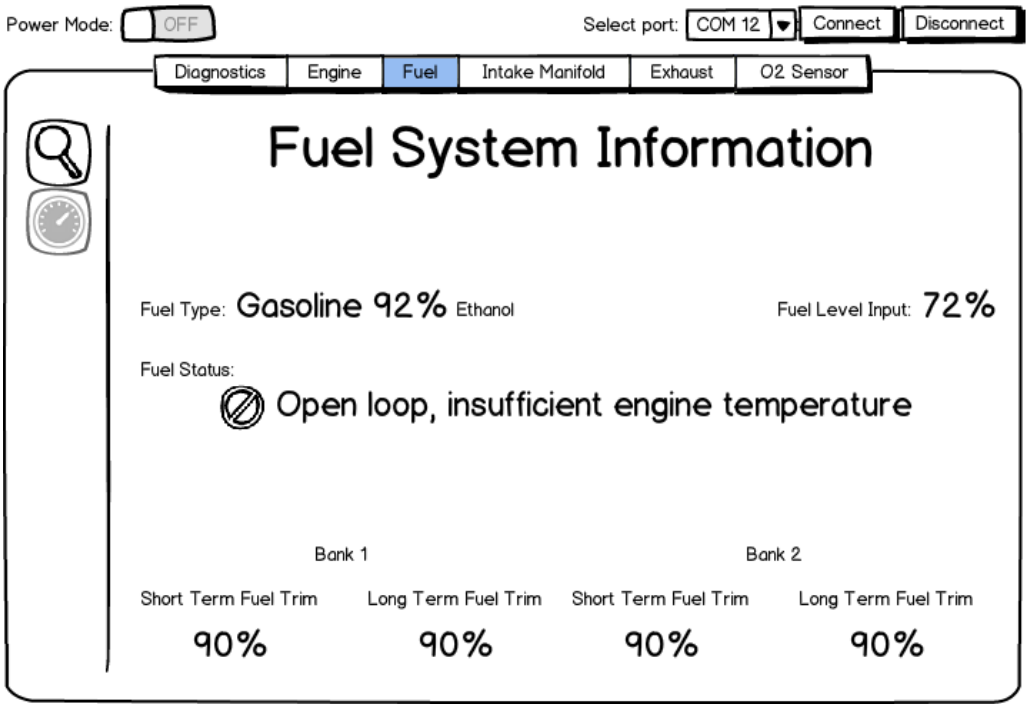
III.1.4.5.1 Spesifikasi Antarmuka Submenu Informasi Dasar

Selain informasi-informasi dasar mengenai bahan bakar, terdapat satu informasi paling penting yang ditampilkan pada submenu ini, yaitu informasi mengenai status pembakaran bahan bakar. Submenu ini akan menampilkan pesan status pembakaran, dengan informasi kemungkinan penyebabnya (jika ada).

Selain informasi tersebut, informasi yang ditampilkan pada submenu ini adalah:

1. Konsentrasi etanol dalam bahan bakar,
2. Tingkat masukan bahan bakar,
3. Penggunaan bahan bakar jangka pendek dan jangka panjang, pada kedua bilik (jika terdapat dua bilik).

Gambar III-10 menunjukkan rancangan antarmuka submenu informasi dasar.

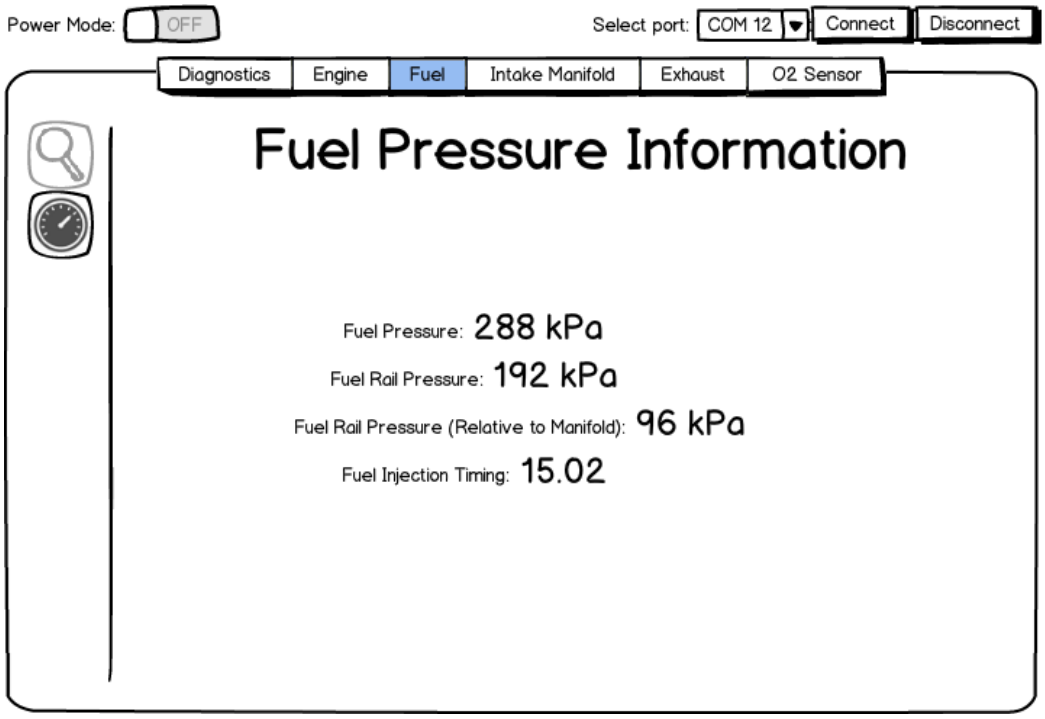


Gambar III-10 Rancangan Antarmuka Submenu Informasi Dasar Bahan Bakar

Sama seperti rancangan-rancangan sebelumnya, nilai-nilai yang dianggap penting oleh pengguna diberi ukuran huruf yang lebih besar ataupun simbol. Hal ini dilakukan untuk mempermudah pembacaan informasi serta sebagai petunjuk visual.

III.1.4.5.2 Spesifikasi Antarmuka Submenu Tekanan

Submenu tekanan hanya berisi sedikit informasi, yaitu: tekanan bahan bakar, tekanan rel bahan bakar, serta *timing* dari suntikan bahan bakar ke mesin. Rancangan antarmuka submenu tekanan dapat dilihat pada Gambar III-11.



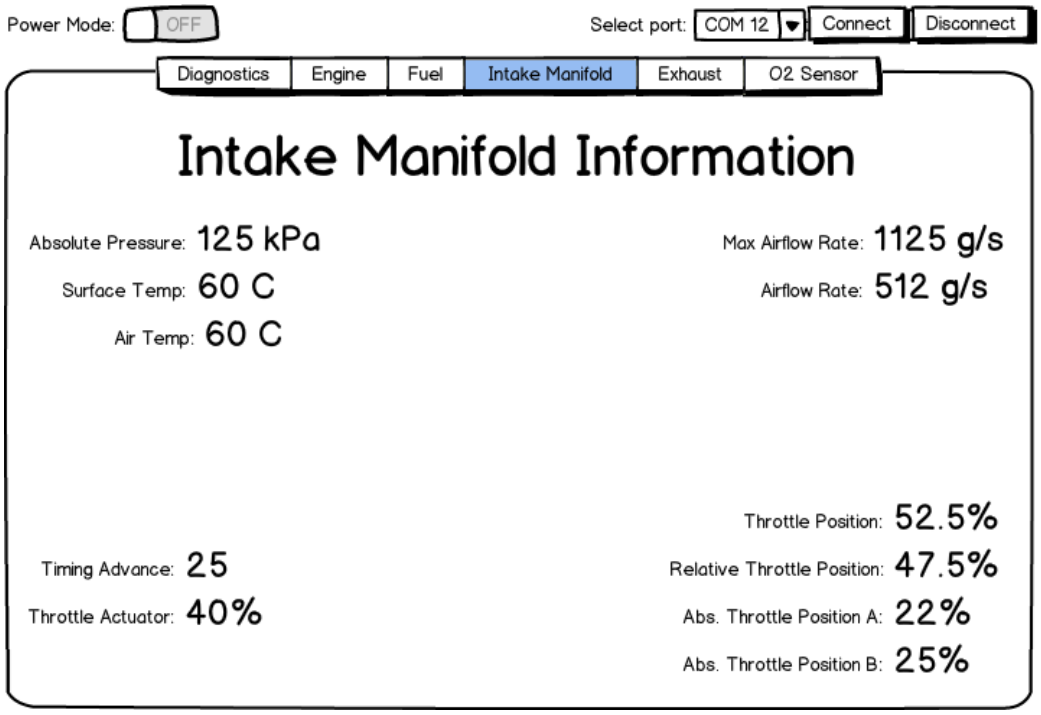
Gambar III-11 Rancangan Antarmuka Submenu Tekanan Bahan Bakar

III.1.4.6 Spesifikasi Antarmuka Menu "Intake Manifold"

Menu "**Intake Manifold**" memiliki kriteria yang sama dengan menu "**Engine**", di mana keduanya bersifat pasif dan tidak memiliki submenu. Adapun informasi-informasi yang ditampilkan oleh menu ini adalah:

1. Tekanan vakum pada *intake manifold*,
2. Temperatur permukaan *manifold*,
3. Temperatur udara di dalam *manifold*,
4. Laju aliran udara,
5. Laju aliran udara maksimal,
6. Posisi katup (absolut dan relatif), serta
7. Tempo dan posisi katup aktuator.

Gambar III-12 menunjukkan rancangan antarmuka menu "**Intake Manifold**".

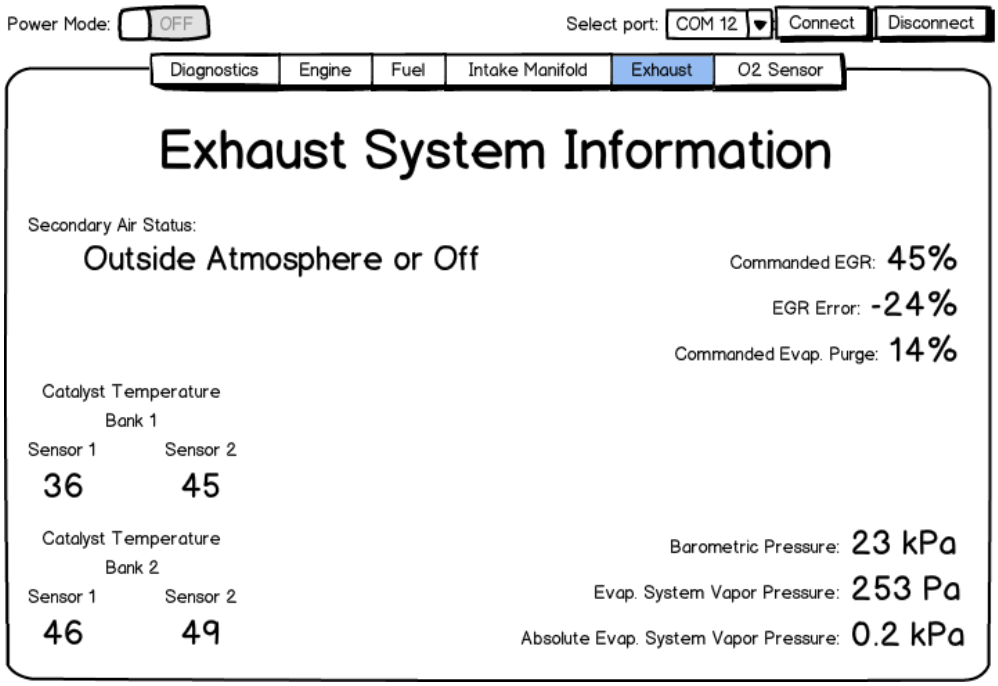


Gambar III-12 Rancangan Antarmuka Menu "Intake Manifold"

Kekosongan pada tengah antarmuka merupakan hal yang disengaja, karena tidak ada informasi yang dominan pada menu ini. Pengelompokan informasi akan lebih berguna dalam memudahkan pengguna mengamati informasi dibandingkan dengan "memaksakan" keberadaan informasi utama.

III.1.4.7 Spesifikasi Antarmuka Menu "Exhaust"

Menu "**Exhaust**" menampilkan informasi yang berhubungan dengan pembuangan mesin. Sama seperti menu "**Intake Manifold**", menu ini bersifat pasif dan tidak memiliki submenu. Gambar III-13 menunjukkan rancangan antarmuka dari menu "**Exhaust**".



Gambar III-13 Rancangan Antarmuka Menu "Exhaust"

Seperti yang terlihat pada Gambar III-13, informasi-informasi yang ditampilkan pada menu ini adalah sebagai berikut:

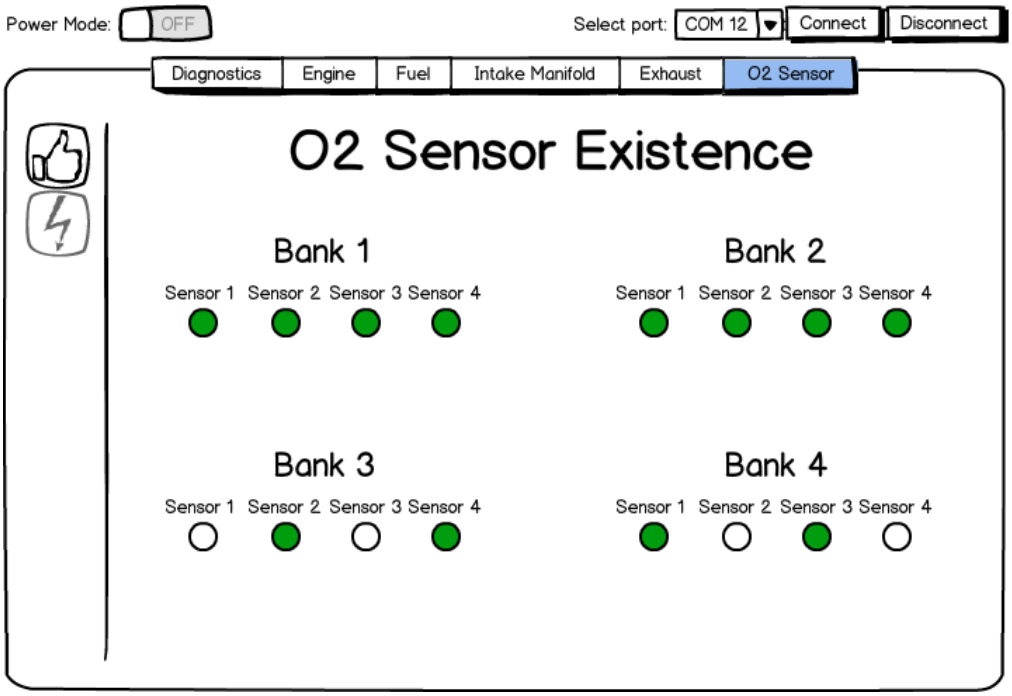
1. Status udara sekunder,
2. Perintah EGR (*Exhaust Gas Recirculation* – Sirkulasi Gas Buangan),
3. Tingkat kesalahan EGR,
4. Perintah pembersihan uap,
5. Tekanan barometrik (sistem pembuangan dan penguapan – absolut dan relatif), dan
6. Temperatur katalisator pada setiap sensor dan bilik.

III.1.4.8 Spesifikasi Antarmuka Menu "O_2 Sensor"

Menu "O_2 Sensor" memberikan informasi yang berhubungan dengan sensor oksigen yang ada pada kendaraan. Menu ini memiliki dua submenu, yaitu submenu keberadaan sensor serta submenu tegangan dan konsentrasi oksigen kendaraan.

III.1.4.8.1 Spesifikasi Antarmuka Submenu Keberadaan Sensor O_2

Seperti namanya, submenu ini menampilkan status keberadaan sensor oksigen pada kendaraan. Gambar menunjukkan rancangan antarmuka submenu ini.



Gambar III-14 Rancangan Antarmuka Submenu Keberadaan Sensor Oksigen

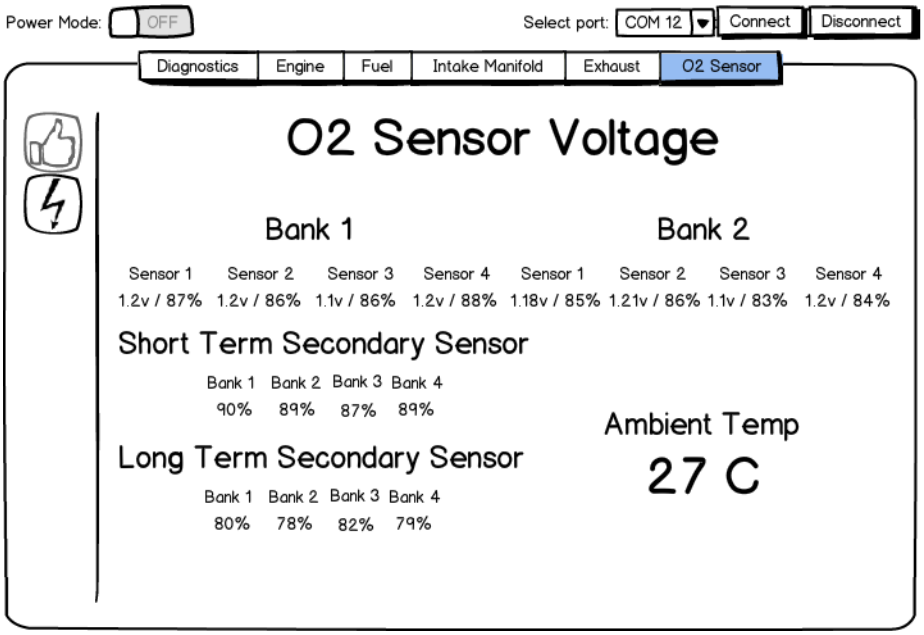
Untuk mempermudah pembacaan informasi, status keberadaan sensor pada setiap bilik ditampilkan dengan model lampu indikator. Lampu hijau berarti sensor ada, sementara lampu putih (tidak menyala) berarti sensor tidak ada.

Submenu ini tidak mengandung banyak informasi, untuk memberikan ruang terhadap model lampu indikator yang digunakan. Terlalu banyak informasi dalam kasus ini akan menyebabkan kesulitan pengguna melakukan perubahan konteks ketika menyerap informasi yang ditampilkan.

III.1.4.8.2 Spesifikasi Antarmuka Submenu Tegangan dan Konsentrasi O_2

Submenu ini memberikan informasi-informasi yang berhubungan dengan tegangan pada sensor oksigen, serta konsentrasi oksigen di dalam sensor tersebut. Adapun informasi-informasi yang diberikan ialah:

1. Informasi tegangan dan konsentrasi oksigen pada tiap sensor yang ada,
2. Konsentrasi oksigen pada sensor sekunder, dan
3. Temperatur udara sekitar kendaraan.



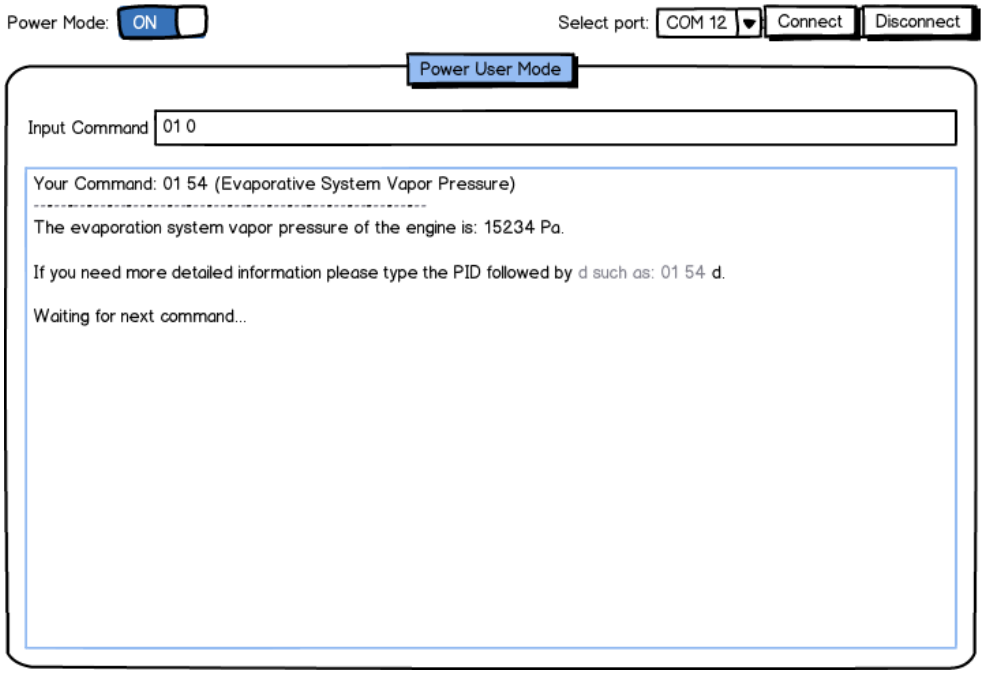
Gambar III-15 Rancangan Antarmuka Submenu Tegangan dan Konsentrasi Oksigen

Gambar III-15 memperlihatkan rancangan antarmuka submenu tegangan dan konsentrasi oksigen.

III.1.4.9 Spesifikasi Antarmuka Power Mode

Pengguna yang ingin mendapatkan kontrol lebih terhadap aplikasi, yang biasanya dikategorikan sebagai "*power user*" dapat mengaktifkan "**Power Mode**" untuk memberikan perintah langsung kepada ECU. Mode aplikasi ini hanya

memberikan tampilan terminal kepada pengguna, di mana pengguna dapat mengisikan perintah PID dan langsung mendapatkan respon dari mesin. Gambar III-16 memperlihatkan rancangan tampilan untuk “Power Mode”.



Gambar III-16 Rancangan Antarmuka Power Mode

Untuk mempermudah pengguna, tidak terdapat tombol untuk mengirimkan pesan ke ECU. Pengiriman pesan ke ECU cukup dilakukan pengguna dengan menekan kunci Enter pada *keyboard*. Setelah pengguna memasukkan perintah, sistem akan membalas dengan detil informasi dari PID jika PID adalah perintah OBD-II. Jika bukan merupakan perintah OBD-II, sistem hanya akan menampilkan respon yang diberikan oleh mesin tanpa perubahan (yang berarti respon diberikan dalam bentuk heksadesimal).

III.2 Perancangan Sistem

Bagian ini menjelaskan perancangan sistem, dari lingkungan pengembangan, garis besar komponen sistem sampai dengan detil dari tiap komponen yang membangun sistem.

III.2.1 Lingkungan Pengembangan Sistem

Sistem dikembangkan di atas lingkungan pengembangan (platform) Java. Java dipilih karena beberapa hal berikut:

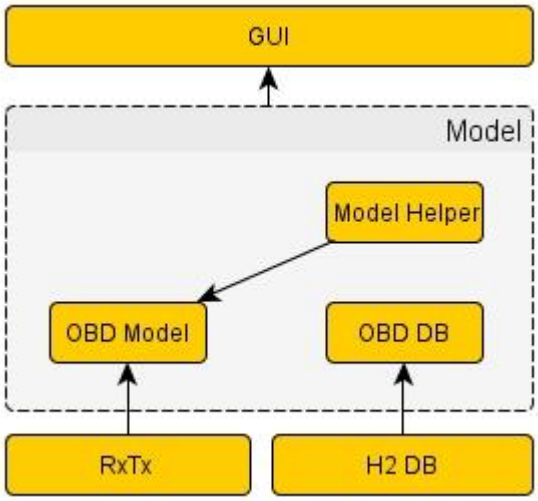
1. Portabilitas. Lingkungan pengembangan dan eksekusi Java dapat berjalan dalam banyak sistem operasi. Portabilitas penting karena target pengguna dari sistem, yang adalah pengguna antusias. Pengguna teknologi antusias umumnya menggunakan banyak sistem operasi, dengan banyak perangkat lunak lain yang spesifik pada sistem operasi tersebut. Pengembangan dalam lingkungan portabel akan memastikan sistem yang dibangun dapat bekerja dengan perangkat lain yang digunakan pengguna tersebut.
2. Kompatibilitas. Sebagai lingkungan pengembangan yang matang, Java memiliki banyak *library* maupun *framework* yang matang untuk mengembangkan sistem apapun. Kompatibilitas dengan Java akan sangat memudahkan pengembangan sistem.

Meskipun dikembangkan dalam lingkungan Java, sistem akan dibangun tidak menggunakan bahasa pemrograman Java, melainkan Scala. Alasan Scala digunakan yaitu akses ke paradigma pemrograman fungsional. Pemrograman fungsional digunakan untuk memudahkan pengkodean fungsi pembacaan respon PID, yang berbeda-beda untuk setiap PID. Pemrograman fungsional akan sangat menyederhanakan hal ini, sehingga akan mempermudah dan mempercepat pengembangan.

Selain akses ke pemrograman fungsional, akses ke pemrograman berorientasi objek juga akan sangat membantu dalam pengembangan aplikasi GUI (Graphical User Interface). Karena dukungan terhadap kedua paradigma yang baik inilah maka Scala dipilih sebagai bahasa pemrograman yang digunakan untuk membangun sistem.

III.2.2 Arsitekur Keseluruhan Sistem

Perangkat lunak diagnosa dikembangkan menggunakan arsitektur dua lapis sederhana, sesuai dengan arsitektur aplikasi standar yang disediakan oleh Scala. Lapisan-lapisan yang ada dalam sistem yaitu lapisan antarmuka (GUI) dan lapisan model. Lapisan antarmuka, seperti namanya mengatur bagian antarmuka dan interaksinya. Lapisan model melakukan abstraksi terhadap perangkat keras dan basis data untuk menurunkan *coupling* antara lapisan antarmuka dengan kedua komponen tersebut. Tidak terdapat lapisan controller seperti model arsitektur sistem pada umumnya, karena arsitektur standar yang diberikan Scala tidak menggunakan model arsitektur MVC (Model-View-Controller) klasik. Gambar III-17 menunjukkan arsitektur keseluruhan sistem.



Gambar III-17 Arsitektur Keseluruhan Sistem

Detil dari tiap-tiap komponen akan dijelaskan pada bagian berikutnya.

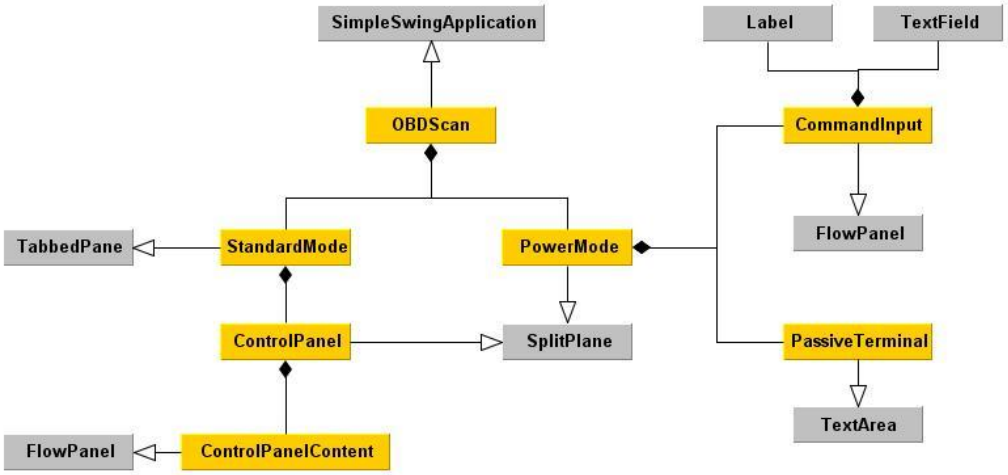
III.2.3 Rancangan Komponen Sistem

Bagian ini akan menjelaskan detil rancangan dari dua komponen utama, yaitu komponen antarmuka dan komponen model. Komponen antarmuka dirancang terutama menggunakan paradigma berorientasi objek, menggunakan UML.

Komponen model, di lain pihak, terdiri dari dua rancangan: rancangan basis data dan rancangan arsitektur aplikasi yang menggunakan paradigma fungsional.

III.2.3.1 Rancangan Komponen Antarmuka

Komponen antarmuka dirancang menggunakan paradigma berorientasi objek, dengan memanfaatkan *toolkit* Swing yang adalah perangkat standar milik Scala untuk mengembangkan aplikasi GUI. Gambar III-18 memperlihatkan *Class Diagram* dari komponen antarmuka sistem.



Gambar III-18 Class Diagram Komponen Antarmuka Sistem

Secara mendasar, aplikasi memiliki dua mode dengan antarmuka yang sangat berbeda, sehingga dirancang dua *Class* berbeda untuk mode tersebut. Standard Mode menggunakan `TabbedPane`, sementara Power Mode menggunakan `SplitPlane`. Pemilihan model antarmuka ini dilakukan sesuai dengan spesifikasi detil yang dijelaskan pada bagian III.1.4.

Antarmuka Power Mode sangat sederhana, yang hanya terdiri dari dua komponen utama. Hal ini menyebabkan antarmuka dibangun dengan `SplitPlane` sederhana, dengan dua komponen utama, yaitu komponen untuk mengisikan perintah dan menerima hasil eksekusi perintah tersebut.

Standard Mode, di lain pihak, merupakan antarmuka kompleks yang terdiri dari banyak komponen dan bentuk. Karenanya komponen ini dibangun dengan abstraksi yang lebih mendalam. Standar Mode memiliki banyak menu, yang masing-masing diakses melalui *tab*, menyebabkan *Class* `StandardMode` dibentuk dari `TabbedPane`.

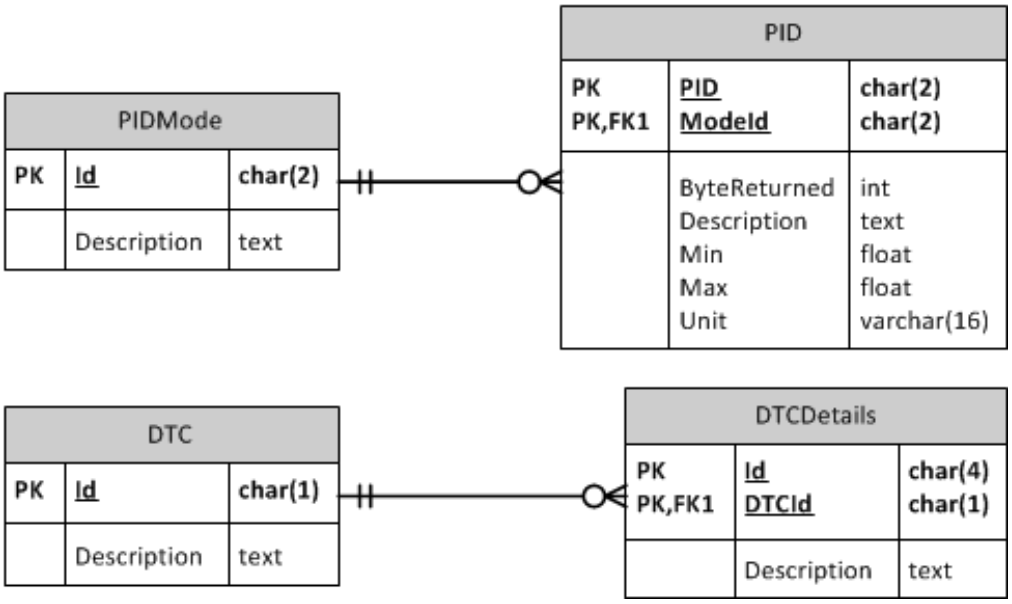
`StandardMode` memiliki `ControlPanel`, yang menampung submenu dari pilihan menu `StandardMode` yang sedang aktif, dan menampilkan isi dari submenu tersebut. Isi dari submenu ditampilkan oleh `ControlPanelContent`. Abstraksi dari `SplitPlane` dan `FlowPanel` dilakukan agar pemilihan dan pergantian menu dapat dilakukan dengan mudah (melalui fitur *pattern matching* pada Scala).

III.2.3.2 Rancangan Komponen Model

Komponen model terdiri dari tiga bagian, yaitu bagian yang berinteraksi dengan basis data H2, komponen yang berinteraksi dengan perangkat keras (RS 232) secara langsung, serta komponen yang bertanggung jawab dalam manajemen berkas penyimpanan data PID yang didukung. Basis data dibutuhkan untuk menyimpan dan membaca deskripsi DTC dan PID, karena jumlah DTC dan PID yang cukup banyak (total lebih dari 3000; dapat dibaca pada dokumentasi OBD-II (SAE International, 2002)) untuk diproses dengan struktur data dalam memori. Penyimpanan data PID yang didukung tidak dilakukan di dalam basis data agar pengguna dapat langsung mengubah berkas jika dibutuhkan.

III.2.3.2.1 Komponen Basis Data

Untuk mempermudah pemasangan, sistem yang dikembangkan menggunakan basis data *embedded* H2, yang dapat langsung digunakan oleh pengguna tanpa melakukan konfigurasi tambahan. Rancangan basis data yang digunakan oleh sistem untuk menyimpan DTC dan PID dapat dilihat pada Gambar III-19.



Gambar III-19 Rancangan Basis Data Penyimpanan DTC dan PID

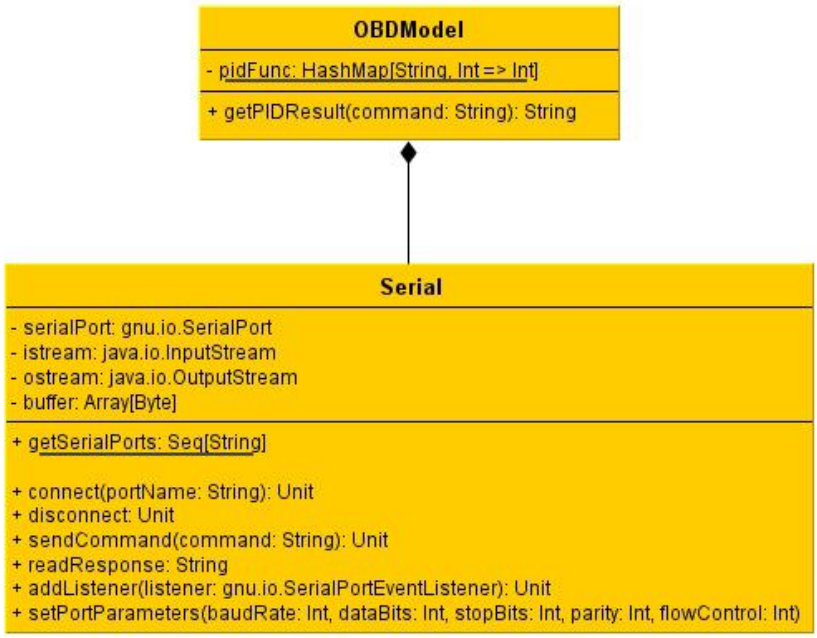
Seperti yang dapat dilihat pada Gambar III-19, basis data yang digunakan dirancang sangat sederhana. Hal ini disebabkan oleh penggunaan basis data *embedded* yang memiliki tipe data dan fitur yang terbatas, serta kriteria performa yang sangat terbatas. Penyerdehanaan model data akan membantu dalam meningkatkan performa dari basis data jenis ini.

Tipe dari *primary key* dipilih sedemikian rupa agar *query* dapat dilakukan secara literal. Pada tabel `PIDMode` misalnya, *primary key* diberi tipe `CHAR(2)` agar *query* dapat dilakukan secara langsung menggunakan nilai yang sama dengan PID (“01”, “02”, dst). Hal ini terutama akan sangat berdampak pada kode sistem di bagian “**Power Mode**”, di mana pengguna dapat memasukkan PID dan mendapatkan informasi mengenai PID tersebut. Rancangan basis data seperti ini akan sangat menyederhanakan kasus penggunaan tersebut.

III.2.3.2.2 Komponen Model RxTx

Komponen model bagian ini melakukan abstraksi terhadap akses port serial (RS 232), pengiriman PID ke mesin, serta pembacaan respon yang diberikan oleh mesin. Karena respon yang diberikan oleh mesin harus dibaca dengan cara yang

berbeda-beda sesuai dengan PID-nya maka komponen ini dirancang dengan paradigma pemrograman fungsional. Begitupun, untuk mempermudah komunikasi dengan bagian lain dari sistem, abstraksi akses RS232 ini dimasukkan ke dalam dua buah *Class*, yang dapat dilihat pada Gambar III-20.



Gambar III-20 Class Diagram Abstraksi SerialPort

Class `Serial`, sesuai dengan namanya, merupakan *Class* yang berinteraksi langsung dengan RS232. `OBDModel` adalah *Class* yang digunakan oleh sistem untuk mengambil hasil perintah PID yang dikirimkan ke mesin. Atribut `pidFunc` merupakan sebuah *hash map* yang menyimpan fungsi pembaca (*decoder*) respon dan PID yang digunakan. Atribut ini bersifat statik sehingga hanya terdapat satu instan dan tidak memakan memori (mengingat banyaknya PID dan fungsi yang ada).

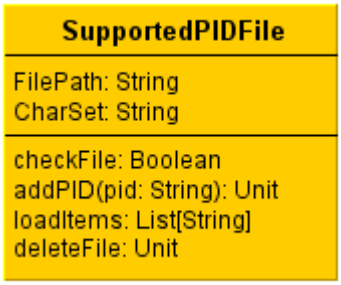
III.2.3.2.3 Komponen Manajemen Berkas PID yang Didukung

Komponen ini memiliki peran utama untuk berinteraksi dengan berkas yang menyimpan data PID yang didukung oleh ECU yang digunakan sistem sebelumnya. Hanya terdapat satu kelas untuk menangani hal ini, yaitu kelas

`SupportedPIDFile`. Sesuai dengan namanya, kelas ini menyimpan data PID yang didukung ke dalam sebuah file.

Penyimpanan dilakukan di dalam file teks sederhana yang dapat dibuka dengan teks editor manapun. Hal ini dilakukan agar pengguna dapat dengan mudah melakukan perbaikan dan perubahan jika memang dibutuhkan, ataupun agar pengguna dapat menuliskan sendiri file ini jika pengguna mengetahui persis perintah-perintah yang didukung oleh ECU-nya. Isi dari file hanyalah PID yang didukung, dengan format satu baris per satu PID.

Adapun diagram kelas dari `SupportedPIDFile` dapat dilihat pada Gambar III-21.



Gambar III-21 Diagram Kelas dari `SupportedBinaryFile`

Fitur-fitur dalam kelas ini cukup jelas, sesuai dengan yang dapat dilihat pada diagram kelas, yaitu: pengecekan keberadaan berkas, penambahan data PID, pengambilan data, dan penghapusan berkas. Penambahan data PID secara otomatis akan menuliskan berkas baru jika berkas tidak ada, dan menambahkan data ke dalam berkas jika berkas ada.

Bab IV Konstruksi, Hasil, dan Evaluasi

Bab ini menjelaskan metode pengembangan yang digunakan untuk konstruksi perangkat lunak, serta hasil dari pengembangan perangkat lunak tersebut. Evaluasi terhadap hasil pengembangan juga dilakukan, untuk memastikan tujuan pengembangan telah tercapai.

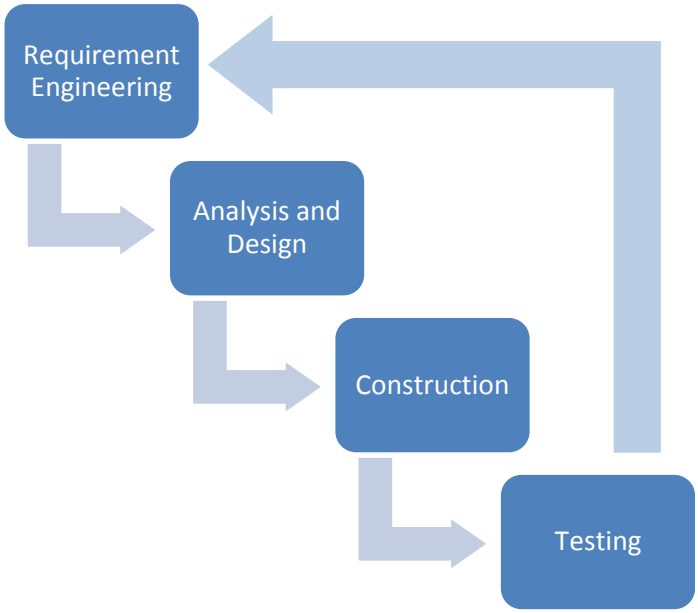
IV.1 Konstruksi Perangkat Lunak

Pembangunan perangkat lunak dilakukan menggunakan metode klasik *waterfall* (Sommerville, 2007). *Waterfall* digunakan untuk memastikan perancangan arsitektur sistem dan struktur data dilakukan dengan benar. Rancangan arsitektur sistem dan struktur data yang benar artinya rancangan yang dihasilkan memiliki duplikasi minimal (*reusable*), memiliki pembagian tugas yang jelas (*modular*), serta sederhana dan mudah dimengerti (*readable* dan *maintainable*). Ketepatan arsitektur sistem dan struktur data sangat penting mengingat:

1. **Besarnya standar OBD-II**. OBD-II merupakan standar yang besar dan rumit, memiliki total 4 protokol komunikasi, dan lebih dari 180 perintah (PID), dengan ribuan pesan kesalahan. Kesalahan rancangan atau arsitekur sistem akan menghasilkan sistem yang memiliki banyak duplikasi atau sistem yang sangat kompleks, dan akan berdampak pada *readability* dan *maintainability* sistem. Karenanya, dilakukan analisa dan perancangan terhadap arsitektur sistem dan struktur data secara menyeluruh terlebih dahulu, untuk memastikan tidak terdapat kesalahan fatal ketika sistem dibangun.
2. **Perbedaan Rumus Perhitungan Respon Setiap PID pada OBD-II**. Karena perhitungan setiap respon yang melakukan pembacaan data dari sensor yang berbeda-beda, hampir setiap PID OBD-II memiliki rumus perhitungan dan jumlah byte respon yang berbeda-beda. Kesamaan hanya ditemui pada pembacaan nilai sensor sejenis, misalnya untuk sensor temperatur (udara, permukaan mesin, dll), byte kembalian akan selalu bernilai 1 byte, dan rumus perhitungan akan selalu A – 40. Pembacaan nilai sensor lain akan memerlukan rumus dan nilai byte kembalian yang berbeda. Hal ini tentu menyebabkan

pembacaan dan kalkulasi respon OBD-II akan memerlukan perancangan yang hati-hati, untuk memastikan struktur data dan arsitektur yang digunakan dapat mencakup seluruh kasus. Jika struktur data dan arsitektur gagal mencakup seluruh kasus, tentunya perawatan kode akan menjadi sulit dilakukan, karena kode akan memerlukan banyak percabangan dalam menghitung nilai respon.

Pengembangan dilakukan sesuai dengan metode klasik *waterfall*: elisitasi kebutuhan, analisa dan perancangan sistem yang diikuti dengan konstruksi dan pengujian sistem. Alur pengembangan yang dilakukan dapat dilihat pada Gambar IV-1.



Gambar IV-1 Gambaran Alur Proses Pengembangan

Analisis kebutuhan dilakukan dengan mencari kebutuhan pengguna berdasarkan wawancara dan mencoba perangkat lunak sejenis yang telah ada di pasaran. Setelah mendapatkan gambaran mengenai kebutuhan pengguna, dilakukan juga analisis terhadap standar OBD-II, untuk memastikan tidak terdapat kebutuhan pengguna yang sulit atau tidak mungkin diimplementasikan dengan standar OBD-II.

Kebutuhan pengguna menghasilkan rancangan awal antarmuka sistem. Melalui rancangan awal ini, maka dilakukan analisa dan perancangan perangkat lunak, untuk memastikan konstruksi dapat berjalan dengan baik. Analisa dan perancangan juga mencakup tahap peninjauan dokumen standar OBD-II, terutama di protokol komunikasi dan format data kembalian dari ECU. Kedua hal ini sangat penting untuk dianalisa dan dirancang dengan benar, karena kesalahan rancangan pada bagian ini akan menyebabkan perubahan rancangan keseluruhan yang sangat besar.

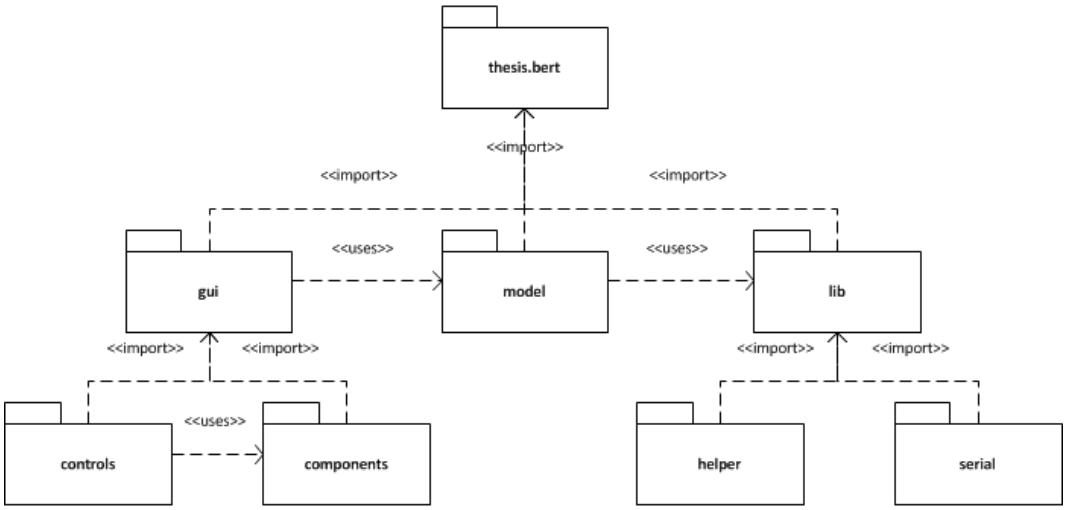
Selesai perancangan, konstruksi kemudian dilakukan, mulai dari pengembangan komponen yang bertanggung jawab dalam berkomunikasi dan membaca nilai kembalian OBD-II, sampai dengan lapisan antarmuka sistem. Pengembangan dilakukan seluruhnya sekaligus, untuk kemudian diujicobakan. Jika terjadi kesalahan pada bagian pengembangan, maka proses akan diulangi lagi dari elisitasi kebutuhan. Proses pengujian dan pengukuran keberhasilan pengujian dapat dibaca pada IV.3.

IV.2 Hasil Pengembangan

Bagian ini akan menjelaskan mengenai hasil dari pengembangan yang dilakukan, dari sisi arsitektur sistem dan struktur data. Analisa dan rancangan yang mendasari pengembangan dapat dibaca pada Bab III. Pembahasan akan dilakukan terlebih dahulu terhadap arsitektur keseluruhan, kemudian detil dari tiap-tiap bagian akan dibahas satu per satu.

IV.2.1 Arsitektur Keseluruhan Sistem

Rancangan sistem bersifat hirarkis, yang berarti terdapat pembagian tingkatan (lapisan) kelas, dengan tingkatan paling atas adalah komponen antarmuka, dan lapisan paling bawah ialah komponen yang berhubungan dengan perangkat keras atau basis data. Diagram paket sistem, yang menunjukkan hirarki lapisan sistem dan seluruh paket sistem dapat dilihat pada Gambar IV-2.



Gambar IV-2 Diagram Paket Sistem

Paket komponen teratas yaitu `thesis.bert`, yang berisi kelas utama (*Main Class*) pada sistem. Di bawah paket ini, terdapat tiga paket yang saling berhubungan, yaitu `gui`, `lib`, dan `model`. Paket `gui`, seperti namanya, bertanggung jawab dalam membangun antarmuka pengguna. Paket `lib` terdiri dari kelas-kelas utilitas, yang membantu paket-paket lain untuk melakukan berbagai hal (misalnya: pembacaan nilai biner, pengiriman perintah ke serial port). Paket `model` adalah lapisan model, yang berkomunikasi dengan basis data dan melakukan pembacaan nilai kembalian dari OBD-II. Untuk lebih mudahnya, Tabel IV-1 memberikan deskripsi singkat kegunaan dari masing-masing paket.

Tabel IV-1 Kegunaan dari Setiap Paket dalam Sistem

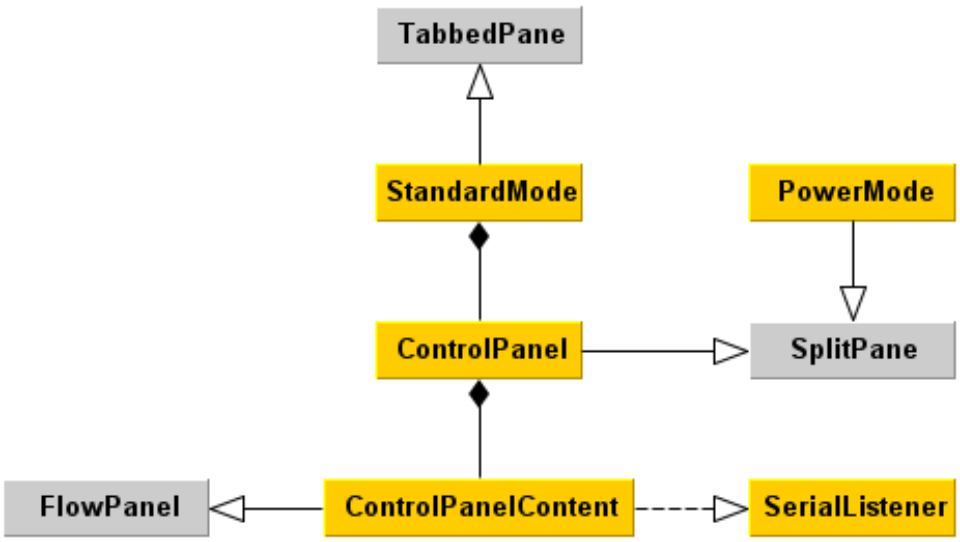
Paket		Kegunaan
`thesis.bert`	=	Kelas utama, titik awal program.
`thesis.bert.gui`	=	Penampung antarmuka (UI Container).
`thesis.bert.gui.controls`	=	Isi dari penampung utama.
`thesis.bert.gui.components`	=	Komponen-komponen antarmuka *custom* yang digunakan oleh sistem.
`thesis.bert.model`	=	Akses database serta perintah OBD-II.
`thesis.bert.lib`	=	Paket penampung kelas utilitas.
`thesis.bert.lib.serial`	=	Penghubung ke port serial.
`thesis.bert.lib.helper`	=	Kelas utilitas untuk berbagai keperluan.

IV.2.2 Lapisan Antarmuka Sistem

Lapisan antarmuka pada sistem memiliki tiga paket utama, yaitu paket `gui`, paket `gui.controls`, dan paket `gui.components`. Paket `gui` merupakan paket utama, yang berisi penampung antarmuka, yang akan digunakan oleh kelas utama. Paket `gui.controls` bertanggung jawab dalam membangun antarmuka sistem, sementara paket `gui.components` berisi berbagai komponen *custom* yang digunakan oleh sistem.

IV.2.2.1 Paket `gui`

Paket `gui` terdiri dari empat kelas, yang dapat dilihat pada Gambar IV-3.



Gambar IV-3 Diagram Kelas pada Paket `thesis.bert.gui`

Karena terdapat dua mode utama dari sistem dengan tampilan yang sangat berbeda, maka terdapat dua kelas penampung untuk masing-masing mode. Seperti namanya, `StandardMode` merupakan kelas untuk menampung antarmuka pada mode standar, sementara `PowerMode` merupakan kelas yang digunakan untuk menampung antarmuka pada mode terminal (*Power Mode*). Karena `PowerMode`

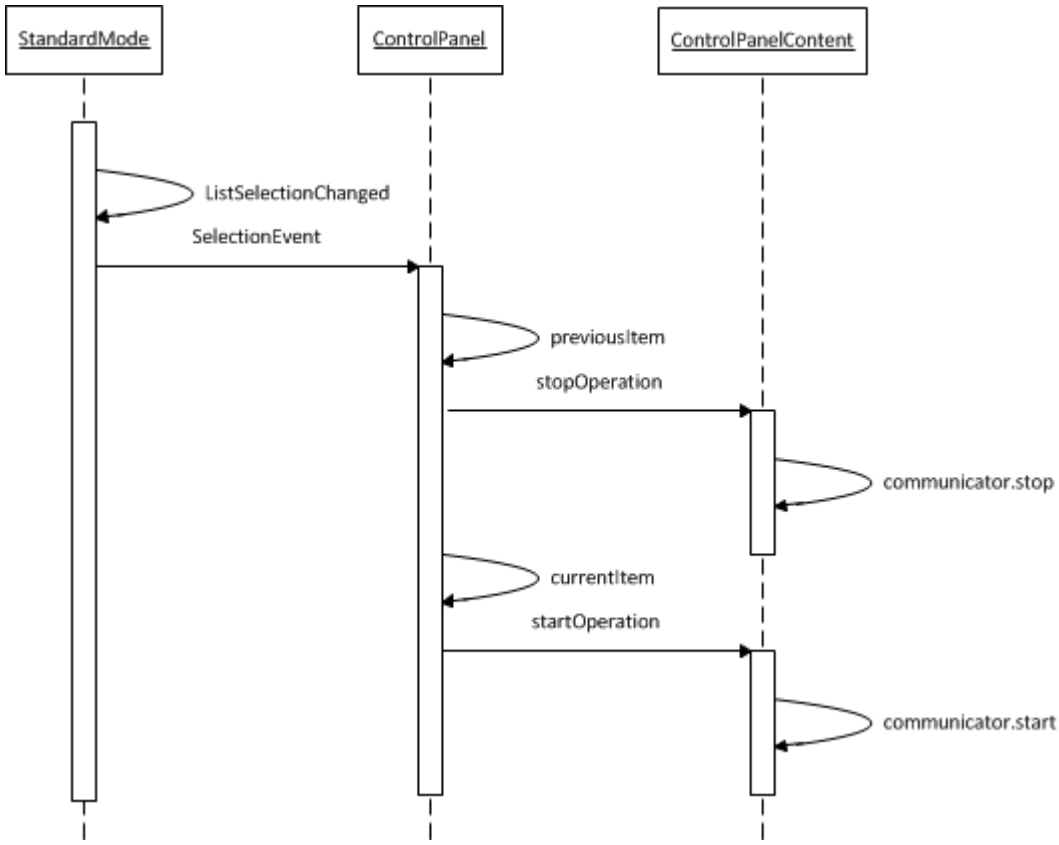
yang bentuknya sangat sederhana (lihat Gambar III-16) maka penjelasan detil hanya dilakukan terhadap `StandardMode`.

`StandardMode` merupakan mode yang menampilkan informasi mesin dari ECU, yang diambil secara periodik, secara terus menerus. Hal ini mengharuskan `StandardMode` dapat berkomunikasi dengan ECU (melalui port serial) secara periodik. Karena `StandardMode` memiliki beberapa halaman, di mana setiap halamannya memberikan informasi yang berbeda-beda, maka komunikasi dengan ECU tidak dapat dilakukan secara langsung pada `StandardMode`. Memberikan peran komunikasi dengan ECU kepada `StandardMode` akan menyebabkan ukuran `StandardMode` menjadi sangat besar, karena `StandardMode` menjadi harus menampung seluruh perintah yang ada, dan menampilkannya sesuai dengan halaman yang sedang dibuka pengguna, dan lalu berkomunikasi dengan ECU dan melakukan pembaruan antarmuka ketika ECU telah memberikan respon. Singkatnya, `StandardMode` akan memiliki terlalu banyak tanggung jawab dan kompleks. Hal ini akan berdampak besar pada maintainability sistem nantinya.

Untuk menyederhanakan `StandardMode`, peran komunikasi dengan ECU diberikan kepada `ControlPanelContent`. `StandardMode` hanya bertanggung jawab dalam memantau halaman yang sedang aktif, dan memberikan perintah kepada `ControlPanel`, yang kemudian meneruskannya ke `ControlPanelContent`, untuk mulai berkomunikasi dengan ECU, sesuai dengan bagian mana yang sedang dilihat oleh pengguna. `ControlPanelContent` kemudian akan melakukan pengiriman perintah ke ECU dan pembaruan tampilan dengan informasi baru ketika ECU telah memberikan respon. Peran dari `ControlPanel` sendiri adalah sebagai wadah untuk menampung beberapa `ControlPanelContent`, dan melakukan pemberian perintah kepada `ControlPanelContent` untuk mulai berkomunikasi.

Sederhananya, `StandardMode` berfungsi untuk memantau `ControlPanel` yang sedang aktif, untuk memberikan perintah kapan mulai mengirimkan kode ke

ECU dan kapan harus berhenti. `ControlPanel`, di lain pihak, berfungsi untuk memantau `ControlPanelContent` yang sedang aktif, untuk memberikan perintah kapan mulai mengirimkan kode dan kapan berhenti. Agar memperjelas, Gambar IV-4 memperlihatkan *sequence diagram* dari ketiga komponen ini.



Gambar IV-4 Hubungan Antara Komponen dalam Paket `thesis.bert.gui`

Setelah mendapatkan pesan "`startOperation`", `ControlPanelContent` kemudian akan mengirimkan perintah-perintah yang relevan dengan halaman yang sedang ditampilkan ke port serial. Perintah-perintah ini disimpan di dalam properti `commands` pada `ControlPanelContent`. Pengiriman perintah kemudian dilakukan secara terus-menerus, dengan memanggil `commands` secara sirkular (berputar). Untuk mempermudah pemahaman, Listing IV-1 memperlihatkan bagaimana pengiriman kode secara sirkular dilakukan.

```
val commands: Seq[String]

val communicator = actor {
  var currentCommand = 0

  loop {
    react {
      case 'start => {
        val commandLength = commands.length - 1
        currentCommand = if (currentCommand == commandLength)
                           0
                         else
                           currentCommand + 1
        serialPort.messenger ! commands(currentCommand)
      }
      case 'stop => exit
    }
  }
```

Listing IV-1 Pengiriman Kode Secara Sirkular

Pada Listing IV-1, `communicator` merupakan properti yang akan melakukan pengiriman perintah ke port serial. Perintah disimpan pada properti `commands`, dan diambil secara terurut melalui variabel `currentCommand`. Jika variabel `currentCommand` telah bernilai sama dengan panjang dari `commands`, maka nilai variabel `currentCommand` akan dikembalikan ke `0`. Hal inilah yang menyebabkan pemanggilan dapat berulang kembali ke perintah pertama pada `commands`, sehingga perintah dikirimkan secara sirkular.

Perintah untuk menjalankan `communicator` kemudian dijalankan secara periodik, sesuai dengan batas bawah pengiriman perintah yang diperbolehkan oleh port serial. Pemanggilan ini dilakukan melalui `actor` lainnya, yang khusus dibuat untuk melakukan penjadwalan pemanggilan `communicator`. Detil implementasi dari `actor` ini dapat dilihat pada Listing IV-2. Method "`startOperation`" pada Listing IV-2 ini adalah merupakan yang dipanggil oleh `ControlPanel` ketika ingin memberi tahu `ControlPanelContent` untuk mulai mengirimkan pesan ke ECU.

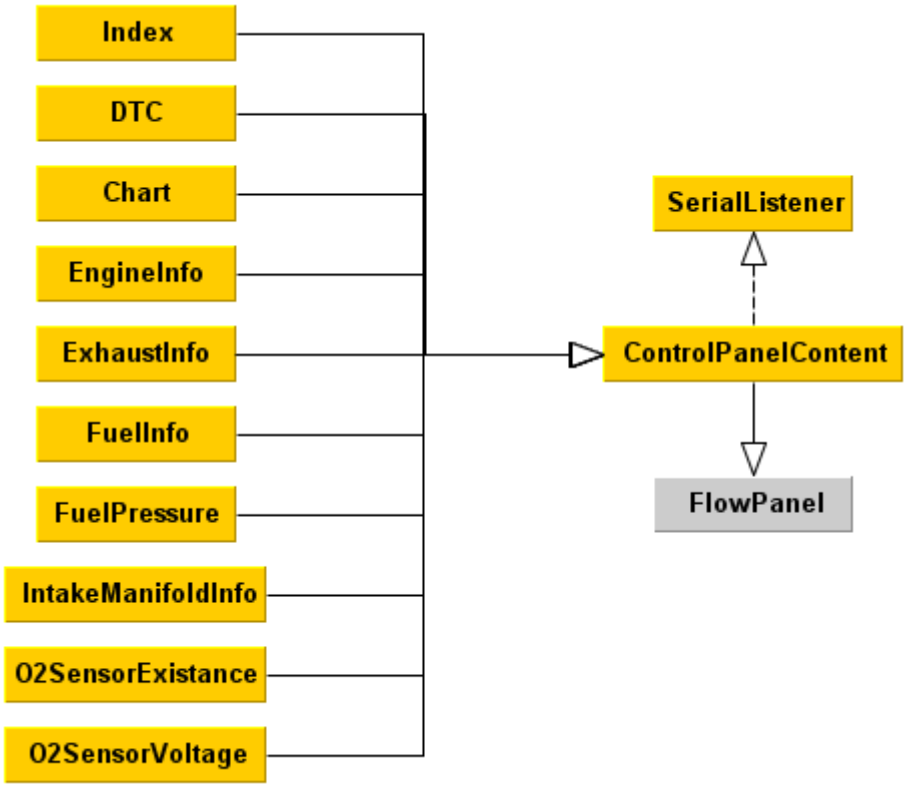
```
protected def scheduler(time: Long)(f: => Unit) = {
  def fixedRateLoop {
    Actor.reactWithin(time) {
      case TIMEOUT => f; fixedRateLoop
      case 'stop   => exit
    }
  }
  Actor.actor(fixedRateLoop)
}

protected var sched: Actor = _
def startOperation = {
  if (sched != null) {
    sched.restart
    communicator.restart
  } else {
    sched = scheduler(Serial.SerialPortDelay) {
      communicator ! 'start
    }
  }
}
```

Listing IV-2 Pemanggilan `communicator` Secara Periodik

Pembahasan lengkap mengenai komunikasi kelas `ControlPanelContent` dengan port serial tidak akan dibahas lebih lanjut pada bagian ini, karena tidak relevan dengan antarmuka. Pembahasan mengenai komunikasi komponen sistem dengan port serial akan dilakukan pada bagian IV.2.4.2.

IV.2.2.2 Paket `gui.controls`

Paket gui.controls merupakan paket yang terdiri dari implementasi sistem terhadap halaman-halaman sistem yang telah dirancang pada bagian III.2.3.1. Komponen-komponen yang terdapat dalam bagian ini memiliki dua tanggung jawab, yaitu menampilkan antarmuka dan memperbaharui tampilan, sesuai dengan nilai kembalian dari ECU. Gambar IV-5 memperlihatkan diagram kelas dari paket `gui.controls`.



Gambar IV-5 Diagram Kelas pada Paket `thesis.bert.gui.controls`

Karena kesederhanaan kelas ini, maka tidak akan dilakukan penjelasan lebih lanjut. Pembaruan antarmuka dilakukan dengan standar (mengubah properti dari komponen antarmuka yang terkait), sedangkan pengiriman perintah-perintah pada setiap halaman telah dibahas pada bagian IV.2.2.1.

IV.2.2.3 Paket `gui.components`

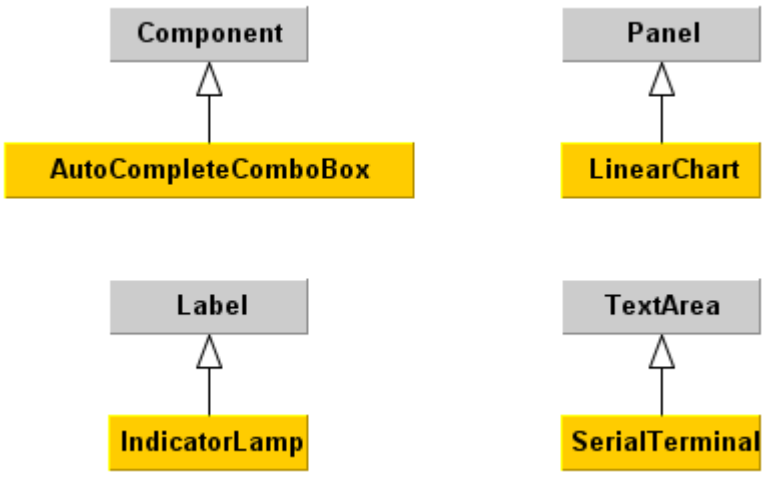
Paket `gui.components` berisi komponen-komponen buatan sendiri yang digunakan oleh sistem. Komponen-komponen yang dibuat sendiri biasanya merupakan komponen yang tidak terdapat pada pustaka standar `scala.swing`, ataupun komponen-komponen yang memerlukan perilaku khusus pada sistem. Jika merupakan komponen yang tidak dimiliki oleh pustaka standar, implementasi dilakukan hanya dengan menuliskan *wrapper* sederhana terhadap komponen pihak ketiga (yang biasanya diimplementasikan dengan menggunakan Java).

Komponen yang memerlukan perilaku khusus akan langsung melakukan subclass terhadap komponen inti, dan mengimplementasikan perilaku yang diinginkan.

Terdapat total empat buah komponen yang dikembangkan, yaitu:

1. `AutoCompleteComboBox`, sebuah komponen yang memberikan fitur pengetikan dan *auto complete* terhadap *combo box*. Komponen ini digunakan pada berbagai *combo box* yang ada dalam sistem, seperti pada Power Mode dan pemilihan nilai sumbu x dan y pada tampilan grafik.
2. `LinearChart`, yaitu komponen yang berguna untuk menampilkan *chart*. Komponen ini merupakan *wrapper* sederhana dari `jfreechart`.
3. `IndicatorLamp`, merupakan komponen yang digunakan untuk mengindikasikan status dari sebuah komponen dalam mesin. Diimplementasikan sebagai sub-komponen dari `Label`, yang hanya menampilkan ikon, sesuai dengan status yang diinginkan (hidup atau mati).
4. `SerialTerminal`, merupakan `TextArea` yang hanya dapat menampilkan teks, sesuai dengan nilai respon dari port serial.

Gambar IV-6 memperlihatkan diagram kelas dari paket `gui.components`.



Gambar IV-6 Diagram Kelas pada Paket `thesis.bert.gui.components`

IV.2.3 Lapisan Model Sistem

Lapisan model, yang berada pada paket `thesis.bert.model` memiliki hanya tiga kelas, yaitu `OBDDB`, `OBDModel`, dan `SupportedPIDFile`. Kelas `OBDDB` merupakan kelas yang memberikan akses ke basis data, sementara `OBDModel` merupakan kelas untuk membaca respon dari ECU, sesuai dengan format OBD-II. Kelas `SupportedPIDFile`, seperti namanya digunakan untuk melakukan pembacaan apakah telah ada data PID dukungan ECU yang tersimpan sebelumnya, maupun melakukan penyimpanan / penghapusan data tersebut.

IV.2.3.1 Kelas `OBDDB`

Kelas `OBDDB` merupakan kelas statis, yang memberikan definisi akses ke basis data. Akses basis data pada sistem menggunakan pustaka Scala Query, yang memungkinkan kita menuliskan query dalam Scala, dengan hanya memberikan definisi basis data. Karenanya, tidak terdapat kode khusus pada kelas ini, selain definisi akses basis data sesuai dengan aturan Scala Query, serta definisi tabel yang juga mengikuti Scala Query. Listing IV-3 menujukkan contoh kode yang digunakan untuk mendefinisikan tabel “DTCDetails”.

```
object DTCDetails extends Table[(String, String, Clob)]("DTCDETAILS") {
  def id = column[String]("ID", O DBType ("CHAR(4)"), O NotNull)
  def dtcId = column[String]("DTCID", O DBType ("CHAR(1)"), O NotNull)
  def description = column[Clob]("DESCRIPTION")

  def * = id ~ dtcId ~ description
  def pk = primaryKey("pk_DTCDetails", id ~ dtcId)
  def fkDTC = foreignKey("fk_DTCDetails_DTC", dtcId, DTC)(_.id)
}
```

Listing IV-3 Definisi Tabel “DTCDetails” pada Sistem

IV.2.3.2 Kelas OBDModel

`OBDModel` merupakan kelas yang memiliki banyak fungsi, yang disimpan dalam hash map, untuk mempermudah pembacaan data yang dikirimkan oleh OBD-II. Tidak terdapat perubahan singnifikan antara perancangan dan implementasi pada

kelas ini. Perancangan serta alasan perancangan pada kelas ini dapat dibaca pada bagian III.2.3.2.2.

IV.2.3.3 Kelas SupportedPIDFile

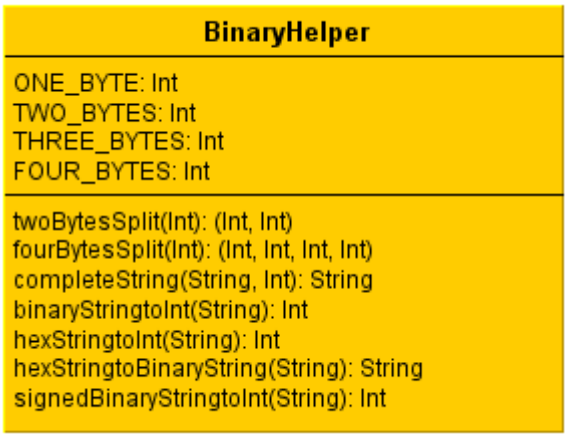
Sama seperti pada kelas `OBDModel`, tidak terdapat perubahan yang signifikan antara implementasi dengan perancangan pada kelas `SupportedPIDFile`. Rancangan serta alasan perancangan pada kelas ini dapat dibaca pada bagian III.2.3.2.3.

IV.2.4 Lapisan Utilitas Sistem

Lapisan utilitas bertanggung jawab dalam memberikan berbagai kelas untuk membantu perhitungan tertentu dalam sistem. Lapisan ini memiliki dua paket utama, yaitu `lib.helper` dan `lib.serial`. bagian ini akan menjelaskan kegunaan dan mekanika (jika perlu) dari kelas-kelas yang ada di dalam paket tersebut.

IV.2.4.1 Paket `lib.helper`

Paket `lib.helper` memiliki hanya satu buah kelas, yaitu `BinaryHelper`. Kelas ini berfungsi untuk membantu dalam melakukan kalkulasi bilangan biner, serta konversi bilangan biner ke basis lainnya (misalnya ke heksadesimal). Karena dirancang untuk digunakan sebagai utilitas, maka kelas ini bersifat statis, dan hanya terdiri dari beberapa properti dan fungsi statis.

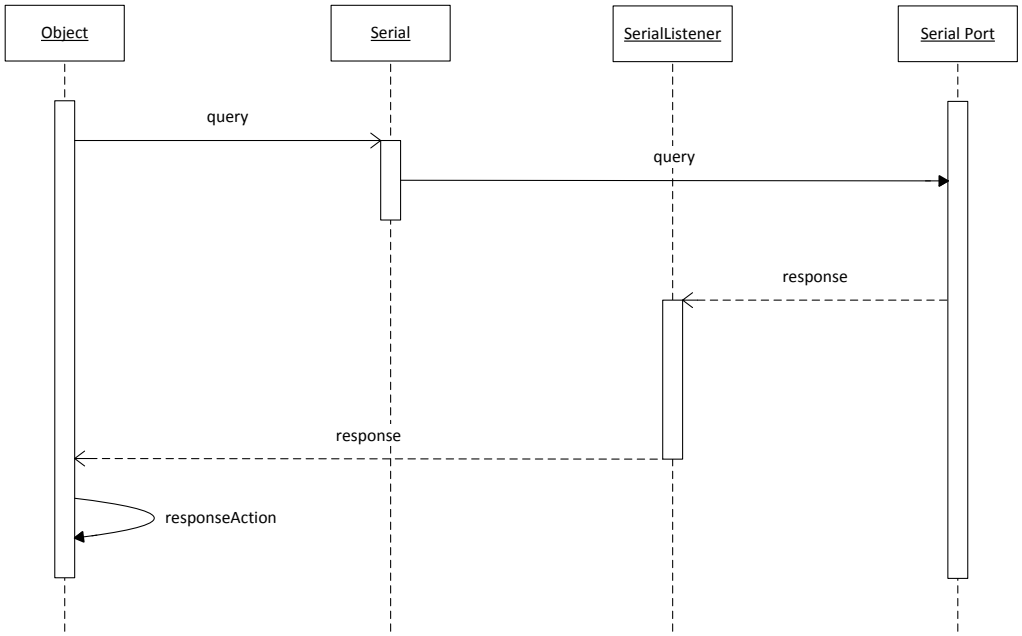


Gambar IV-7 Diagram Kelas dari `BinaryHelper`

Karena kesederhanaannya, maka tidak diperlukan banyak penjelasan mengenai kelas `BinaryHelper`. Gambar IV-7 menunjukkan diagram kelas dari `BinaryHelper`, dengan properti dan nama method yang cukup dapat menjelaskan kegunaannya.

IV.2.4.2 Paket `lib.serial`

Seperti namanya, paket ini bertanggung jawab dalam menghubungkan sistem dengan port serial. Paket ini terdiri dari sebuah kelas, yaitu `Serial` dan sebuah *trait*, yaitu `SerialListener`. Baik `Serial` maupun `SerialEvent` dirancang untuk berkomunikasi secara *event-based*, mengikuti sifat dasar dari port serial. Pembukaan dan penutupan koneksi, serta pengiriman data ke port serial diatur oleh kelas `Serial`, sementara `SerialListener` memiliki tugas membaca respon dari port serial ketika port serial telah mengirimkan respon. Gambar IV-8 menunjukkan interaksi antara `Serial` dan `SerialListener` dengan objek yang menggunakannya.



Gambar IV-8 Interaksi Objek, `Serial`, `SerialListener`, dan Serial Port

Perhatikan bahwa pada Gambar IV-8 objek "Serial Port" bukan merupakan objek dalam arti objek pada pemrograman, melainkan adalah perangkat keras (serial port yang sebenarnya). Karena `SerialListener` adalah sebuah *trait*, maka objek "Object" harus mengimplementasikan `SerialListener`, yang akan memberikan notifikasi kepada Object ketika Serial Port memberikan respon. Object kemudian akan melakukan pemrosesan, untuk kemudian melakukan penampilan terhadap respon. Pemrosesan ini dapat saja melibatkan kelas pada paket `lib.model`.

Pemrosesan respon diserahkan kepada objek yang menggunakan paket ini karena penampilan dari respon yang berbeda-beda. Misalnya, respon pada Power Mode akan ditampilkan dalam bentuk teks, sementara mayoritas respon pada Standard Mode akan diproses dan menjadi sumber data untuk pembaruan antarmuka. Karena `Serial` tidak dapat mengetahui pemanggilnya, maka tanggung jawab pemrosesan respon diberikan kepada objek yang mengimplementasikan `SerialListener`.

IV.3 Evaluasi Hasil

Bagian ini menjelaskan evaluasi terhadap hasil dari sistem yang dibangun, apakah telah dapat mencapai tujuan yang ditetapkan pada bagian I.3 atau tidak. Evaluasi dilakukan dari tiga sisi, yaitu komunikasi sistem dengan ECU, kelengkapan perintah OBD yang didukung, serta kemampuan sistem dalam memberikan informasi mesin yang akurat, dan *real-time*. Pada bagian akhir juga dilakukan evaluasi performa sistem, untuk melihat berapa banyak sumber daya sistem yang digunakan untuk menjalankan perangkat lunak.

Seluruh evaluasi diujikan pada tiga lingkungan pengujian yang berbeda, yaitu pada simulator OBD melalui perangkat lunak OBD Simulator, pada ECU Mercedes Benz E320 1998 V-Engine, dan pada ECU Chevrolet Aveo 2003. Pengujian dilakukan terhadap simulator dan mesin sebenarnya untuk mendapatkan perbandingan dari beberapa implementasi OBD-II, yang akan membantu dalam memastikan implementasi OBD-II pada perangkat lunak yang dikembangkan adalah benar.

IV.3.1 Komunikasi Sistem dengan ECU

Komunikasi sistem dengan ECU dilakukan melalui port serial, menggunakan chip ELM 327 untuk membantu dalam memproses protokol komunikasi sesuai standar OBD-II. Dalam hal ini, sistem dapat berkomunikasi dengan ECU tanpa masalah, selama mesin mendukung standar OBD-II.

Pengujian terhadap kemampuan komunikasi dilakukan dengan menjalankan sistem pada dua mobil berbeda, yaitu Mercedes Benz E320 1998 V-Engine dan Chevrolet Aveo 2003. Pada kedua kendaraan tersebut, sistem dapat berkomunikasi dengan mesin melalui OBD-II tanpa ada masalah berarti.

IV.3.2 Kelengkapan Perintah OBD-II yang Didukung

Terdapat total 181 perintah OBD-II yang dapat digunakan. Sistem yang dikembangkan mendukung 144 perintah dari seluruh perintah tersebut. Adapun perintah-perintah yang tidak didukung merupakan perintah yang ada pada mode 05 dan 09 (seluruh PID). Perintah pada mode 05 dan 09 tidak didukung oleh sistem karena adanya inkonsistensi antara dokumentasi yang didapatkan mengenai format nilai kembalian perintah OBD-II dengan nilai yang dikembalikan pada mesin sesungguhnya. Karena terdapat kesamaan pola antara nilai kembalian pada mesin Mercedes dan Chevrolet, maka diasumsikan kesalahan ada pada dokumentasi yang digunakan. Dokumentasi alternatif yang dapat digunakan sebagai pembanding tidak dapat ditemukan, sehingga diputuskan untuk tidak mendukung fitur-fitur tersebut.

Secara singkat: sistem mendukung **79.55%** perintah-perintah yang ada pada standar OBD-II.

IV.3.3 Akurasi Penampilan Informasi pada Sistem

Sistem berkomunikasi dengan OBD-II melalui port serial. Sesuai dengan natur dari port serial, komunikasi tidak dapat dilakukan secara sinkronus. Hal ini menyebabkan sistem tidak bisa mendapatkan informasi yang *real-time*, karena sistem harus mengirimkan perintah satu demi satu, dan setiap pengiriman perintah harus dibatasi dengan jeda waktu untuk menunggu port serial selesai mengirimkan jawaban. Selain itu, OBD-II juga tidak memiliki perintah atau mode untuk mengirimkan beberapa perintah sekaligus, sehingga menyebabkan setiap perintah harus dikirimkan satu demi satu, secara serial.

Pada kebanyakan halaman dalam Standard Mode (dan seluruh perintah pada Power Mode), jeda waktu ini bukan merupakan masalah. Masalah baru akan terjadi pada submenu Grafik pada menu Diagnostik. Karena grafik yang ditampilkan harus memetakan informasi pada sumbu X dan sumbu Y (2 dimensi), adanya jeda antara informasi yang didapatkan pada sumbu X dan sumbu Y akan

menyebabkan akurasi informasi tidak tepat. Adapun jeda informasi ini akan menyebabkan informasi dapat salah diinterpretasikan jika pengguna tidak hati-hati dalam membaca informasi. Gambar IV-9 memperlihatkan urutan langkah pengiriman perintah, antara perintah untuk data sumbu x dengan perintah untuk data sumbu y.

Keterangan Gambar:
1. Pengiriman Perintah Sumbu X
2. Respon Perintah Sumbu X
3. Jeda Serial Port
4. Pengiriman Perintah Sumbu Y
5. Respon Perintah Sumbu Y

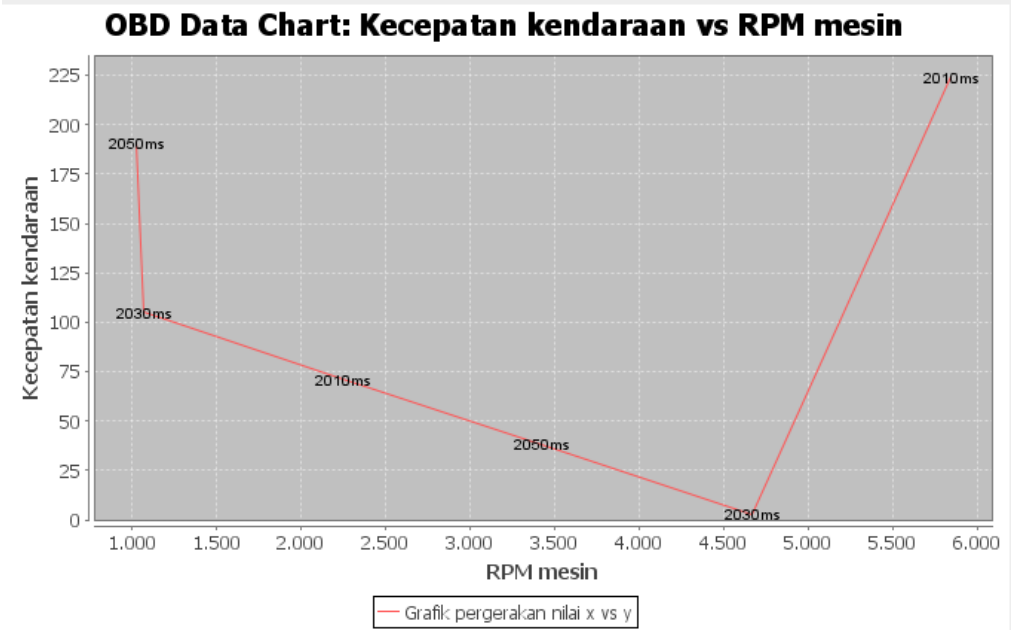
Gambar IV-9 Urutan Langkah Pengiriman Perintah Menu Grafik pada Sistem

Seperti yang dapat dilihat pada Gambar IV-9, terdapat jeda yang sangat panjang antara informasi yang diberikan pada sumbu x dan sumbu y. Tabel IV-2 memperlihatkan jeda waktu rata-rata pada setiap mesin pengujian. Jeda waktu ini menyebabkan pengguna sistem harus mencoba untuk menghasilkan nilai sumbu x yang konsisten selama jeda tersebut, untuk mendapatkan nilai sumbu y yang lebih tepat.

Tabel IV-2 Perbandingan Jeda Waktu Rata-Rata Port Serial

Perangkat	Rata-rata jeda
OBD Simulator	1550.304 ms
Mercedes Benz E320 1998 V-Engine	2100 ms
ECU Chevrolet Aveo 2003	2800 ms

Solusi yang ditawarkan oleh sistem untuk masalah ini ialah dengan memberikan informasi kepada pengguna jeda waktu antara informasi sumbu X dan sumbu Y. Dengan mengetahui jeda waktu tersebut, pengguna dapat memperkirakan rata-rata jeda, dan menjalankan mesin dengan stabil pada rata-rata waktu tersebut, untuk mendapatkan informasi yang lebih akurat. Gambar IV-10 memperlihatkan plot yang digunakan sistem untuk menampilkan grafik.



Gambar IV-10 Tampilan Jeda Waktu Pada Grafik

(Catatan: Angka menyatakan jeda waktu data sumbu Y relatif ke sumbu X)

IV.3.4 Evaluasi Performa Sistem

Evaluasi performa sistem dilakukan dengan menjalankan sistem dengan prosedur tertentu, dan mencatat sumber daya yang digunakan oleh sistem. Pengukuran performa (baik memori maupun CPU) dilakukan menggunakan perangkat lunak *profiler* Java VisualVM. Untuk menghemat ruang, laporan lengkap dari *profiler* tidak akan disertakan dalam tesis ini. Laporan lengkap *profiler* dapat dibaca pada CD yang menyertai tesis.

IV.3.4.1 Prosedur Pengujian

Prosedur yang digunakan untuk melakukan evaluasi performa yaitu:

1. Menghidupkan perangkat lunak.
2. Koneksi ke port serial yang digunakan oleh ELM327.
3. Masuk ke submenu Grafik pada Standard Mode.
4. Memilih parameter grafik sebagai berikut:

a. Sumbu X: PID 01 0C (RPM Mesin).
b. Sumbu Y: PID 01 0D (Kecepatan Kendaraan).

5. Membiarkan aplikasi berjalan selama 5 menit.
6. Melakukan *disconnect* dan mematikan aplikasi.

Submenu Grafik dipilih sebagai tempat pengujian karena submenu ini merupakan modul yang memerlukan paling banyak sumber daya (dengan melakukan pengiriman perintah, pembacaan respon, dan penggambaran grafik secara terus menerus).

IV.3.4.2 Lingkungan Pengujian

Lingkungan pengujian yang digunakan untuk melakukan evaluasi performa perangkat lunak yang dikembangkan dapat dilihat pada Tabel IV-3.

Tabel IV-3 Lingkungan Pengujian Sistem

Perangkat Keras		
Prosesor	:	Intel Core i5-2410M (2.3 GHz)
Memori	:	8 GB 1333 MHz DDR3
Resolusi	:	1366x768
Hard Drive	:	500GB SATA (5400 RPM)
Perangkat Lunak		
Sistem Operasi	:	Windows 7 Proffesional 64-bit
Versi Java	:	java version "1.7.0_09" Java(TM) SE Runtime Environment (build 1.7.0_09-b05) Java HotSpot(TM) 64-Bit Server VM (build 23.5-b02, mixed mode)
Versi Scala	:	Scala code runner version 2.9.2

Catatan untuk Tabel IV-3:

- Resolusi layar disertakan karena penggambaran antarmuka (terutama grafik pada submodul Grafik) dilakukan menggunakan CPU, yang menjadikan resolusi sangat penting dan relevan untuk pengukuran performa.
- Kecepatan *hard drive* disertakan karena penggunaan basis data H2 dan berkas penyimpanan PID oleh sistem. Performa untuk membaca kedua data tersebut

(yang akan selalu dipanggil pada awal perangkat lunak dijalankan) sangat bergantung kepada kecepatan *hard drive*.

- Versi java dan versi scala didapatkan dengan menjalankan perintah `java --version` dan `scala --version` berurutan.

IV.3.4.3 Hasil Pengujian Performa Sistem

Hasil pengujian yang dilakukan mengindikasikan bahwa performa dimiliki oleh sistem telah optimal. Mayoritas sumber daya memori dan CPU digunakan oleh Sistem sangat minimal, dengan nilai penggunaan maksimal sekitar 25 MB (dari total 8GB memori sistem, atau sekitar **0.003%** dari total memori lingkungan pengujian). Performa sistem secara keseluruhan dan ringkas dapat dilihat pada Gambar IV-11 pada halaman 74.

Adapun rincian penggunaan memori selama menjalankan aplikasi sesuai dengan prosedur yang dipaparkan pada bagian IV.3.4.1 dapat dilihat pada Tabel IV-4.

Tabel IV-4 Performa Memori pada Sistem

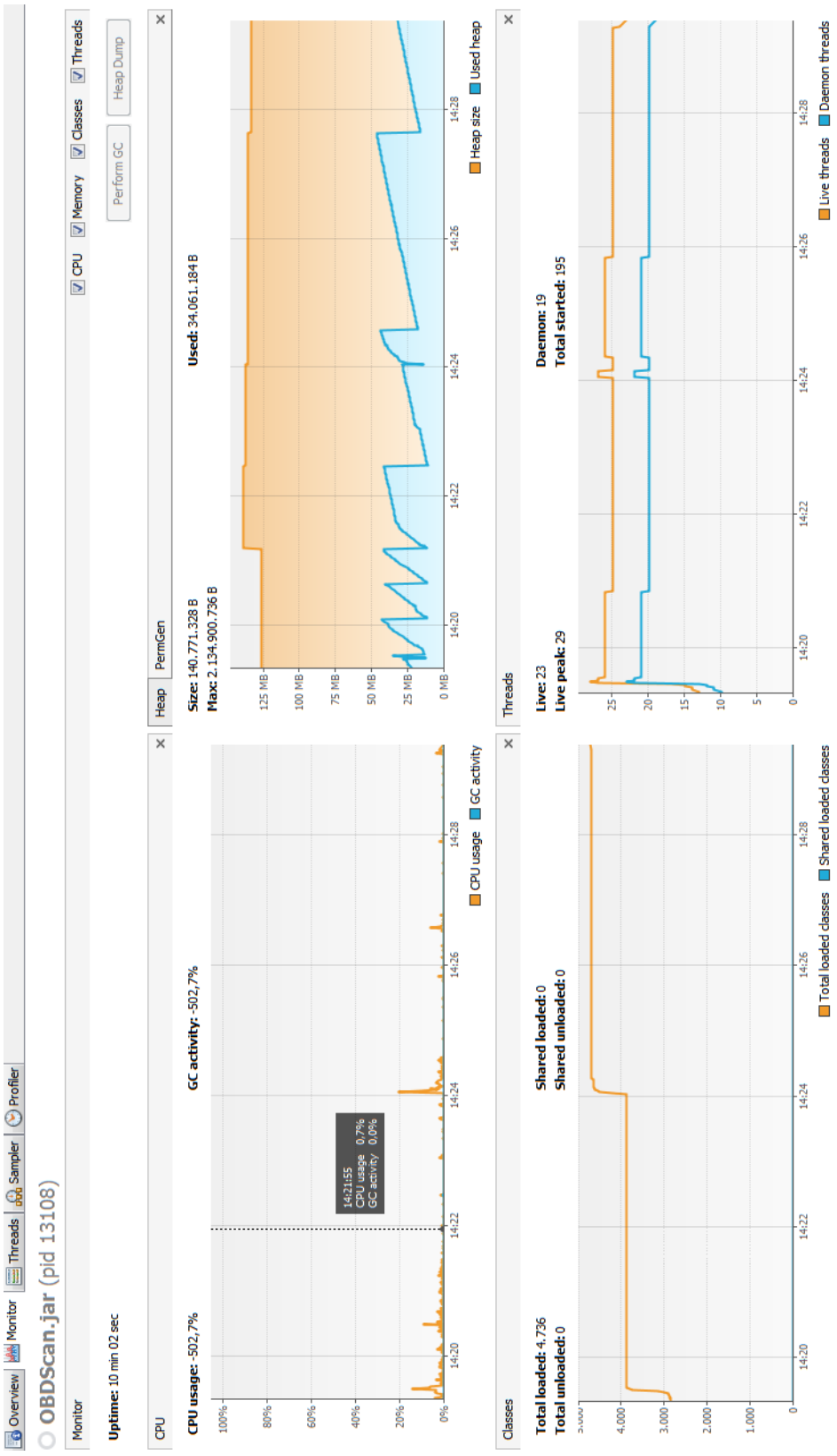
Metrik		Nilai
Rata-rata Memori Hidup (Live Bytes)	:	5.687.448 B (5.4 MB)
Total Objek Aktif	:	82.080 objek
Total Alokasi Objek	:	363.080 objek
Rata-rata Usia Objek	:	1.2 detik
Total Pembuatan Objek	:	2137
Total Objek dalam Sistem	:	5605

Sistem menggunakan rata-rata 5.4 MB memori, dengan maksimal 25 MB (Gambar IV-11) tepat sebelum subsistem GC (*Garbage Collection*) pada JVM dijalankan. Secara total, terdapat 5605 objek, yang tidak semuanya diinisialisasi. Objek yang tidak diinisialisasi dapat berupa objek *native* JVM (misal: int) ataupun objek anonim, yang banyak digunakan oleh Scala. Pengguna memori terbanyak ialah objek `org.h2.util.CacheObject`, yaitu sebesar 1 MB.

Dari sisi CPU sendiri, sistem juga menggunakan sumber daya yang sangat minimal. Penggunaan CPU pada sistem berada pada kisaran 1% dari total sumber

daya yang disediakan, dengan puncak penggunaan berada pada kisaran 20%. Mayoritas sumber daya CPU digunakan untuk melakukan sinkronasi *thread*, karena pengiriman perintah ke PID yang dirancang untuk dijalankan secara paralel oleh sistem.

Sayangnya, VisualVM tidak dapat memberikan informasi yang relevan pada bagian ini karena keterbatasan arsitektur Java (Stack Overflow, 2011). Hal ini menyebabkan tidak terdapat metrik yang lebih mendetail pada data *profiler* yang dihasilkan selain penggunaan CPU secara keseluruhan, dan penggunaan waktu CPU relatif terhadap komponen sistem keseluruhan (yang tidak menunjukkan penggunaan sumber daya sistem). Begitupun, utilisasi CPU rata-rata 1% cukup untuk menunjukkan bahwa sistem menggunakan CPU secara efisien.



Gambar IV-11 Ringkasan Performa Sistem Secara Keseluruhan

Bab V Kesimpulan dan Saran

Bab ini menjelaskan kesimpulan yang didapatkan selama pelaksanaan penelitian dan penulisan tesis ini. Bagian saran memberikan penjelasan mengenai perbaikan atau peluang penelitian lanjutan yang tidak dilakukan pada tesis ini.

V.1 Kesimpulan

Adapun kesimpulan yang didapatkan selama pelaksanaan penelitian dan penulisan tesis ini adalah sebagai berikut:

1. ECU memberikan fungsionalitas yang sangat menguntungkan manufaktur dan pengendara mobil, sambil tetap menjaga kesederhanaan arsitekturnya. Komunikasi dengan ECU juga dapat dengan mudah dilakukan, karena adanya standar OBD-II.
2. Perangkat lunak diagnosa mesin mobil berdasarkan standar OBD-II dapat dikembangkan dengan sangat mudah, karena keterbukaan spesifikasi OBD-II. Hal ini tentunya sangat membantu bengkel dalam melakukan reparasi tanpa memerlukan bantuan dari manufaktur.
3. Pembacaan fitur-fitur non-standar OBD-II pada ECU sulit dilakukan karena proteksi dari manufaktur terhadap ECU yang sangat ketat.
4. Pengembangan perangkat lunak model *waterfall* cocok digunakan untuk membangun perangkat lunak dengan spesifikasi yang telah jelas dan tidak berubah, meskipun terdapat kompleksitas yang tinggi (seperti pada kalkulasi respon PID OBD-II) pada perangkat lunak tersebut.
5. Port serial merupakan media komunikasi yang tidak optimal untuk digunakan dalam mengambil data *real-time*, tetapi port serial memiliki kompatibilitas tinggi. Jika mengembangkan perangkat yang memerlukan informasi *real-time*, sebaiknya tidak menggunakan port serial, melainkan media lain (misalnya: port paralel) yang memungkinkan untuk mengirimkan beberapa perintah sekaligus.

V.2 Saran

Beberapa hal yang sebaiknya dilakukan untuk penelitian selanjutnya yaitu:

1. Percobaan pemanfaatan teknik rekayasa balik protokol komunikasi jaringan (Cui, Kannan, & Wang, 2007) dan kode mesin untuk mendapatkan perintah-perintah non-standar pada mesin, ataupun perintah pada OBD-II yang belum terdokumentasi secara lengkap.
2. Pengembangan protokol komunikasi OBD-II untuk mendukung pengiriman beberapa perintah sekaligus. Pengembangan protokol komunikasi ini akan memerlukan pengembangan lanjutan pada sisi ECU juga.
3. Pengembangan sistem pakar yang memantau data operasional mesin dan memberikan berbagai rekomendasi, misalnya rekomendasi kinerja mesin relatif terhadap performa operasional mesin. Sistem pakar dapat menggunakan data dari OBD-II untuk memantau kinerja dan memberikan rekomendasi.

Daftar Pustaka

Autologic Software. (2012). *Autologic Software Technical Specification for Mercedes Benz Vehicle.* Autologic Software.

BreakingPoint Labs. (2009, 1 13). *Automated Protocol Reverse Engineering | BreakingPoint Labs*. (BreakingPoint Labs) Dipetik 8 15, 2012, dari http://www.breakingpointsystems.com/resources/blog/automated-protocol-reverse-engineering/

Comparetti, P. M., Wondracek, G., Kruegel, C., & Kirda, E. (2009). Prospex: Protocol specification extraction. *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, 110-125.

Cui, W., Kannan, J., & Wang, H. J. (2007). Discoverer : Automatic Protocol Reverse Engineering from Network Traces. *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium* (hal. 1-14). Berkeley, CA: USENIX Association.

Cui, W., Kannan, J., & Wang, H. J. (t.thn.). Discoverer: Automatic protocol reverse engineering from network traces. *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, 1-14.

Cui, W., Peinado, M., Chen, K., Wang, H. J., & Irún-Briz, L. (2008). Tupni: Automatic reverse engineering of input formats. *Proceedings of the 15th ACM Conference on Computer and Communications Security*, *10*(2008), 391–402.

Eilam, E. (2005). *Reversing: Secrets of Reverse Engineering.* Indianapolis: Wiley Publishing, Inc.

Elec_Intro. (2010, April 12). *obd fault code_Automation-Drive*. Dipetik August 2012, dari Automation-Drive: http://www.automation-drive.com/obd-fault-codes

European Union. (2010). *Commision Regulation (EU) No. 461/2010*. Dipetik July 27, 2012, dari http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2010:129:0052:0057:EN:PDF

GT-Shop. (2011, November). *Kleemann S2 Kompressor комплект with ECU Mercedes CLK500 & CLK55 V8 5spd W209 02+*. Dipetik August 2012, dari GT-Shop: http://gt-shop.ru/catalog/130392/201382.html

Lyberty.com. (2005, January 9). *OBD-II Drive Cycle (Reset Car Diagnostic Monitor).* Dipetik September 27, 2012, dari http://www.lyberty.com/car/drive-cycle.html

Movin on GPS. (2012, January 26). *OBD-II GPS Tracking*. Dipetik August 2012, dari Movin on GPS: http://www.movinongps.com/obd2-gps-tracking/

O'Connor, D. (2009). *An Embedded Automotive Monitoring Device Automon.* Cork: Cork Institute of Technology.

Ordersky, M., Spoon, L., & Venners, B. (2010). *Programming in Scala Second Edition.* Walnut Creek: Artima Press.

Ritcher, M. (2006). Understanding the ECU, What it does and How it works. MC2 Magazine.

SAE International. (2002). *SAE J1979 Revised APR2002.* SAE.

Sommerville, I. (2007). *Software Engineering (8th Edition).* Addison-Wesley.

Stack Overflow. (2011, 11 15). *profiler - Why VisualVM Sampler does not provide full information about CPU load (method time execution)?* Dipetik 01 09, 2013, dari Stack Overflow: http://stackoverflow.com/questions/8130277/why-visualvm-sampler-does-not-provide-full-information-about-cpu-load-method-ti

Sutherland, J. (2010). *Scrum Handbook.* Boston: Scrum Training Institute Press.

Toyota Motor Sales. (2005, January 17). *Engine Controls - Input Sensors.* Dipetik August 22, 2012, dari Autoshop 101: http://www.autoshop101.com/forms/h24.pdf

Volkswagen Group of America. (2000). *K-Line Communication Description.*