Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Semicondutores, Instrumentação e Fotônica

Simulador de Ambiente Automotivo para Injeções Eletrônicas

Gustavo Covizzi Menna Barreto Alonso

Dissertação submetida à faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica, sob orientação do Prof. Dr. Carlos Alberto dos Reis Filho

Banca Examinadora:

Dr. Carlos Alberto dos Reis Filho – FEEC/Unicamp

Dr. Furio Damiani – DSIF/FEEC/Unicamp
Dr. Antônio Carlos Fiore de Mattos – Cenpra

Campinas, 12 de Novembro de 2004

AL72s

Alonso, Gustavo Covizzi Menna Barreto
Simulador de ambiente automotivo para injeções eletrônicas / Gustavo Covizzi Menna Barreto Alonso.-- Campinas, SP: [s.n.], 2004.

Orientador: Carlos Alberto dos Reis Filho
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Aquisição de dados. 2. Simulação (Computadores). 3. Programação visual (Computação). 4. Engenharia automotiva. 5. Automóveis – Motores – Sistemas de Automação - Testes. 6. UML (Linguagem de modelagem padrão). 7. Automóveis – Equipamento elétrico – Testes. I. Reis Filho, Carlos Alberto dos. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Resumo

Este trabalho irá descrever o desenvolvimento de um sistema baseado em computador pessoal, que simula um ambiente automotivo para unidades de controle de sistemas de injeção eletrônica (*centralinas*) e adquire as informações relativas à sua atuação sobre o veículo. O usuário controla os diversos parâmetros do ambiente simulado e verifica a resposta da *centralina* através de uma interface gráfica interativa e acessível via *mouse*.

Este sistema visa propiciar uma assistência técnica eficiente, diagnosticando problemas em campo com rapidez e confiabilidade, substituindo os simuladores manuais atuais que são adaptados para um único modelo de *centralina* e dependem de outros equipamentos como um osciloscópio para visualização dos sinais de interesse.

Além de substituir todo um conjunto desses simuladores por um único sistema, nossa abordagem também oferece a possibilidade de efetuar testes dinâmicos e fornece uma ferramenta que atenda a diversos usuários com diferentes necessidades. Outras características a serem apresentadas pelo nosso sistema são portabilidade, adequação ao ambiente industrial e flexibilidade, para permitir uma fácil atualização a novos equipamentos de aquisição de dados (prevenção contra a obsolescência).

Summary

This work describes the development of a PC-based system, who simulates an automotive environment for Electronic Fuel Injection Control Units (EFIs) and acquires information about their acting over the vehicle. The user is able to control several parameters of the simulated environment and verify the EFI performance through an interactive graphical interface, accessible by mouse.

This system intends to provide an eficient technical support, promptly and reliably diagnosing field problems, replacing the existing manual simulators. Indeed, these manual simulators are generally adapted to a single EFI model and hang upon other equipments such as an osciloscope for singals visualization.

The solution presented in this work intends to replace a hole set of these manual simulators for a single system, with the capability of performing dynamic tests, being a tool to supply different users with different requirements. Other outstanding characteristics of the developed system are portability, industrial environment compliance and flexibility, to allow easy upgrade to new data acquisition equipments.

"What kind of man would live where there is no daring?
I don't believe in taking foolish chances,
but nothing can be accomplished without taking any chance at all."
– Charles Augustus Lindbergh (1902 – 1974)

"Tudo o que uma pessoa possa imaginar, outras podem tornar real."
– Júlio Verne (1828 – 1905)

"A maneira de fazer é ser."
– Lao Tsé (604 – 531 a.C.)

A todos os meus familiares e amigos

Agradecimentos

Ao meu orientador pela perseverança de não deixar esse trabalho inacabado.

Aos muitos amigos de faculdade e do Laboratório de Pesquisas Magneti Marelli que, por sua participação nesse trabalho ou simplesmente por estarem por perto nessa importante etapa da minha vida, são co-responsáveis pela minha formação intelectual e moral, especialmente mas não somente a Érico de Lima Azevedo, Danilo Barreto de Araújo, Cleber Akira Nakandakare, Jorge Arturo Polar Seminário, Danielle Melo, Marcos Vinicius Gil Gomes, Marcos Mauricio Pelícia, Marcelo Arturo Jara Perez, Leandro Ferrari Crocomo e Luiz Alberto Castro de Almeida.

Aos meus pais, irmãos e avós por me darem as condições materiais e psíquicas para o meu desenvolvimento pessoal e social.

A minha esposa Amaya Gonzalez Gascue por completar-me e encher minha vida de alegria.

Índice Geral

Índice de Figuras

Índice de Tabelas

Capítulo 1 – Introdução

1.1 Um panorama da eletrônica automotiva

No início da década de 80, as unidades de controle utilizadas em veículos eram gerenciadas por microprocessadores de 8 bits e controlavam apenas as funções mais básicas do motor como razão ar-combustível e temporização da ignição. No início da década de 90, algumas dessas unidades de controle já utilizavam microprocessadores de 16 bits e seu uso se propagou para outros sistemas veiculares como transmissão, anti-travamento de freios e suspensão. No decorrer da década de 90, unidades de controle mais poderosas e complexas utilizando microprocessadores de 32 bits, vêm sendo utilizadas para controlar sistemas completos de *powertrain* e chassis. Essas unidades de controle executam algoritmos mais sofisticados e, em alguns casos, integram funções anteriormente distribuídas em diversas outras unidade de controle independentes[1].

As modificações nas unidades de controle não se referem apenas a capacidade de processamento ou número de tarefas a serem executadas, mas passam por toda uma revisão dos processos utilizados no decorrer do seu projeto. Tradicionalmente, a maior parte dos desenvolvimentos era feita pelos engenheiros utilizando-se de métodos de tentativa e erro. Atualmente, devido não apenas à complexidade crescente da eletrônica, mas também principalmente das características do mercado atual com requisitos de segurança cada vez mais severos e níveis de emissões cada vez mais estritos, pressões constantes para redução de custos e competitividade, e nivelamento dos recursos tecnológicos das empresas com a massificação do conhecimento especializado; a avaliação do veículo completo deve ser executada sob as condições mais extremas. As instalações de simulação oferecem essa facilidade com um amplo leque de opções, incorrendo em um custo fixo inferior e reduzindo o tempo de desenvolvimento imensamente quando comparadas com condições reais de testes. Além de verificar se um desenvolvimento será bem sucedido antes de haver muitos custos envolvidos, métodos de simulação *off-line* realistas são necessários pelas seguintes razões:[2]

- ✓ Desenvolvimento das estratégias de controle;
- ✓ Análise de modos de falha;
- ✓ Performance do veículo sob diferentes condições de direção;
- ✓ Condições de teste adequadas para diversas superfícies de rodagem (gelo, neve compactada, p.e.) não disponíveis no momento, pois seriam necessárias uma pista de testes dedicada e clima apropriado;
- ✓ Condições reproduzíveis (Não há nada como gelo polido p.e., o coeficiente de atrito varia drasticamente com a temperatura);
- ✓ Situações de teste rápidas e seguras são potencialmente aleatórias com estratégias de controle em início de desenvolvimento, mesmo nas melhores instalações disponíveis.

1.2 Motivação do Projeto

Antes de iniciarmos as considerações sobre os fatores que geraram nosso interesse no desenvolvimento deste ambicioso projeto, é necessário garantirmos uma compreensão exata de alguns termos a serem utilizados. O ambiente hardware-in-the-loop (HitL) será melhor descrito no decorrer desta introdução. Portanto vamos às seguintes definições[13]:

- Verificação: o processo de avaliação de um sistema para determinar se os produtos de uma determinada fase de desenvolvimento satisfazem os requisitos impostos no início daquela fase.
- Validação: o processo de avaliação de um sistema, no final do processo de desenvolvimento do software, para determinar se ele satisfaz os requisitos funcionais sistêmicos em sua totalidade.
- Ambiente Estático de Teste: Um ambiente de teste, eletronicamente equivalente ao de um veículo, que permite a um usuário especificar um único e irreproduzível conjunto de dados de entrada para a unidade de controle automotiva.
- Ambiente Dinâmico de Teste: Um ambiente de teste hardware-in-the-loop (HitL), similar a um veículo real, que permite a um usuário especificar um conjunto de dados de entrada para a unidade de controle e/ou comandos do motorista, reproduzível e programável.

Como conseqüência da evolução das unidades de controle eletrônicas para aplicações automotivas, que passaram a trabalhar com microprocessadores mais poderosos, executando mais tarefas e com algoritmos mais complicados; as rotinas de software que controlam seu funcionamento cresceram para além da capacidade de um ou dois engenheiros poderem especificar, desenvolver, testar e manter todo o sistema. Para garantir um alto nível de qualidade no desenvolvimento dessas unidades de controle, estratégias de verificação de software avançadas devem ser implementadas.

Os métodos de produção de software tradicionais relegam as atividades de verificação para o final do ciclo de desenvolvimento do produto, o que resulta em um "teste dos erros do programa". A abordagem atual reza que essas atividades devem ser realizadas a cada estágio do seu ciclo de desenvolvimento, desde testes para cada módulo de software até os testes finais de integração, permitindo a identificação de problemas com antecedência e até minimizando sua ocorrência. Geralmente os níveis de testes se dividem em três categorias: unidade, sistema e veículo[1].

Os testes ao nível de unidade verificam requisitos de software que necessitam de ferramentas de teste e *debugging* intrusivas para demonstrar adequadamente a conformidade do software com as especificações de projeto. A maioria dos testes realizados pelo grupo de Engenharia de Software pertence a esta categoria.

Os testes sistêmicos requerem um ambiente de teste integrado à unidade de controle para verificar os requisitos do sistema. A maioria dos testes desta categoria é realizada pelo grupo de Engenharia de Sistemas, pois já envolve conhecimento da física associada ao ambiente de atuação da unidade de controle.

Já os testes veiculares requerem o uso de um veículo para demonstrar a conformidade com as especificações e fazer a validação final do projeto, após avaliação do funcionamento da unidade de controle quando em conjunto com os diversos outros sistemas eletrônicos presentes no veículo. Esses testes são exigidos e realizados em sua maioria pelas montadoras que são clientes desse produto.

Além destas considerações quanto às tarefas a serem executadas que devem ser feitas durante as etapas iniciais de projeto, deve ser levado em conta que a qualidade dos muitos produtos de eletrônica automotiva embarcada depende fundamentalmente da sua confiabilidade durante TODA a sua vida útil. E em um ambiente extremamente agressivo e ruidoso, o correto funcionamento dessa eletrônica depende de diversas circuitarias auxiliares como: proteções contra transientes de tensão/corrente e picos de tensão reversa; estabilizadores da tensão de referência (bateria); filtros para o ruído introduzido em sinais de baixa amplitude. Além disso, esses produtos ainda devem se adequar a diversas normas específicas de Compatibilidade e Imunidade Eletromagnética (EMC & EMI) emitidas pelas organizações ISO e DIN[4]. No entanto, mesmo em um projeto que suporte tudo isso eficientemente, ainda persistem as disfunções provocadas por componentes danificados ou imprecisos, erros na linha de montagem, descalibração, desgaste natural e até mesmo fadiga mecânica devido a vibrações intensas e constantes. Apesar de algumas destas causas poderem ser minimizadas com a adoção de programas de qualidade que garantam uma taxa de produtos defeituosos tão baixa quanto se queira, alguns problemas ocorrem inevitavelmente em vários produtos em circulação, e para os clientes que os adquirirem ela será de 100%.

Uma forma de minimizar essa insatisfação em potencial é prover aos clientes uma assistência técnica eficiente, capaz de resolver a maioria destes problemas com rapidez, onde quer que ele se encontre. Isso pressupõe a existência de um sistema capaz de efetuar diversos testes em campo e utilizar as funções de diagnóstico próprias do produto a fim de identificar o problema e solucioná-lo, quando possível, ou indicar que o produto deve ser trocado.

Os equipamentos utilizados atualmente para satisfazer a ambos os requisitos: auxiliar no projeto de novas unidades de controle e funcionar como ferramenta de diagnóstico para assistência técnica; podem ser classificados basicamente em simuladores estáticos ou dinâmicos (HitL). Deve ser enfatizado que nenhum destes equipamentos pretende e nem tem a capacidade de substituir completamente os testes veiculares.

Os simuladores estáticos atuais contêm circuitos eletrônicos que fornecem cargas elétricas equivalentes àquelas encontradas em um veículo real. Esses simuladores são montados em caixas nada portáteis, cujo Painel Frontal contém potenciômetros, chaves e outros controles para produzir os sinais de excitação – comandos do motorista (chave de partida, posição do pedal do acelerador/borboleta, etc.) e sinais dos sensores (rotação do motor, temperatura do líquido de arrefecimento, etc.) – e LEDs para monitorar as saídas referentes pela *centralina* em questão, pois eles são projetados para atender a um único modelo em particular. O operador deve variar cada sinal manualmente, tentando executar um padrão de teste, porém sem o compromisso de reproduzir uma situação real ou possível.

Dessa forma a repetibilidade dos testes, que podem requerer padrões de dados complexos, depende da habilidade do operador em ajustar estes dispositivos.

Adicionalmente, esses simuladores não têm capacidade de gerar a resposta do veículo resultante dos comandos da *centralina* sobre seus atuadores. Eles devem ser utilizados junto com osciloscópios, multímetros e outros instrumentos para visualização dos sinais de interesse e também podem ser utilizados junto com um PC para ativar funções internas de diagnóstico e obter informações sobre falhas detectadas pela *centralina* através de uma linha serial. Os simuladores estáticos também não permitem a geração de documentação de teste devido à sua incapacidade de aquisição de dados[1][5].

Um simulador HitL (ou mais simplesmente HiL) emula os sensores e atuadores de um sistema de controle para recriar um ambiente de operação, similar a um veículo real, porém de uma forma controlada. Idealmente, a unidade de controle deve ser incapaz de distinguir o ambiente de simulação de um ambiente real. Os elementos básicos de um simulador HiL podem ser categorizados em máquina de processamento, recursos de entrada/saída (E/S), condicionadores de sinal e interface com o usuário, conforme podemos ver na Figura 1. As máquinas de processamento realizam as tarefas de planejamento dos sinais no tempo de acordo com *scripts* de teste, execução de modelos dinâmicos e aquisição de dados interna. Os cartões E/S são tipicamente conversores analógico-digitais e digital-analógicos, oferecendo resolução de 12 ou 16 bits, bem como linhas digitais TTL. Condicionadores de sinal realizam as operações necessárias para enviar/receber sinais da/para a unidade de controle. Finalmente a interface do usuário é o meio para comunicar informações de/para o usuário ou computadores externos. Atividades podem incluir a criação de scripts de teste, definição de critérios *pass*/*fail*, execução de testes e apresentação de resultados graficamente. Esses quatro elementos unidos fornecem um ambiente para repetir seqüências de teste de forma precisa para efetivamente investigar e documentar a operação de uma unidade de controle[5].

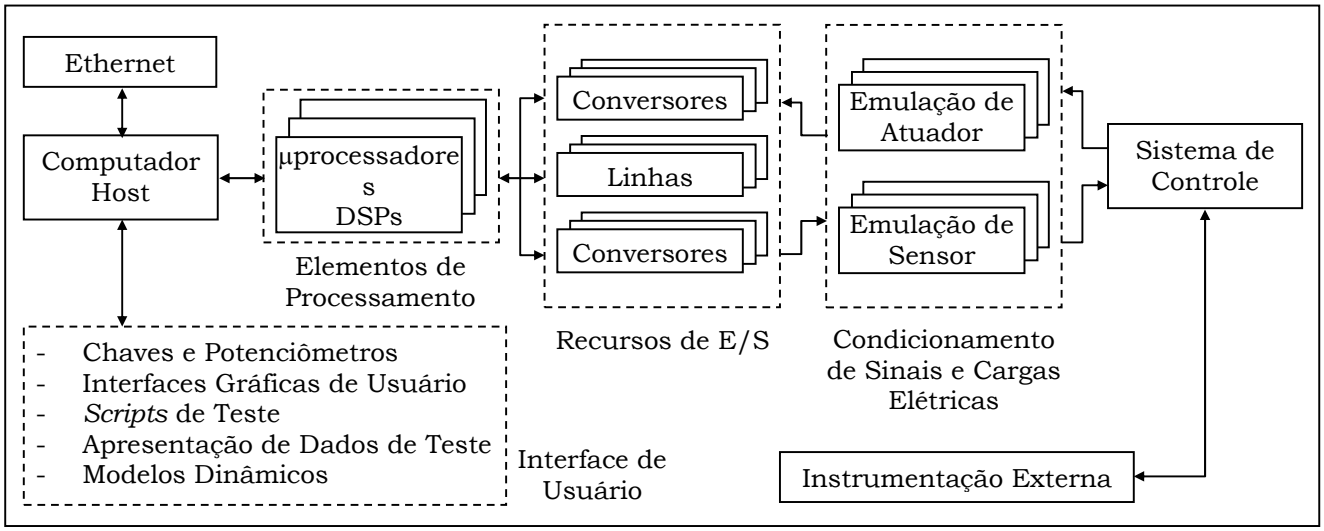


Figura 1: Elementos-chave de um Simulador Automotivo HiL

Equipamentos que satisfazem essa definição já são utilizados em algumas empresas, porém estamos interessados em sua evolução futura. Uma visão para a próxima geração de simuladores eletrônicos HiL deve atender a seguinte diretriz: "A realização de um simulador *real-time* portátil e econômico capaz de executar modelos dinâmicos (simples) com recursos modulares de entrada/saída, núcleo digital atualizável, *scripts* para testes automatizados reproduzíveis, *data logging* para análise e documentação, interfaces gráficas de usuário (GUI) e recursos amigáveis para o usuário, facilitando o teste sistêmico de unidades de controle automotivas". Citamos a seguir algumas características da filosofia de projeto a ser adotada para a viabilização destes equipamentos:[5]

- ✓ Os principais fatores para possibilitar a migração da tecnologia HiL de laboratórios especializados para bancadas de engenharia comuns são o custo e a facilidade de utilização. A capacidade de o cliente desenvolver habilidades próprias para personalizar e realizar manutenção no equipamento direciona a realização desses simuladores para a utilização de soluções disponíveis comercialmente sempre que possível. Os recursos de E/S devem ser modulares, possibilitando fácil conexão de tipo encaixe e reconfiguração do software para se acomodar ao novo mapeamento dos sinais. Todos os sinais devem ser condicionados e protegidos para evitar danos às placas caso haja conexões incorretas. Um ciclo de vida prolongado para o sistema depende da flexibilidade para atualização das máquinas de processamento à medida que a tecnologia progride. Para atender às necessidades futuras que venham a surgir durante sua utilização, o projeto deve possuir arquitetura aberta permitindo uma fácil integração de componentes de hardware e software de outros fabricantes.

- ✓ Um modelamento dinâmico do comportamento do veículo e atividades de verificação da unidade de controle utilizando-se de scripts de teste requerem o fornecimento e aquisição de sinais para/do controlador em tempo real. Para atingir características de tempo real, o tempo de execução de um laço de simulação deverá ser inferior a 0,1Φ, sendo Φ a constante de tempo mais rápida na dinâmica dos subsistemas veiculares ou o tempo de execução do laço de controle da unidade eletrônica. Um laço de simulação se inicia com a máquina de processamento realizando a leitura dos dados de E/S da unidade de controle através da interface de hardware existente (recursos de E/S e condicionamento de sinais). Em seguida estes dados são processados para se avaliar a dinâmica do sistema, os valores fornecidos através da GUI são checados e um novo conjunto de dados para atualização do ambiente simulado é disponibilizado para escrita. Durante essa etapa, os dados adquiridos e os que serão simulados, bem como variáveis intermediárias de simulação, devem estar disponíveis para armazenamento e a GUI deve ser atualizada. Finalmente os sinais especificados pela máquina de processamento são enviados à unidade de controle através da interface de hardware. O sistema operacional deve ser capaz de manter a temporização do laço de simulação e sinalizar passos omitidos.
- ✓ A GUI deve ser amigável e permitir reconfiguração do painel frontal para atender a diferentes usuários. Haverá cinco modos de operação da GUI: manipulação direta dos sinais do simulador através de botões, chaves e controles de rolagem e observação dos resultados; gravação desses sinais de entrada para posterior reprodução; criação de um *script* de teste na GUI com um *clock* artificial para simulação em tempo real; execução de *scripts* de teste preexistentes em tempo real com *data logging* sincronizado-temporizado e visualização de dados adquiridos em simulações anteriores.

Nosso objetivo aqui será desenvolver um sistema capaz de efetuar testes em diversos produtos de eletrônica automotiva, sendo que focaremos nosso protótipo em unidades de controle de injeção e ignição eletrônicas, que doravante denominaremos de *centralinas*. O sistema será um simulador de todo o ambiente do veículo perante as *centralinas*, verificando seu comportamento em uma situação de funcionamento pseudo-real, conforme os conceitos de simuladores HiL discutidos acima. Toda a comunicação entre o simulador e a *centralina* deverá ser feita através do seu conector.

Os equipamentos que foram tomados como parâmetro de desenvolvimento, são os simuladores estáticos utilizados pela Magneti Marelli do Brasil primariamente para atividades de assistência técnica junto às montadoras de veículos suas clientes. Outros equipamentos que apresentam características de simulação dinâmica ainda estavam em desenvolvimento na matriz desta empresa e foram projetados para atender primordialmente suas necessidades locais, sem levar em conta as particularidades do mercado brasileiro.

Nossa abordagem visa não somente substituir todo um conjunto dos simuladores estáticos atuais por um único sistema portátil, como também oferecer as possibilidades de efetuar testes dinâmicos e registrar os resultados obtidos sem o auxílio de outros equipamentos, fornecendo ferramentas configuráveis por software que atendam às necessidades de diferentes usuários. Adicionalmente o sistema deve ser modular, para ser utilizado em diversas *centralinas* reaproveitando configurações anteriores, e flexível, para permitir uma fácil atualização a novos equipamentos de aquisição de dados. Eventualmente não iremos atender a todas as características dos simuladores HiL citados acima, para nos atermos às necessidades reais do mercado brasileiro: tanto atuais quanto esperadas.

Como a descrição de um sistema desse porte ficaria incompleta se nos limitássemos às possibilidades dessa dissertação em papel, acrescentamos um CD no encarte que contém diversas informações complementares bem como todo o código-fonte do software gerado no escopo deste trabalho.

Figura 2: Típicos equipamentos de teste "estáticos"

1.3 Metodologia

Para atender as características de um simulador HiL mencionadas no Item 1.2 acima com a melhor relação custo-benefício, optamos desde o início pela utilização de conceitos de co-projeto hardware-software. O interesse em metodologias que se utilizem destes conceitos tem aumentado à medida que os sistemas digitais crescem em complexidade e aumenta a disponibilidade de tecnologias de implementação, trazendo a necessidade de um balanceamento dos prós e contras nas decisões de se utilizar elementos hardware ou software para cada funcionalidade. Essas novas metodologias de projeto requerem que software e hardware sejam co-projetados buscando otimização com respeito a desempenho e custo. Podemos citar três aspectos principais que tem estimulado o interesse nesta área:

- ✓ A necessidade de sistemas computacionais mais eficientes perante o usuário, podendo requerer arquiteturas particulares para sistemas operacionais ou hardware mais adaptado ao software que se executa sobre ele;
- ✓ Novas arquiteturas baseadas em circuitos programáveis de hardware podem melhorar a velocidade de execução de operações específicas ou emular o projeto de novos sistemas hardware;
- ✓ Progressos recentes em ferramentas de síntese e simulação para circuitos hardware vêm abrindo caminho para a integração de ambientes CAD ao co-projeto hardware-software[6][7].

Embora tenham surgido recentemente diversas metodologias que assim se autodenominam, elas não parecem compartilhar muitas características em comum. A grande maioria dos sistemas digitais atuais consiste de uma combinação de componentes hardware e componentes software trabalhando sobre uma plataforma hardware. Apesar de sua interdependência, esses componentes são projetados utilizando-se de diferentes formalismos, linguagens e ferramentas de representação, simulação e síntese. Dessa forma, podemos considerar que a característica em comum dessas metodologias é sua motivação: fornecer uma maneira de proceder com os projetos dos componentes hardware e software em paralelo, com revisões e interações entre si, onde componentes migram de uma abordagem para outra à medida que são avaliados aspectos de performance, programabilidade, custos de desenvolvimento e fabricação, confiabilidade, potência consumida, manutenção e futuras evoluções do projeto[8][9]. Uma esquema genérico que não se aplica a nenhuma metodologia em particular pode ser visto na Figura 3 a seguir[8].

Apesar de que os conceitos e metodologias mencionados acima são utilizados mais freqüentemente para projetos de sistemas embutidos digitais – fato evidenciado em alguns termos utilizados na Figura 3, como "síntese de hardware", p.e. –, consideramos que a abordagem ainda permanecia válida. Pela natureza distinta do nosso sistema, não poderíamos utilizar nenhuma das ferramentas de co-projeto disponíveis atualmente, no entanto levar em consideração esses conceitos, por si só já trazia benefícios suficientes para justificar sua utilização. Alguns exemplos de decisões de projeto relacionadas a particionamento hardware software seriam, por exemplo:

- o tipo de conversores A/D e D/A a se utilizar – conversores simples, que trabalham com valores digitais cujo escalamento deve ser feito em software, conversores "inteligentes" configuráveis para uma variedade de níveis de tensão de operação ou mesmo conversores com diversas entradas/saídas multiplexadas.
- Filtragem/processamento dos dados adquiridos em hardware, software de baixo-nível (microprocessador, DSP) ou software de alto-nível (computador pessoal)

Seguindo o esquema de co-projeto mostrado na Figura 3: Esquema de Co-Projeto Hardware-Software a seguir, temos que a "Especificação do Sistema" será descrita no Capítulo 3, a "Elaboração de Algoritmos" nos Capítulos 4 e 5, o "Particionamento Hardware-Software" e "Síntese de Interface" no Capítulo 4 e a "Síntese de Hardware" está distribuída entre os Capítulos 4 e 7. Para as atividades reunidas sob a denominação "Síntese de Software", será utilizada uma metodologia específica descrita com maior detalhamento no Capítulo 4 e sua realização está descrita no Capítulo 6. As atividades de Simulação, Verificação e Avaliação estão descritas no Capítulo 8.

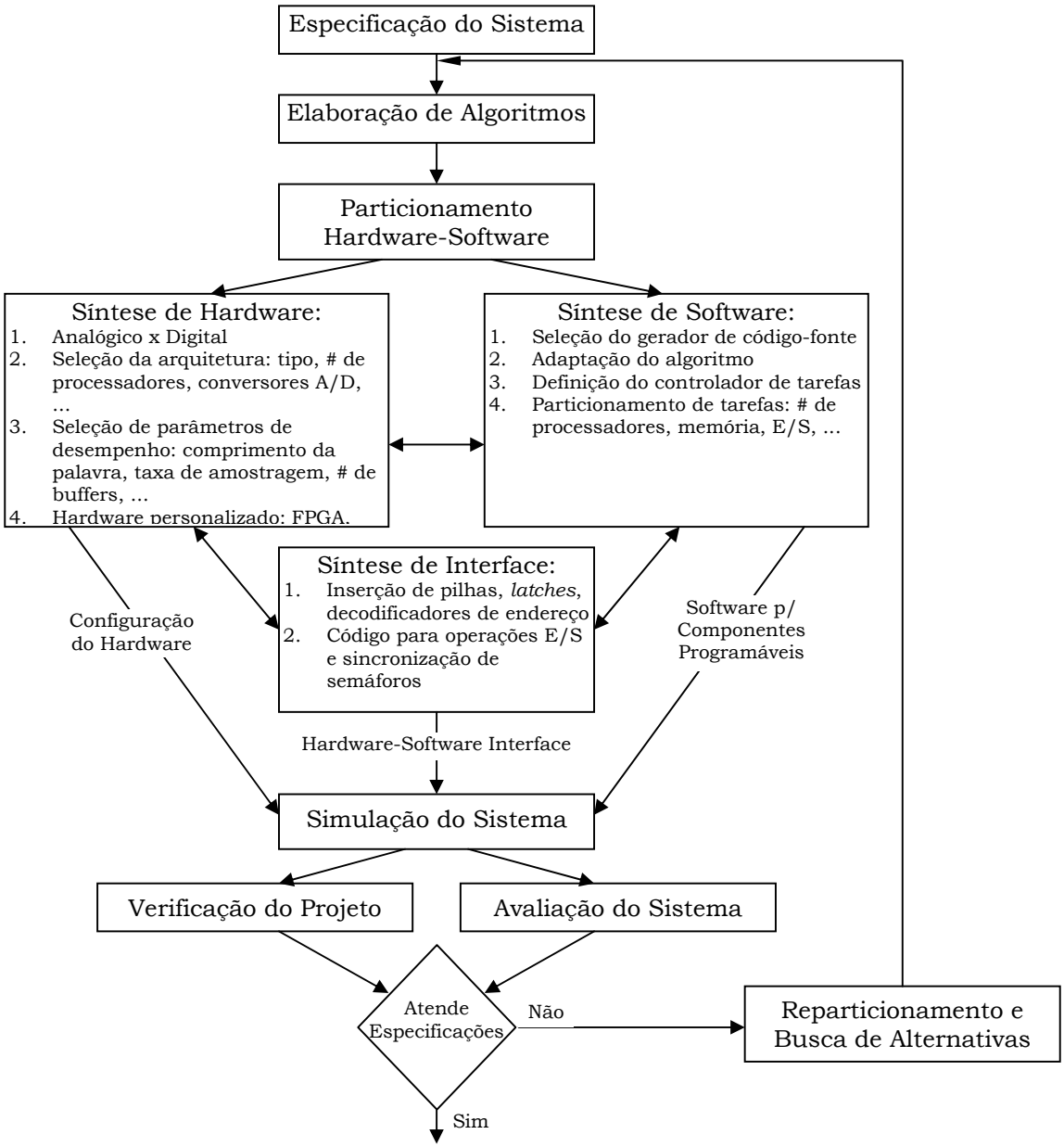


Figura 3: Esquema de Co-Projeto Hardware-Software

Capítulo 2 – Unidade de Controle de Injeção e Ignição Eletrônicas

2.1 Conceitos Básicos

As *centralinas*, também denominadas ECUs (*Electronic Control Units*) são unidades eletrônicas para controlar basicamente as funções de alimentação e ignição do motor. Através de parâmetros calculados com base nas informações provenientes de diversos sensores, elas determinam a quantidade de combustível a ser fornecida a cada cilindro, o avanço de ignição e a energia de cada faísca para permitir uma combustão adequada da mistura.

De acordo com a posição e o número de bicos injetores, podemos classificá-las em: *single-point* – apenas um bico injetor, posicionado na entrada do coletor de admissão ou corpo de borboleta; *multi-point* – um bico injetor para cada cilindro do motor, posicionados próximo às respectivas válvulas de admissão ou *direct-injection* – um bico injetor para cada cilindro do motor, posicionados no corpo do cilindro e injetando combustível dentro da câmara de combustão.

Sistemas de injeção *multi-point* podem ainda ser classificados de acordo com a forma de atuação dos bicos injetores. Na forma mais simples todos os bicos injetores atuam simultaneamente, expelindo uma fração da quantidade de combustível necessária por ciclo motor; mas as mais utilizadas são: seqüencial grupada – os bicos injetores atuam em pares correspondentes a cilindros defasados de 180º, expelindo metade da quantidade de combustível necessária por ciclo motor (em um motor de 4 cilindros, há um grupo para os bicos injetores referentes aos cilindros 1,4 e outro para os referentes aos cilindros 2,3 p.e., atuando alternadamente) – e seqüencial fasada – cada bico injetor atua independentemente, expelindo a quantidade de combustível necessária por ciclo motor e otimizada para cada cilindro (em um motor de 4 cilindros, os bicos atuam na seqüência 1-3-4-2, p.e.). Sistemas *direct-injection* são inerentemente seqüenciais fasados.

As *centralinas* a que se destinam o simulador são do tipo *multi-point* seqüencial fasado, para motores ciclo Otto a gasolina misturada ou não com álcool anidro, e/ou álcool etílico.

Outras funções executadas por *centralinas* são: o controle do fluxo de ar em marcha lenta, através de um *by-pass* na borboleta controlado por um motor de passo ou uma válvula eletromagnética; auto-adaptatividade quando ocorrem mudanças na carga externa ao motor, como o compressor do ar-condicionado, freios ABS ou direção hidráulica; controle das válvulas EGR (*Exhaust Gas Recirculation*) e Cannister (purga dos vapores do tanque); gerência do ângulo de avanço da ignição; monitoramento da tensão de bateria para efetuar ajustes nos tempos de injeção e de carga da bobina de ignição (Dwell); autodiagnose e comunicação com outras *centralinas* e sistemas testadores através de uma linha serial.

2.2 Modo de atuação

A quantidade de combustível a ser injetado deve manter a razão ar-combustível da mistura em torno do valor estequiométrico α=14,4 (que corresponde à razão relativa λ=1). Uma mistura com 10% de excesso de ar (λ=1,1) é requerida para queimar todo o combustível presente, resultando em máxima economia, enquanto que uma mistura com 10% de excesso de combustível (λ=0,9) é requerida para queimar todo o oxigênio, resultando em máxima potência[10][11].

Com informações provenientes dos sensores de rotação, temperatura e pressão do ar, e valores de calibração, as *centralinas* calculam a quantidade de ar entrante no motor a cada ciclo através de um método de medida indireta conhecido como regime motor – densidade de ar aspirado, ou *speed density*[12]. Esse método se baseia no cálculo da massa de ar aspirado para determinação da massa de combustível a ser injetado que garanta a titulação desejada da mistura. As principais equações envolvidas neste cálculo estão mostradas abaixo:

$$V_R = \mu_V \cdot V_T = \mu_V \cdot \frac{RPM}{2} \cdot cc \qquad (1)$$

Vr é o volume real de ar aspirado por ciclo, Vt o volume teórico, μ_V o rendimento volumétrico e cc a cilindrada do motor.

$$M_R = V_R \cdot \frac{P}{T} \qquad (2)$$

Mr é a massa de ar aspirado por ciclo, P a pressão absoluta no coletor de admissão e T a temperatura absoluta do ar aspirado.

$$Q_C = \frac{1}{\alpha} \cdot K \cdot M_R = K \cdot \frac{\mu_V}{\alpha} \cdot \frac{RPM}{2} \cdot cc \cdot \frac{P}{T} \qquad (3)$$

Qc é a quantidade de combustível a ser injetado, K a constante de fluxo do injetor e α a razão ar-combustível desejada.

É principalmente através da escolha da razão ar-combustível, que a *centralina* implementa a estratégia de gerenciamento do motor mais adequada para o momento. Por exemplo, quando ocorrem variações bruscas na posição da borboleta, a *centralina* enriquece a mistura para obter maior potência do motor; da mesma forma, se a borboleta permanece fechada por um certo tempo, a *centralina* suspende totalmente o fornecimento de combustível. Além disso, pequenas variações na rotação do motor, pressão ou temperatura do ar, fazem com que a razão ar-combustível efetiva seja diferente da calculada. O monitoramento da mistura é feito pela sonda lambda, que fornece o valor da razão ar-combustível, a partir da observação da quantidade de oxigênio nos gases de escape.

Além de proporcionar máximo aproveitamento do combustível, o gerenciamento da titulação da mistura também visa minimizar os níveis de emissões gasosas, principalmente de gás carbônico (CO2). Já a formação de óxidos de nitrogênio (NOx) aumenta significativamente com o aumento da temperatura de combustão. Uma forma de diminuir essa temperatura é conduzir os gases de escape de volta à câmara de combustão, para que esses gases inertes diluam a mistura ar-combustível. Essa recirculação dos gases de escape pode ser feita de duas formas: recirculação interna, obtida com temporização adequada das válvulas (*overlap*); ou externa, com a utilização de válvulas EGR. A estratégia de controle dessas válvulas deve regular a adição de gases de escape mantendo um alto nível de torque, baseando-se em informações da posição da borboleta, rotação do motor, temperatura do líquido de arrefecimento e pressão do ar no coletor de admissão[1][13].

Além da composição da mistura, um parâmetro importante para que haja uma combustão adequada é a energia de ativação que desencadeia a explosão, fornecida pela faísca da vela. A ECU controla a quantidade de energia armazenada no campo magnético da bobina de ignição através do tempo em que ela é polarizada antes de cada faísca.

Ao relacionarmos esse tempo com o tempo que dura um ciclo motor completo que por convenção correspondem a 360º (duas voltas da árvore de manivelas, antes conhecida como virabrequim), obtemos o chamado ângulo de Dwell.

Finalmente, a combustão da mistura não é instantânea, e para melhor aproveitamento da energia liberada na explosão, esta deveria acontecer no instante de maior compressão da mistura. Levando-se em conta o tempo de propagação da chama, conclui-se que a faísca deve acontecer alguns instantes antes do ponto morto superior do pistão (PMS). Esse tempo de antecedência, quando relacionado com o tempo de duração de uma volta da árvore de manivelas, corresponde ao ângulo de avanço. Ângulos de avanço menores que o ideal (que varia conforme a condição atual do motor) representam um mal-aproveitamento da energia da combustão, pois o pistão já estará em movimento descendente. Ângulos de avanço maiores que o ideal acarretam no fenômeno de detonação ou “bater-pino” – como o pistão ainda está em movimento ascendente, a explosão atinge seu apogeu e força-o para baixo antes do PMS se opondo ao sentido de rotação do motor.

A Figura 4 a seguir mostra um diagrama simplificado para a família de *centralinas* MIW da Magneti Marelli[14].

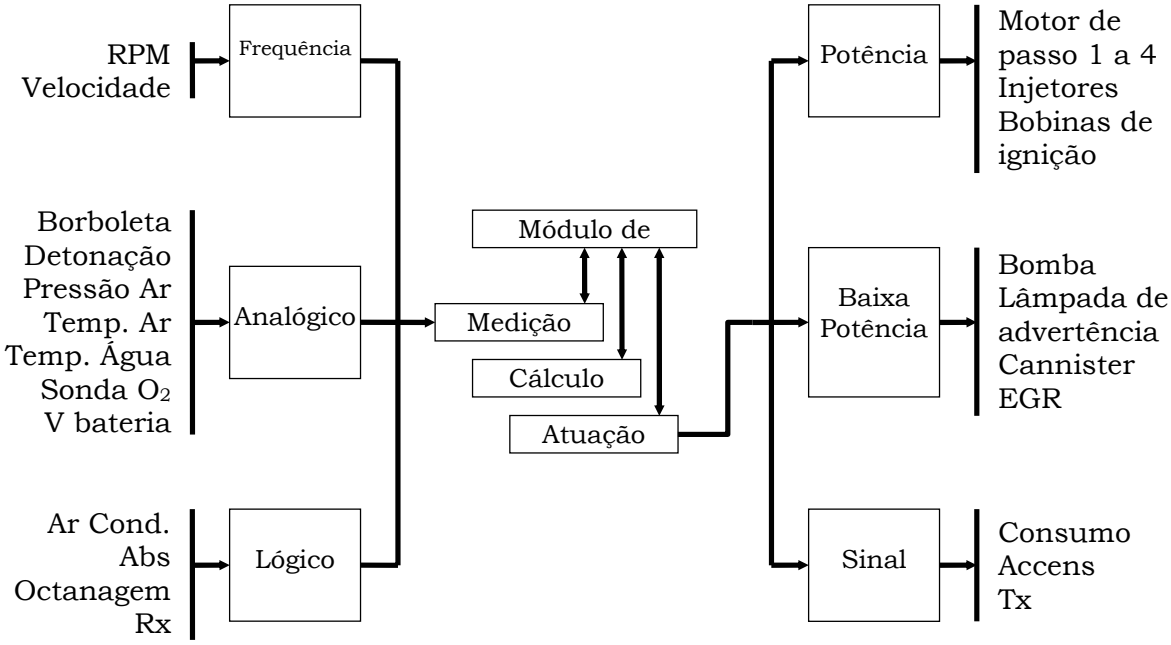


Figura 4: Diagrama de blocos de *centralina* MIW

Capítulo 3 – Elaboração das Especificações

Conforme descrito no Capítulo 1, o sistema simulador a ser desenvolvido tem como objetivos principais efetuar testes e identificar problemas em *centralinas*, solucionando-os quando possível. Adicionalmente, ele deve servir como ferramenta de auxílio ao projeto de novos produtos, antecipando informações importantes quanto ao comportamento de protótipos sem a necessidade de submetê-los a situações reais de rodagem. O sistema deve ser portátil e toda a comunicação entre o simulador e a *centralina* deverá ser feita através de um único cabo.

Apesar da pretensão, esse simulador tem como objetivo final realizar uma simulação *off-line* do ambiente de um veículo, ou seja, os atuadores e outros sub-sistemas complexos seriam modelados e incorporados ao software. Entretanto, esse tipo de simulação só seria alcançável com a utilização de computadores mais rápidos e processamento paralelo utilizando multi-processadores. Devido à limitação dos recursos disponíveis atualmente, necessitamos utilizar atuadores e outras cargas reais para serem conectadas à *centralina* assegurando seu funcionamento adequado.

O sistema deve prover gerenciamento de acesso e privilégios de usuários para evitar mau uso do sistema por pessoal desautorizado. Deverá haver somente um administrador do sistema capaz de realizar cadastramento de usuários, adicionar e/ou remover recursos de interface de hardware e configurar a interface de usuário. Futuras expansões do sistema deverão fornecer ferramentas de auto-configuração dos recursos de hardware e possibilidade dos usuários personalizarem o Painel Frontal do simulador.

O sistema deve ser flexível a ponto de permitir uma fácil atualização a novos dispositivos de geração e aquisição de sinais. A capacidade inicial do sistema deve ser suficiente para:

- Simulação dos sinais de até 16 sensores analógicos (transdutores diversos, termistores NTC e PTC, acelerômetros, sondas lambda, rotação do motor, etc.) e sua desconexão.
- Simulação e verificação do estado de chaves *ON-OFF* e relés.
- Simulação dos eventos *key-on* e *key-off*.
- Cargas reais (bicos injetores, bobinas de ignição, motores de passo, etc.) para serem conectadas e desconectadas à *centralina* com o sistema em funcionamento.
- Aquisição de até seis formas de onda de corrente com informações de temporização.
- Aquisição de sinais PWM e *quadrature clocks*.

Sua configuração para utilização com um determinado modelo de *centralina* deverá ser feita totalmente por software de maneira modular, ou seja, deve haver a possibilidade de reaproveitamento de configurações anteriores. Nesta configuração o usuário fornece ao sistema as características de todos os sensores e atuadores que devem ser conectados à *centralina* em suas condições normais de operação. Com base nas informações de configuração, o sistema deve emitir uma lista de conexões a fim de orientar o usuário para a confecção do cabo que irá conectar esta *centralina* ao sistema simulador.

Os testes são executados através da simulação do ambiente de um veículo para as *centralinas*, ou seja, o sistema deve gerar sinais elétricos correspondendo a todas as grandezas físicas mensuradas pelos diversos sensores que compõe um determinado sistema de injeção e ignição, criando uma situação de funcionamento pseudo-real. A verificação do funcionamento deverá ser feita primeiramente através do monitoramento de grandezas elétricas em cada um dos atuadores do sistema de injeção e ignição. Como ferramenta auxiliar, o sistema deve ser capaz de ativar e obter os resultados das funções de diagnóstico próprias de cada *centralina* através de sua linha de comunicação serial.

Para atender aos requisitos das diferentes aplicações que pretendemos satisfazer com este simulador, devem ser implementadas diferentes estratégias de teste: *open-loop*, *closed-loop* e híbrido[1][15].

O teste *open-loop*, cujo conceito é mostrado na Figura 5, corresponde à simulação estática, onde os sinais correspondentes a cada sensor da *centralina* podem ser acessados e variados manualmente de forma independente, sem o compromisso de representar um ambiente plausível. Esse teste é aplicável às situações onde as condições de teste podem ser estabelecidas pela ação direta do usuário de forma razoável. Uma suposição implícita neste método de teste, é que a resposta do veículo aos comandos da *centralina* pode ser desprezada, compensada ou caracterizada pela maneira que os sinais correspondentes aos sensores são gerados ao longo do tempo. Essa suposição ganha maior embasamento com a utilização de *scripts* de teste que serão comentados a seguir.

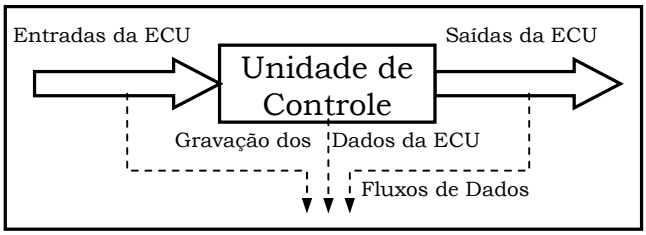


Figura 5: Estratégia de Teste *Open-Loop*

A principal característica da estratégia de teste *closed-loop*, mostrada na Figura 6, é que os dados de entrada são ações do motorista. A partir de um modelamento do comportamento do veículo, o sistema avalia as ações do motorista juntamente com a atuação da *centralina*, para determinar internamente o estado atual do ambiente, calcular o valor do sinal que seria emitido por cada sensor naquela condição, e atualizar os sinais de entrada da *centralina*. Para atingir uma estrutura modular de simulação, o modelamento das dinâmicas veiculares e sistêmicas deve ser particionado em categorias como *powertrain*, chassis e motorista/ambiente, que são divididos ainda em subsistemas.

Cada componente dentro de um subsistema pode ser descrito em diferentes níveis de fidelidade, utilizando-se modelos de maior ordem para as dinâmicas principais e de menor ordem para as secundárias. Estes modelos podem ser analíticos ou empíricos de acordo com a disponibilidade de informações existente. No caso de subsistemas complexos, é muito difícil obter relações paramétricas com soluções em forma fechada.

Nestes casos é mais conveniente a obtenção de uma tabela *look-up* cobrindo toda a região de operação, porém esta poderia se tornar muito grande o que seria inviável com as capacidades limitadas de um PC. Nesses casos deve haver uma ponderação entre a precisão do modelo e a velocidade execução requerida ou disponível. Como alternativa, os dados podem ser divididos em diferentes séries com diferentes equações de regressão cobrindo cada série. Entretanto, devido às descontinuidades em suas extremidades, um único polinômio de maior ordem ou função especial como exponenciais ou trigonométricas seria mais conveniente[2][15].

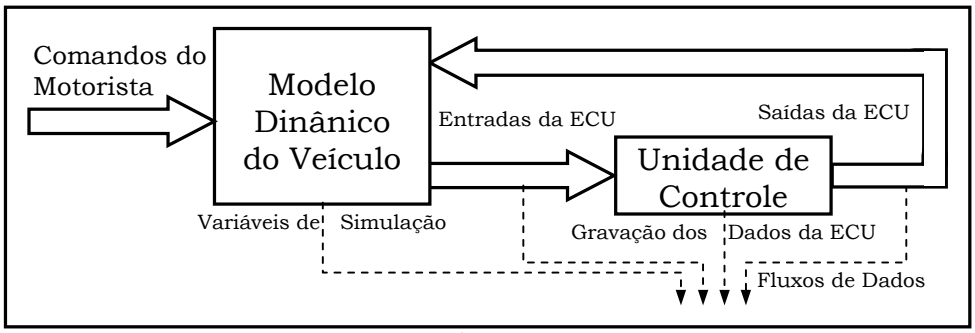


Figura 6: Estratégia de Teste *Closed-Loop*

A estratégia híbrida de testes é uma combinação das duas anteriores, onde alguns sinais da *centralina* podem ser manipulados diretamente, paralelamente à execução do modelamento dinâmico do veículo produzindo os sinais remanescentes. A Figura 7 mostra um esquemático dessa estratégia.

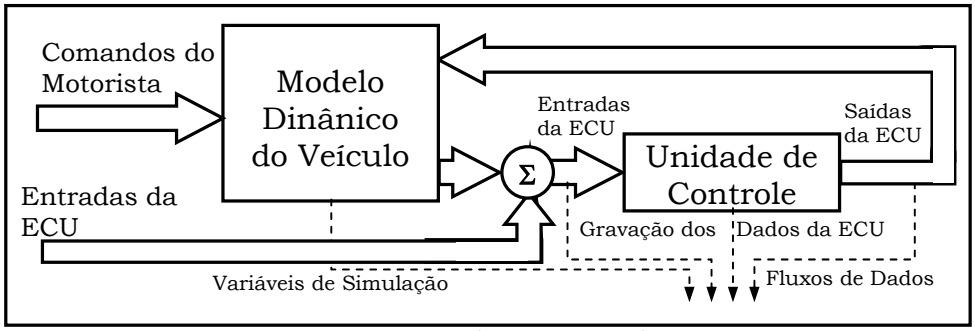


Figura 7: Estratégia de Teste Híbrida

O simulador pode controlar automaticamente a operação das variáveis do sistema usando arquivos de *script*. *Scripts* podem ser criados gravando as ações do usuário em tempo real ou utilizando o editor de *scripts*, onde diversas funções como rampas, ondas quadradas, etc. estão disponíveis. Além destas duas maneiras, um dos mais poderosos recursos da utilização de *scripts* é a reprodução de dados adquiridos de veículos sob teste após a aplicação de rotinas de conversão. Um detalhe importante é que os *scripts* tanto podem ser utilizados para estratégias de teste *open-loop* quanto *close-loop*. A diferença é que no primeiro caso ele conteria os valores dos sinais de entrada que devem ser simulados para a *centralina* e opcionalmente valores esperados para os sinais de saída da mesma. No segundo caso ele os valores contidos representariam as ações do motorista, pois os sinais de entrada para a *centralina* seriam gerados internamente.

Os resultados obtidos com o monitoramento da atuação da *centralina* e das suas funções de diagnóstico próprias devem ser registrados sem o auxílio de outros equipamentos e armazenados em base de dados. A verificação do funcionamento da *centralina* pode ser feito com a definição de critérios *pass/fail*.

A interface de usuário deve ser responsável por carregar as propriedades dos modelos a serem simulados e iniciá-los, executar *scripts*, especificar o arquivo de saída e encadear *scripts* para automatizar testes.

Capítulo 4 – Arquitetura do Sistema

A primeira abordagem considerada buscava desenvolver um sistema proprietário dedicado (módulo de simulação), que se comunicaria com um software rodando em um computador pessoal (PC) através de uma linha serial RS-232. Este módulo seria composto basicamente de: um microprocessador disponível comercialmente, com interface serial RS-232 já incorporada; memória RAM para o programa de teste; dispositivos de geração e aquisição de sinais analógicos (conversores D/A e A/D); temporizadores/contadores para os sinais de freqüência, *duty cycle*, etc.; interface serial compatível com a utilizada pela *centralina*; entradas e saídas digitais 0-12V. Ainda são necessárias, circuitaria para condicionar todos os sinais de entrada e saída aos níveis de tensão e corrente requeridos pela *centralina* e adequados a cada dispositivo de interface e uma placa com conectores para os dispositivos automotivos que, devido à complexidade de suas características de tensão/corrente (geralmente indutivas e variáveis), são inadequados para um modelamento matemático e necessitam estar presentes. Tais dispositivos não simuláveis são:

- 4 bicos injetores MPI
- 1 bico injetor SPI
- 3 bobinas de ignição
- 1 motor DC
- 1 válvula cannister
- 1 válvula EGR
- 4 relés automotivos

Nessa abordagem, imaginamos um banco de dados de *centralinas* baseado em PC. Cada *centralina* seria representada por um registro contendo informações relacionadas com a pinagem e as rotinas de software do respectivo programa de teste a serem desembarcadas no sistema simulador para que ele simule e adquira os sinais envolvidos. Todo o programa de teste deve ser desembarcado no módulo simulador antes de sua utilização para evitar desperdiçar tempo de processamento com comunicação de dados durante a simulação, portanto a memória RAM deve ser suficiente para acomodá-lo. A Figura 8 abaixo mostra o algoritmo básico do sistema e indica as atividades sob responsabilidade do módulo de simulação. A estrutura desse módulo de simulação pode ser vista na Figura 9.

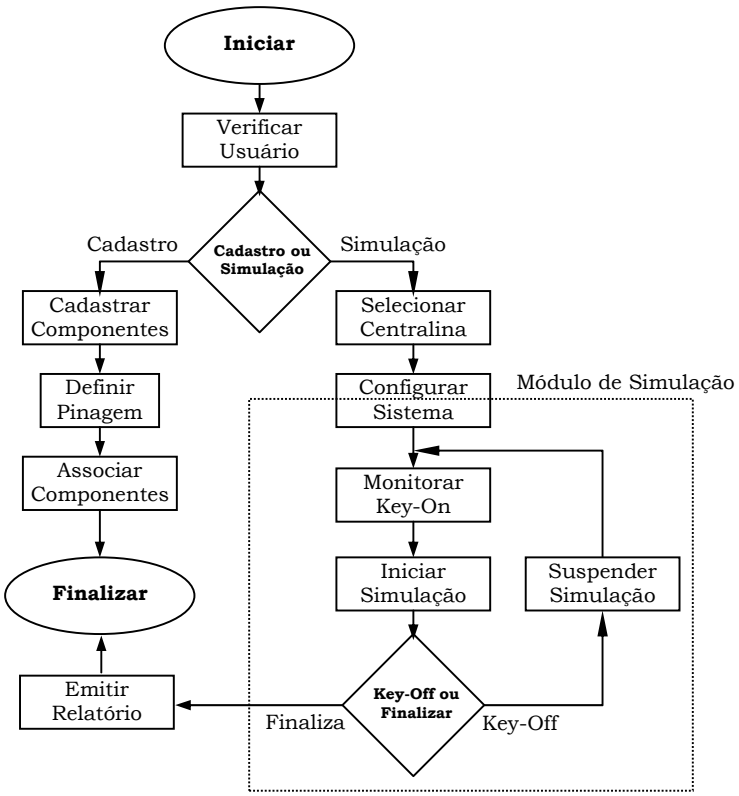


Figura 8: Algoritmo básico do Sistema Simulador

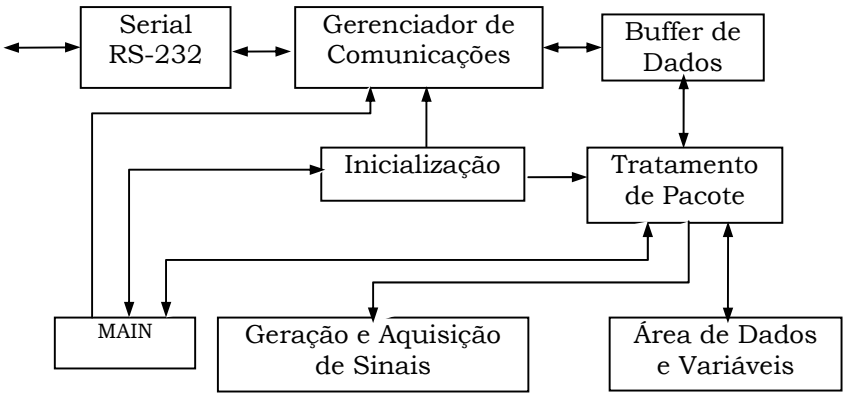


Figura 9: Estrutura do Módulo de Simulação

A função de cada bloco na figura acima pode ser descrita como se segue:

- Serial - Montagem dos bytes e teste de paridade para verificar bytes adulterados.
- Gerenciador de Comunicação - Controla o enchimento do *buffer* e a estrutura dos pacotes, verificando seu comprimento e *checksum*.
- Tratamento de Pacote - Analisa o tipo de mensagem e carrega as rotinas de simulação ou dados na área de RAM correspondente. Também monta as mensagens que devem voltar para o computador (dados adquiridos).
- Inicialização - Controla o Gerenciador de Comunicação e o Tratamento de Pacote até receber a rotina MAIN, quando deixa de atuar e lhe passa o controle.
- MAIN - Software específico para cada *centralina*, que gerencia o recebimento das rotinas de simulação, fornecendo os respectivos endereços de armazenamento para o Tratamento de Pacote. Ao término desse processo é o responsável pelo gerenciamento de tarefas, dividindo o tempo de processamento entre comunicação, geração e aquisição de dados.
- Rotinas de Simulação - Softwares específicos para cada modelo de sensor ou atuador que quando associados a um programa MAIN, definem um modelo de *centralina*.

A partir dessa idéia inicial, começamos os primeiros estudos de particionamento hardware/software. Nesse momento as necessidades de modularidade, facilidades de manutenção e atualização e principalmente um curto tempo de desenvolvimento, levaram o sistema a se reestruturar e incorporar o máximo de elementos em software, além de nos direcionar a utilizar, sempre que possível, elementos hardware *off-the-shelf*.

Para a reestruturação desse módulo de simulação, utilizamos o conceito de instrumentação virtual, onde cada instrumento é um co-projeto hardware-software que pode agregar várias funções distintas. A decisão sobre o que deve ser implementado em hardware e software, visa um aproveitamento máximo das vantagens de cada uma destas abordagens e já deve considerar um computador pessoal como plataforma.

Não cabe aqui discutir se esse é um fato positivo ou negativo, mas a realidade é que essa decisão nos será de certa maneira imposta pelos fabricantes de equipamentos de instrumentação disponíveis no mercado. Portanto placas de geração e aquisição de sinais (DAQ *boards*) realizam as funções de interface com o mundo exterior e todo o processamento é feito ao nível de software. Dessa forma, uma mesma configuração de hardware pode ser compartilhada por vários instrumentos, cujos desempenhos melhoram automaticamente, à medida que a plataforma é atualizada. O instrumento também se beneficia do desenvolvimento de novos sistemas operacionais, arquiteturas cliente-servidor, tecnologias de conectividade com alcance mundial como a Internet e de compartilhamento de dados como OPC (OLE f*or Process Control*) [16].

4.1 Definição de Plataformas de Hardware e Software

A escolha da plataforma de software ocorreu paralelamente à definição do hardware. Isso implicou que a plataforma fornecesse *drivers* para as placas escolhidas, facilitando ao máximo a comunicação entre ambos. Já neste primeiro aspecto, os softwares LabVIEW e LabWindows/CVI se mostraram a opção mais adequada, visto que eles possuem a mais vasta biblioteca de *drivers* para DAQs, dos mais diversos fabricantes. Outros ambientes avaliados que não atenderam aos requisitos do sistema foram: ADAC TestPoint e HP VEE, por não possuírem os *drivers* para DAQs necessários. Compiladores C++ em geral demandariam um enorme tempo de desenvolvimento e grandes esforços para futuras atualizações. O mesmo vale para ambientes como Delphi e Visual Basic. Como ambos são funcionalmente equivalentes e possuem um compilador que gera programas executáveis, a escolha final foi pelo LabView, unicamente por ser um ambiente de programação gráfica com alguns conceitos de orientação a objeto, que propiciaria uma maior facilidade para atualizações futuras e reduziria substancialmente o tempo de desenvolvimento do sistema.

Para garantir total compatibilidade entre o software e o hardware e pela comprovada qualidade, optamos por placas DAQ fabricadas também pela National Instruments. As placas deveriam possuir diversos canais analógicos de entrada e de saída, com taxas de amostragem da ordem de 100KS/s, linhas digitais de entrada e saída e contadores/temporizadores. As placas escolhidas para satisfazer essas necessidades foram:

- AT-AI-16XE-10
  - 16 canais de entrada analógica *single-ended* ou 8 diferenciais, c/resolução de 16 bits a uma taxa de amostragem de 100KS/s
  - 8 linhas TTL de E/S
  - 2 contadores/temporizadores de 24 bits c/ freqüência máxima de 20MHz
- AT-AO-6 e AT-AO-10
  - 6 ou 10 canais de saída analógica, c/ resolução de 12 bits e uma taxa de atualização máxima de 300KS/s, unipolares ou bipolares
  - 8 linhas TTL de E/S

Para a validação do projeto, o simulador foi desenvolvido em uma versão de bancada, e somente numa segunda etapa ele será implementado em um laptop. Essa atualização já foi levada em conta durante a elaboração do software, garantindo um esforço mínimo de desenvolvimento para implementá-la. A plataforma de hardware atual é um IBM PC, com processador de 100MHz e 64MB de RAM. O gabinete IBM se mostrou fundamental para esta implementação, pois foi o único no qual pudemos alocar 3 placas ISA com mais de 30 cm cada! Esse problema não ocorrerá na versão laptop, no entanto necessitaremos de vários *slots* PCMCIA tipo II e atualização rápida de vídeo, o que limitará a escolha do modelo apropriado.

Após essas definições, passemos a uma revisão da organização do sistema em relação à abordagem com o módulo de simulação descrita na Figura 9 acima. Com a utilização de placas DAQ fabricadas, estamos incorporando o módulo de simulação física e logicamente ao PC, com a garantia extra de estarmos lidando com componentes eletrônicos de alta precisão e com desempenho já bem caracterizado e de eficiência comprovada.

O bloco “Geração e Aquisição de Sinais” é implementado através dos canais analógicos de entrada e saída, contadores/temporizadores e linhas TTL das placas. Os blocos de software “MAIN” e “Rotinas de Simulação” passam a ser armazenados na mesma mídia (*hard-disk*), executados pelo mesmo processador (Pentium 100) e implementados na mesma linguagem de alto nível (LabVIEW 4.1) utilizada para o banco de dados de *centralinas*, eliminando a necessidade dos blocos “Serial RS-232”, “Gerenciador de Comunicação”, “Tratamento de Pacote”, “*Buffer* de Dados” e “Inicialização”. A “Área de Dados e Variáveis” passa a ser a própria RAM do PC. A interface entre as placas e o software é o barramento ISA do PC (PCMCIA na versão laptop) e a comunicação é gerenciada pelos *drivers* fornecidos pelo fabricante.

Para completar a descrição do processo “Síntese de Hardware”, nos falta ainda a circuitaria de condicionamento dos sinais de entrada e saída para os níveis de tensão e corrente adequados às placas DAQ e as cargas automotivas esperadas pela *centralina*. Apesar de também haverem diversos tipos de equipamentos fabricados pela National Instruments para o condicionamento de sinais e até para a simulação de alguns tipos de cargas, decidimos realizar o desenvolvimento de um módulo de condicionamento de sinais específico para nosso sistema.

O problema é que a *centralina* possui diversas funções de auto-diagnose e adaptação a situações de falha de um ou mais componentes. Como cargas indutivas como os bicos injetores e as bobinas de ignição não seriam simuláveis de forma satisfatória, correríamos o risco de “falsear” os resultados da simulação e por isso decidimos embutir cargas reais no mencionado módulo de condicionamento de sinais. Esse módulo possui um conector para cada placa (os cabos são fornecidos pelo fabricante) e um conector para a *centralina*, cujo chicote deve ser construído pelo usuário para cada modelo de *centralina*, baseado em informações fornecidas pelo sistema simulador. A versão de bancada está esquematizada na Figura 10 e será descrita em detalhes no Capítulo 7.

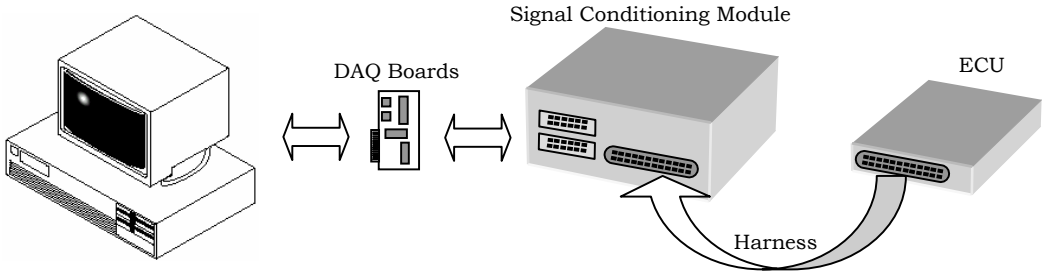


Figura 10: Esquema Físico do Simulador Versão Bancada

Capítulo 5 – Modelamento

O grande desafio para o desenvolvimento de sistemas de engenharia baseados em software é tornar essa atividade um processo industrial. Por ser um ramo de atividade relativamente novo, ao contrário de sistemas de gerenciamento de bancos de dados e automatização de tarefas administrativas, ele ainda está por alcançar um nível de maturidade que estabeleça práticas racionais já consensadas pelo mercado para seu desenvolvimento e exploração como produtos comerciais. Sem detrimento da criatividade como principal ferramenta do projetista, o que se busca é um processo de desenvolvimento que englobe todo o ciclo de vida do projeto, desde a especificação dos requisitos até as atividades de manutenção e aprimoramento. A cada etapa do projeto, esse processo deve gerar resultados palpáveis e independentes do indivíduo que o execute, portanto qualquer pessoa qualificada para uma operação deve ser capaz de executá-la de forma similar; permitindo a alocação de tarefas para vários grupos até mesmo subcontratados; fazendo uso (e reuso) de blocos de construção e componentes predefinidos e com a possibilidade de planejar e calcular o processo com grande precisão.

O desenvolvimento de um sistema pode ser visto como um processo de produção de descrições que o modelam. Até mesmo o código fonte da versão final é um modelo que pode ser compreendido pelos programadores e pelo compilador. Essas descrições são feitas através de modelos com um detalhamento que vai sendo incluído progressivamente, à medida que os modelos se tornam mais formais e incorporam mais estrutura e menos abstração ao sistema.

Sendo o avanço tecnológico inevitável, torna-se de capital importância prevermos o fator obsolescência e obtermos o máximo de vida útil para o sistema. A única maneira de garantir essa premissa, seria a utilização de uma abordagem que possua as características descritas acima intrinsecamente, e tais abordagens são necessariamente orientadas a objeto. Após a avaliação de diversos métodos como: Object Modelling Technique (OMT)[17], Fusion[18] e Object-Oriented System Analysis (OOSE) [19]; optamos por uma variante deste último conhecida como Objectory[19].

O primeiro modelo do Objectory é o modelo de requisitos, que identifica os *use cases*. A partir dele, iremos desenvolver os outros modelos de forma incremental e realimentada até chegarmos ao modelo de implementação, que corresponde ao próprio sistema desejado. Esse desenvolvimento é incremental porque as particularidades do sistema que dependem do ambiente de desenvolvimento específico devem ser inseridas gradualmente, gerando modelos cada vez mais detalhados. Isso garante uma base sólida e quase imune a mudanças, pois o sistema deve prescindir do ambiente como especificado nos requisitos. E é realimentado porque à medida que avançamos, encontramos inconsistências que devem ser resolvidas ou alterando-se um pouco a estrutura do modelo problemático, ou voltando atrás e remodelando o comportamento inconsistente em modelos anteriores. A decisão sobre qual alteração a ser feita deve considerar a manutenção da rastreabilidade entre os modelos com a independência do ambiente em modelos anteriores.

Também é importante salientar que um modelamento orientado a objeto não implica necessariamente no uso de uma linguagem orientada a objeto. Aqui precisamos deixar claro alguns pontos:

- o LabVIEW em sua versão 4.1 não suporta diretamente programação orientada a objetos
- o LabVIEW só permite a declaração explícita de classes a partir da versão 6.0
- As ferramentas para UML disponíveis não possuem geração automática de código para LabVIEW

Mesmo assim consideramos que a linguagem de programação gráfica LabView 4.1 era a escolha adequada e consideramos possível e importante tentar incorporar as principais características de orientação a objeto em nosso sistema como: encapsulamento de dados e funções em objetos; abstração e polimorfismo. Obviamente a transição dos modelos para o código fonte será muito mais dificíl, mas muitos comportamentos podem ser emulados e outros aproximados, como veremos no Ítem 5.6.

Para a representação dos diversos modelos do Objectory utilizamos a UML – Unified Modelling Language em sua versão 1.1[20]. Essa linguagem representa um esforço conjunto de diversas empresas e organizações – liderado pelos renomados metodologistas Grady Booch, Ivar Jacobson e Jim Rumbaugh através de sua empresa Rational Software – em padronizar as diversas representações utilizadas em métodos de desenvolvimento de software, aproveitando as vantagens apresentadas por cada um deles e sofrendo atualizações à medida que novas idéias vão surgindo. A elaboração dos modelos foi feita com a ferramenta CASE (Computer Aided Software Engineering) Rational Rose em sua versão Demo. Infelizmente essa versão da ferramenta permite apenas um número limitado de entidades por projeto (30 classes e atores, 10 usos de caso, 10 estados) e os modelos gerados não puderam ser totalmente integrados através dela. Como não iríamos necessitar das funções de geração automática de código porque nosso sistema seria implementado de qualquer forma em LabVIEW, essa limitação foi contornável.

Desde o início deste trabalho em 1998, a UML continuou sendo aperfeiçoada e hoje está na versão 2.0 sendo amplamente utilizada em quase todos os segmentos da indústria, inclusive no desenvolvimento do software das próprias unidades eletrônicas a serem testadas por este sistema. Essa aceitação só foi possível porque a UML é o resultado de um fenomenal trabalho que uniu os maiores especialistas acadêmicos em programação orientada a objeto junto com indústria. Em nossa opinião, isso demonstra que a escolha foi mais que acertada. A Figura 11 abaixo mostra o processo de desenvolvimento da UML e a Figura 12 mostra a origem das mais importantes contribuições nesse processo.

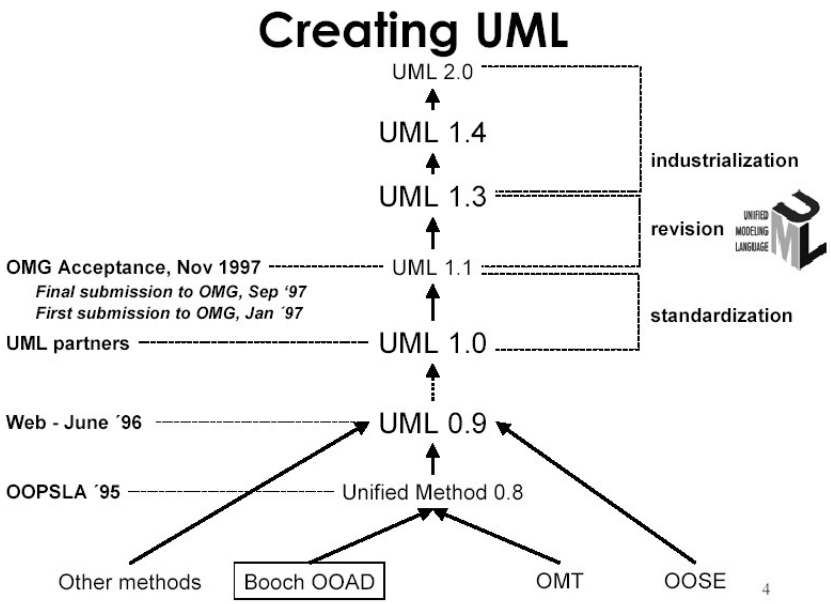


Figura 11: Desenvolvimento da linguagem UML

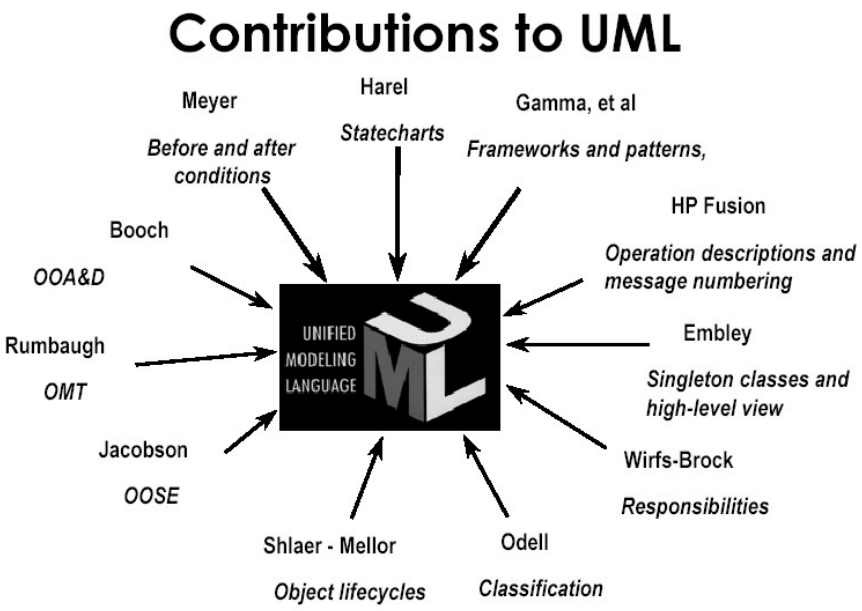


Figura 12: Contribuições para a linguagem UML

5.1 Modelo de Requisitos – Diagramas de *Use Cases*

O primeiro modelo a ser criado para o nosso sistema é o Modelo de Requisitos. Esse modelo deve representar a funcionalidade do sistema sob o ponto de vista do usuário. Ele parte de um documento de requisitos, que nada mais é do que uma descrição textual do comportamento esperado e das funções executadas pelo sistema, com o auxílio de figuras onde houver necessidade; e mais importante, elaborado em conjunto entre seus projetistas e usuários. Nosso documento de requisitos já foi descrito resumidamente no Capítulo 3, na seção Elaboração das Especificações. Esse modelo começa com a identificação dos atores – onde estes representam os papéis que um usuário pode ter perante o sistema – e não uma pessoa em particular que irá interagir com ele.

O segundo passo consiste em identificar os *use cases* possíveis para cada ator. Cada *use case* deve representar uma tarefa a ser executada pelo sistema, a fim de satisfazer o conjunto de tarefas que caracteriza o papel de cada ator. O conjunto dos *use cases* para esses atores deve representar toda a funcionalidade disponível no sistema. Gostaríamos de enfatizar o caráter modular desse método, salientando que novas funcionalidades podem ser acrescentadas ao sistema incluindo-se novos *use cases* no modelo atual, sem alterar em nada sua essência e reaproveitando comportamentos semelhantes. Somente após todos os *use cases* estarem identificados, buscaremos estabelecer relações de herança entre os atores, procurando aqueles que possam exercer todas as funções de um outro e mais alguma(s). Para a *centralina*, foram identificados 5 atores, que podem ser vistos na Figura 13 abaixo com suas relações de herança e seus respectivos casos de uso.

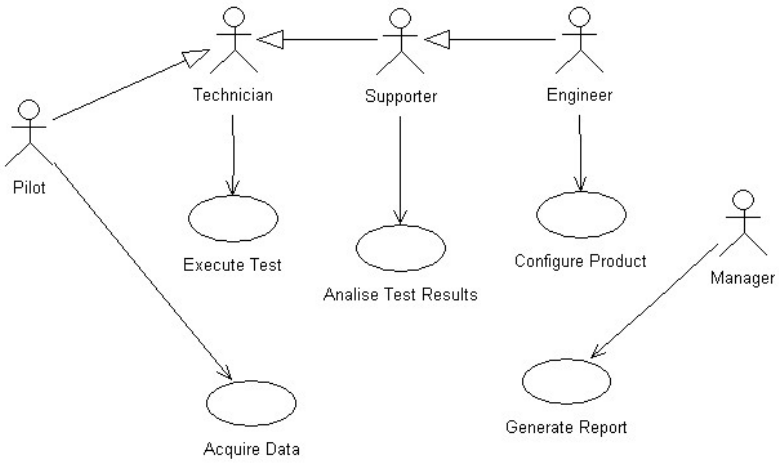


Figura 13: Atores e casos de uso identificados

No escopo deste trabalho foram considerados apenas 3 deles: Técnico, Engenheiro e Gerente;.que podem ser vistos na Figura 14 a seguir.

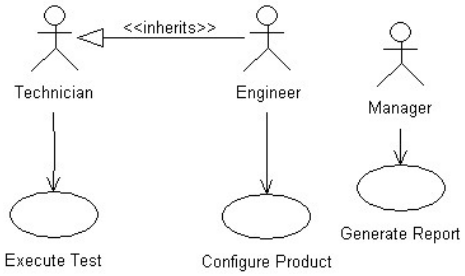


Figura 14: Atores e casos de uso relevantes para nosso sistema

Nem todos os diagramas de *Use Cases* necessitam de atores. De acordo com a definição já mencionada acima, cada *use case* deve representar uma tarefa a ser executada pelo sistema. Quanto mais complexo o sistema, mais difícil se torna detalhar todas as tarefas em um único diagrama porque ele se tornaria incompreensível. Neste caso a solução é utilizarmos. A decisão se uma sub-tarefa deve ou não ser descrita como *sub use case* é de ordem prática, basta avaliar se ela será utilizada ou não por outras tarefas. Os *sub use cases* para o *use case* "Execute Test" podem ser vistos na Figura 15 abaixo.

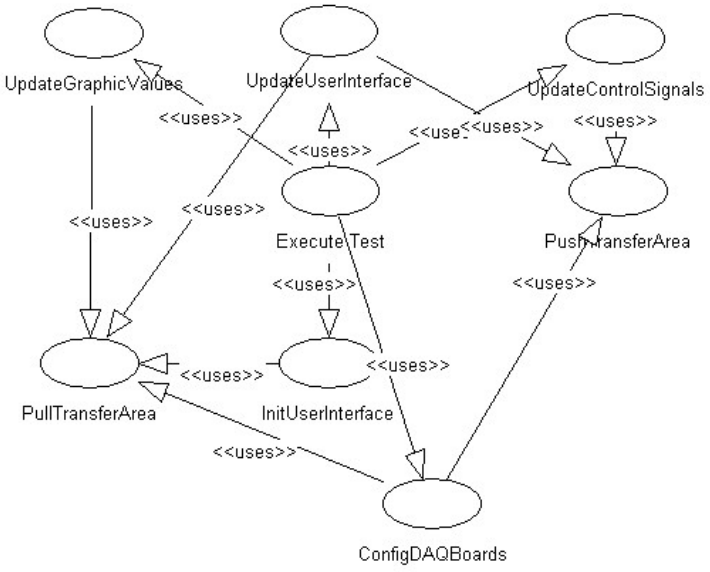


Figura 15: Diagrama de *Use Cases* para "Execute Test"

Para completar a descrição funcional do sistema simulador, apresentamos no Apêndice A alguns dos *use cases* identificados.

5.2 Modelo de Objetos do Domínio do Problema – Diagrama de Classes

O próximo modelo a ser criado é o Modelo de Objetos do Domínio do Problema, descrito principalmente através de Diagramas de Classes. Esse modelo deve representar a estrutura e sub-estrutura do sistema através de classes, atributos, operações ou métodos e associações. Estudando-se os *use cases* deve-se identificar toda referência a possíveis entidades que contenham informações geradas pelo ou necessárias para o sistema, bem como entidades que executem tarefas. Em seguida, devem ser identificadas relações de dependência entre essas entidades por interpretação do texto dos *use cases*. Obviamente esta é uma tarefa iterativa, onde a interpretação abstraída do texto deve ser continuamente verificada com os clientes e futuros usuários do sistema para verificar se ela é correta e atende as especificações dadas. As entidades identificadas são chamadas de Classes e sua definição bem como a definição das relações de interdependência entre elas é a etapa mais crítica de todo desenvolvimento de sistema orientado a objeto. Não iremos descrever aqui as discussões intermináveis durante essa etapa do projeto, neste ponto infelizmente com pouco apoio da parte mais interessada – nosso patrocinador –, mas iremos descrever os pontos principais da solução proposta.

Da análise dos *use cases* apresentados no Apêndice A, identificamos rapidamente Classes como base de dados, teste, produto, componente, área de transferência e muitas outras. Para exemplificar, do Modelo de Requisitos temos que a base de dados é composta de produtos, componentes e testes. Um Produto é composto por diversos componentes associados a um padrão de pinagem e um painel frontal e é capaz de executar um ou mais testes. A relação entre essas principais classes do sistema pode ser vista na Figura 16.

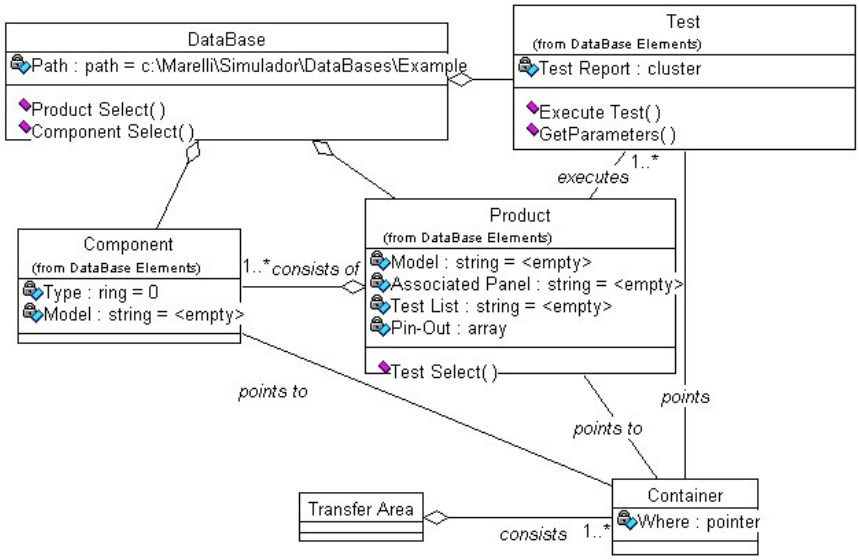


Figura 16: Principais classes do Sistema simulador

Obviamente esse único diagrama acima não é suficiente para descrever toda a estrutura do sistema. Estudando os requisitos apresentados no Capítulo 3 e os tipos de sensores e atuadores com os quais as *centralinas* estão conectadas, pudemos classificar esses componentes em nove grupos distintos, conforme sua função para a *centralina* e a forma como seus respectivos sinais gerados/adquiridos serão tratados pelo simulador. Esse detalhamento das classes Produto e Componente pode ser visto na Figura 17 abaixo. Propriedades comuns a todos os tipos de componente como "Tipo" e "Modelo" pertencem à classe genérica "Componente" e cada um dos distintos tipos possui suas propriedades específicas como "Tabela de Valores" ou "Valor Máximo". Por fim, a Figura 18 na próxima página mostra mais algumas classes importantes para o sistema

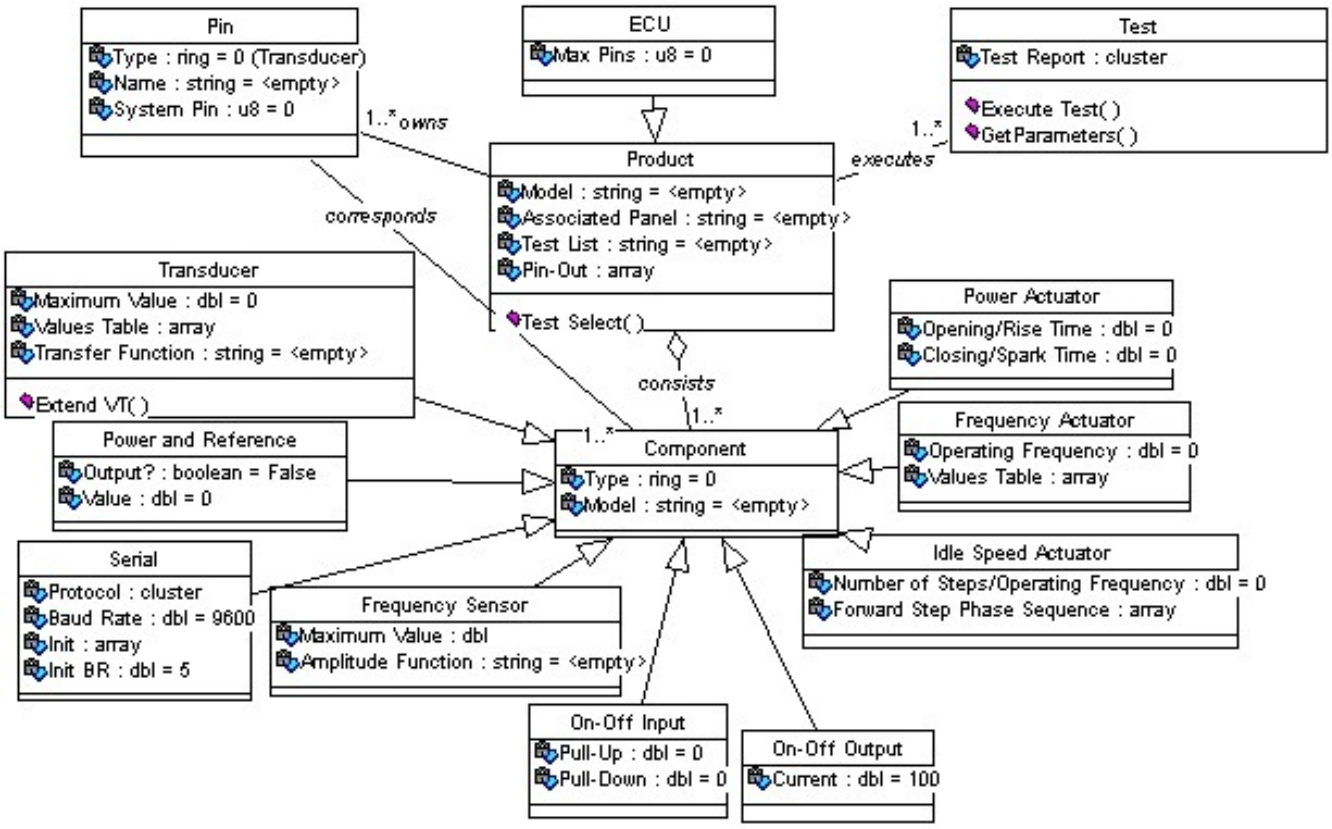


Figura 17: Detalhamento das classes Produto e Componente

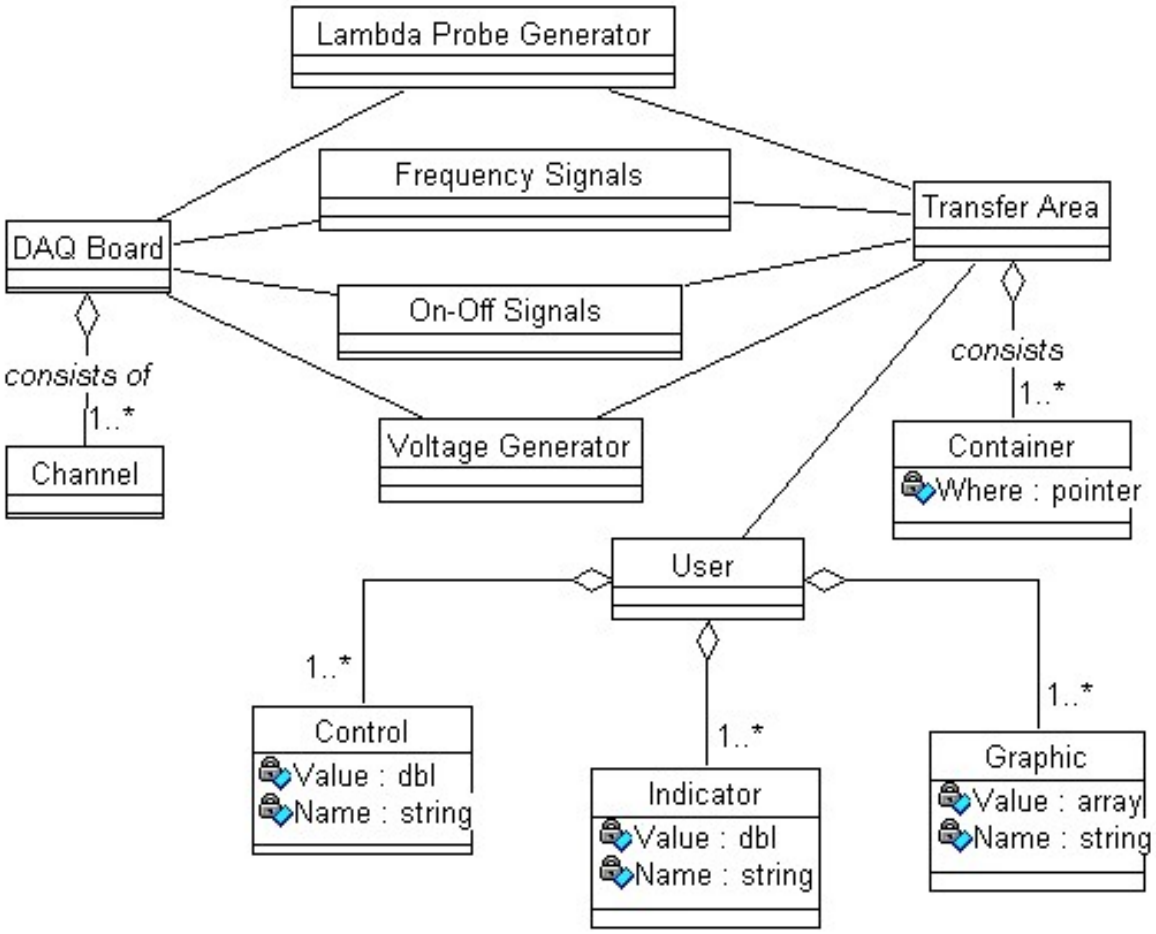


Figura 18: Outras classes importantes

5.3 Modelo Dinâmico – Diagramas de Objetos

Como pudemos observar, os diagramas de classes representam a estrutura estática do sistema, quais os tipos de informação a serem manejados e como eles estão associados, mas não têm nenhuma informação quanto ao momento em que cada tipo de informação deverá ser utilizado. Os diagramas de classes são atemporais.

Para representar o comportamento interno do sistema, utilizamos o Modelo Dinâmico. Esse modelo se compõe de distintos tipos de diagramas, tantos quantos se façam necessários para conseguir abranger os diversos pontos de vista necessários para possibilitar uma análise detalhada do sistema. Ao contrário dos diagramas de classes que representam todas as possibilidades do sistema de uma só vez (independente se essa possibilidade está ou não sendo utilizada), utilizaremos agora Diagramas de Objetos, que mostram uma foto do sistema em um determinado instante e sob um determinado foco. Os elementos desses diagramas são "objetos", "links" e "mensagens" – as instâncias (realizações em tempo de execução) de classes, associações e solicitações de execução de métodos.

Os principais tipos de diagramas de objetos são os Diagramas de Colaboração (foco na troca de mensagens entre objetos) e os Diagramas de Seqüências (foco na interação entre objetos). Nas Figuras 19 e 20 a seguir mostramos exemplos de diagramas de colaboração iniciados por um objeto e por um usuário do sistema.

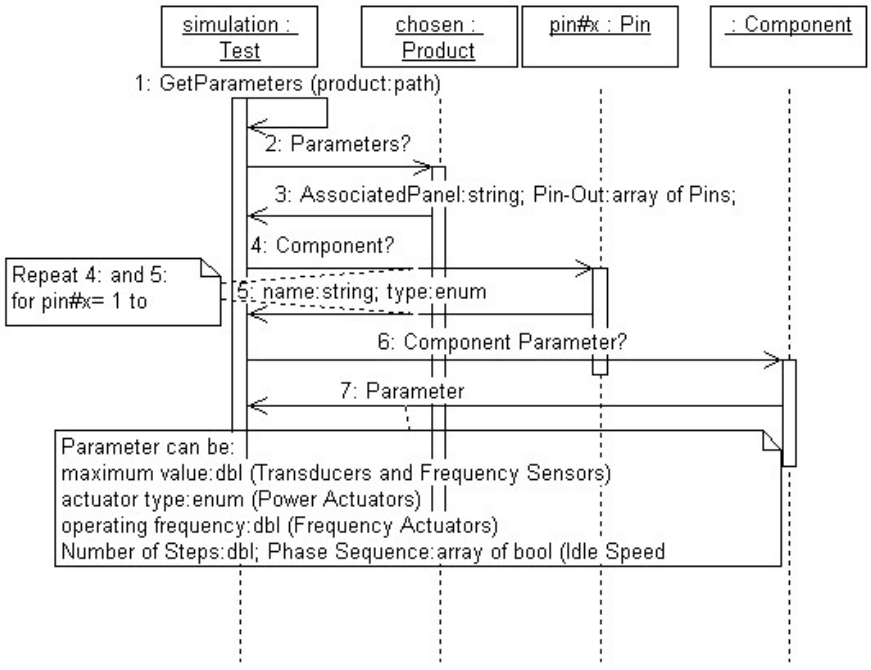


Figura 19: Exemplo de Diagrama de Colaboração iniciado por objeto

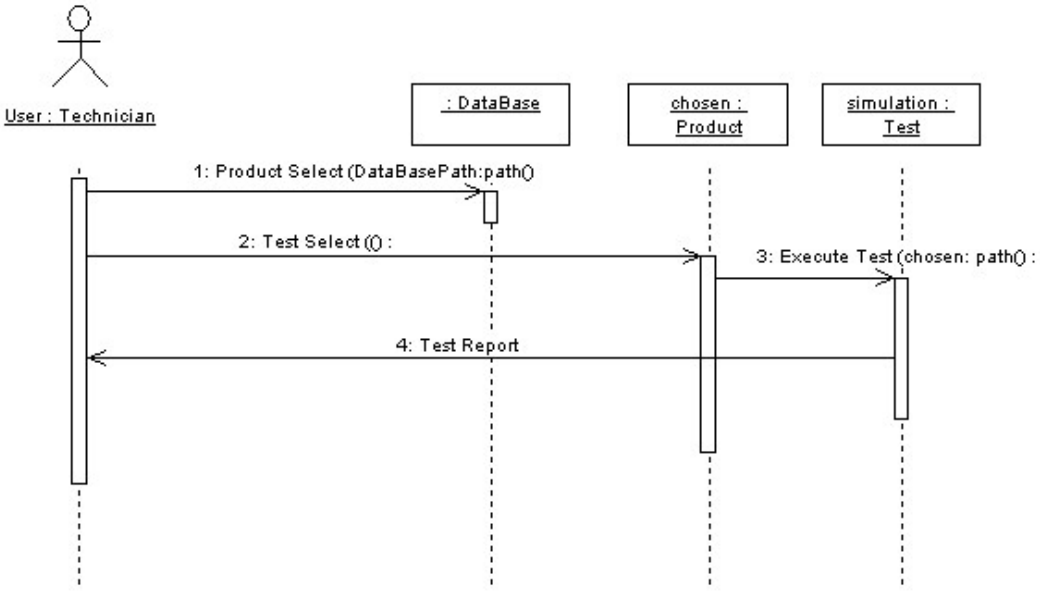


Figura 20: Exemplo de Diagrama de Colaboração iniciado por usuário

Não iremos nos estender mais nesses dois diagramas não somente porque sua representação gráfica ocupa muito espaço, mas principalmente por uma particularidade do LabVIEW: por ser uma linguagem de programação gráfica, o código-fonte de um sistema aí desenvolvido se assemelha muito a diagramas de objetos. Ou seja, quase toda a informação presente nos diagramas de objetos pode ser ou diretamente inferida do código-fonte ou indiretamente com ajuda dos diagramas de estado, com os quais iremos dedicar especial atenção no Item 5.4 logo a seguir. No Capítulo 6 iremos ainda examinar o código-fonte das principais sub-rotinas do sistema.

5.4 Modelo Dinâmico – Diagramas de Estados

Um diagrama de estados é uma representação gráfica do comportamento dinâmico de uma ou mais classes. Cada estado representa um estado de equilíbrio (estático ou dinâmico) possível para esta classe, ou seja, um estado não representa necessariamente uma situação de espera, mas sim uma determinada seqüência de atividades que será executada para um determinado contexto.

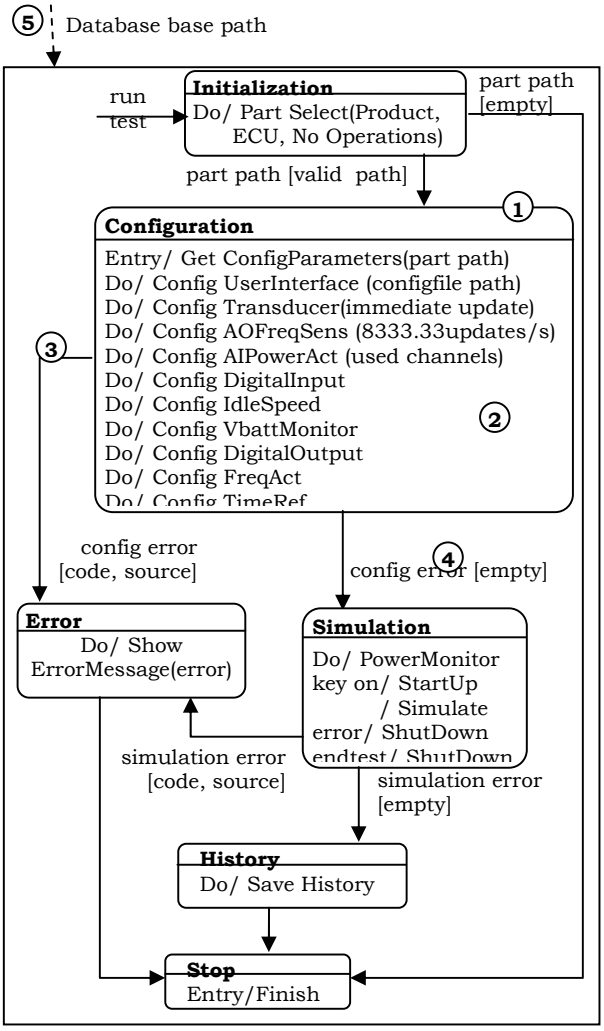


Figura 21: Diagrama de Estados para a classe Teste

Neste diagrama também estão representadas todas as possíveis transições de estado com os respectivos eventos que condicionam essa transição. Os eventos que iniciam uma transição podem ser tanto externos à classe em questão (mensagens) como também resultado de operações internas ou até mesmo ocorrências periódicas.

A Figura 21 na página anterior mostra o diagrama de estados para a classe "Teste" já mencionada na Figura 17. Nessa figura podemos reconhecer estados (1), atividades a serem executadas neste estado (2), transições (3), eventos de transição (4), mensagens (5), etc.

As atividades dentro de cada estado estão descritas de forma seqüencial, porém algumas delas são iterativas e sua conclusão depende também do estado de outras classes. Nestes casos, necessitamos de diagramas de sub-estados, para descrever o comportamento dinâmico dessas outras classes. A Figura 22 mostra o diagrama de sub-estados para a atividade "Simulate" dentro do estado "Simulation" da Figura 21 anterior. Um elemento interessante neste diagrama é a ativação simultânea de diversos estados em diferentes classes (6).

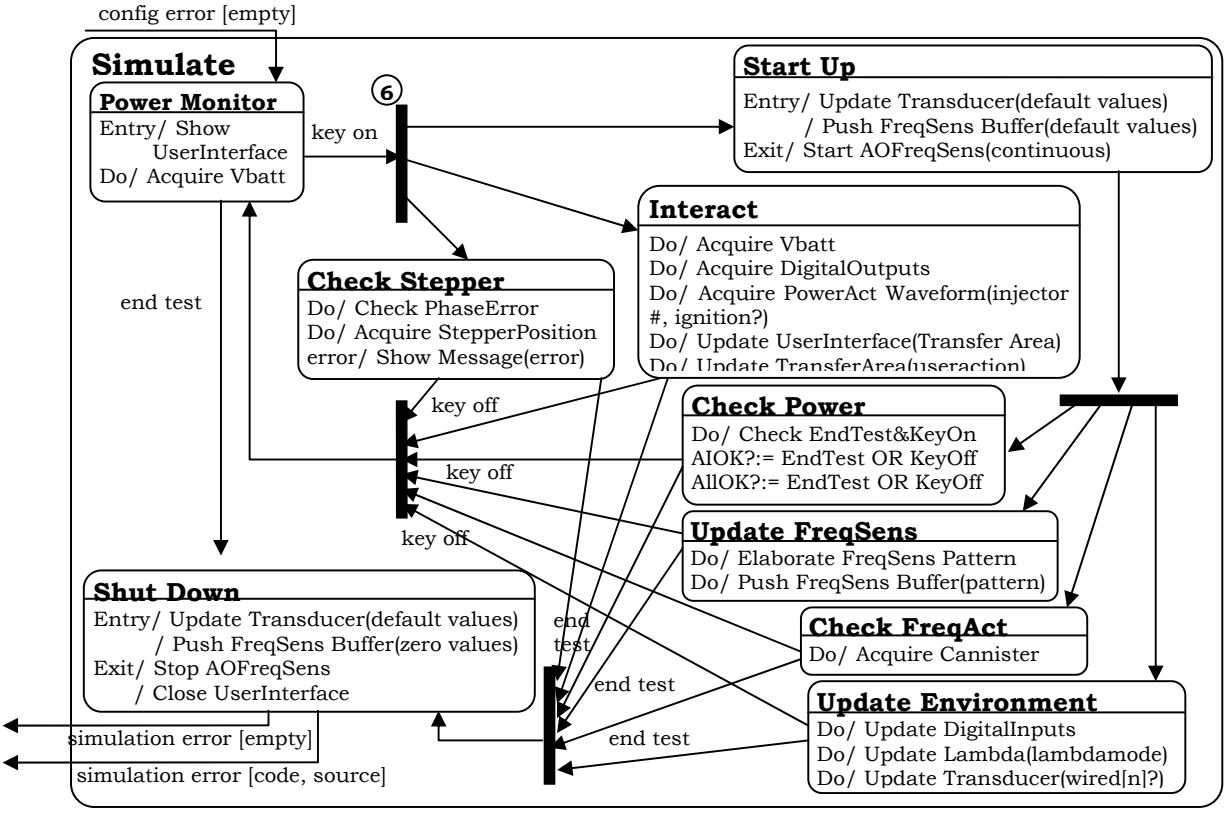


Figura 22: Diagrama de Estados para a Simulação

A tradução de um diagrama de estados para código de programação é basicamente a conversão de um modelo direcionado a eventos para um modelo seqüencial, a dificuldade é como definir a periodicidade de verificação da ocorrência de um possível evento que dispare uma transição de estado sem comprometer de forma significativa a funcionalidade do sistema. A solução em LabVIEW utilizada por nosso sistema é uma mescla de conceitos adquiridos através de uma lista de discussão com outros programadores espalhados pelo mundo com alguns toques pessoais. A Figura 23 mostra os elementos de programação utilizados e o texto a seguir explica resumidamente a função de cada um deles:

1) Estrutura “While” – define os limites do diagrama de estados em questão. O código em seu interior será ciclicamente executado até que ocorra a condição de parada.

2) Constante de tipo “enum” – define o estado inicial para esse diagrama.

3) Condição de parada – define o estado final para esse diagrama de estados, após o qual a execução deverá ser terminada. Neste exemplo todas as atividades como fechamento de arquivos, término de acesso a bancos de dados, etc. deverão ser realizadas no “Estado 4”.

4) Estrutura “Case” – define os limites de cada estado. Dentro de suas fronteiras está o código a ser executado para um determinado estado. Em sua essência, uma estrutura Case aceita valores numéricos, mas através da definição de tipo “enum” podemos traduzir essa informação numérica nos nomes dos possíveis estados. Só é possível visualizar um estado de cada vez, neste exemplo vemos o código correspondente ao “Estado 1”. Usando as flechinhas ao lado do nome do estado atual podemos chavear para os trechos de código correspondentes aos demais estados.

5) Contador – define o tempo mínimo de execução do estado, para garantir que quando uma classe estiver em um estado onde poucas atividades são executadas, ela não consuma recursos do sistema desnecessariamente em verificações de eventos.

6) As atividades em si a serem executadas nesse estado.

7) Verificação de evento de transição – dependendo da ocorrência ou não de evento, define se a classe de permanecer no estado atual ou mudar para algum outro.

8) Registro de deslocamento – recebe um valor da verificação de evento no final de um ciclo de execução e o disponibiliza para o próximo ciclo. Neste caso ele recebe o próximo estado a ser executado (eventualmente o mesmo atual) e esse valor é utilizado para controlar a estrutura “Case” e definir qual trecho de código deverá ser executado a seguir.

9) Estrutura “Se” –uma estrutura auxiliar opcional, usada em estados cujas atividades devem ser executadas apenas uma vez e depois a classe permanece esperando a ocorrência de um evento de transição. Através da comparação entre o último estado executado (valor do registro de deslocamento no ciclo t-2) e o novo estado requerido (valor do registro de deslocamento no ciclo t-1), pode-se evitar a execução de um trecho de código.

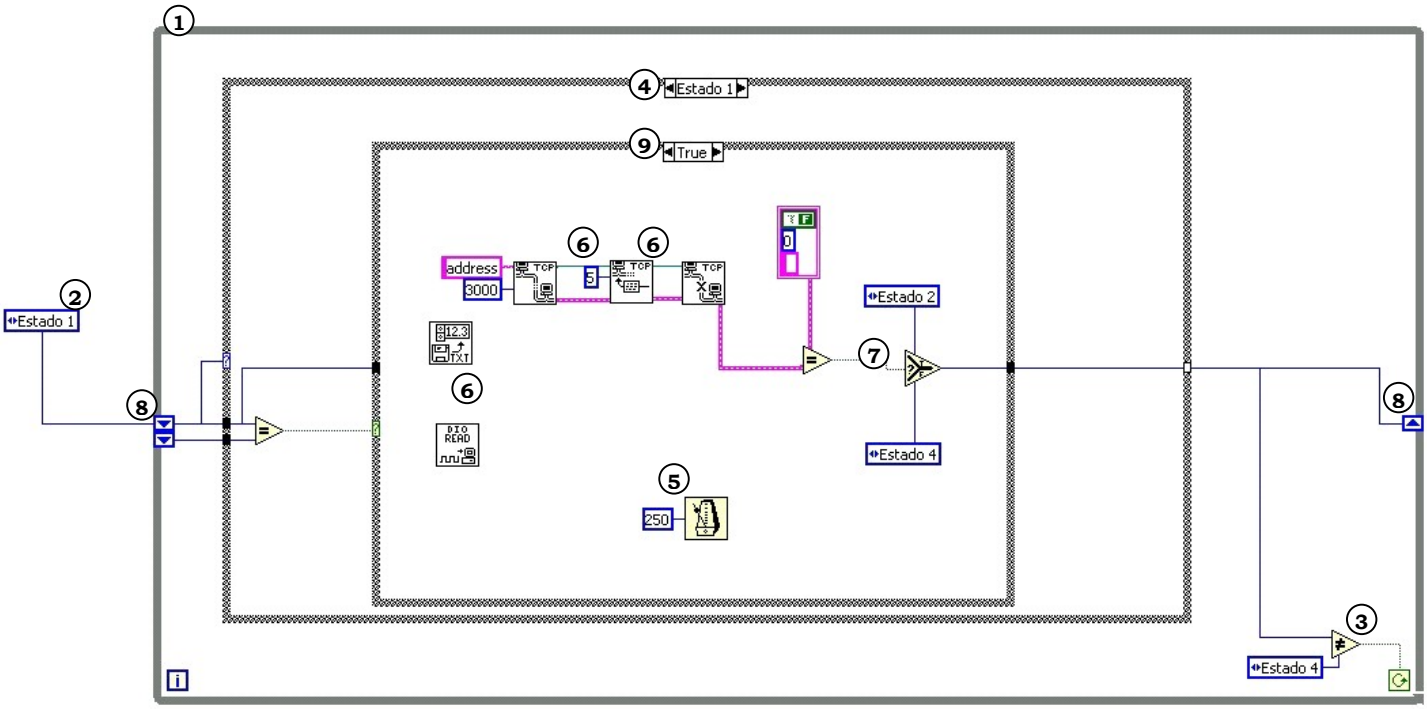


Figura 23: Realização de Diagrama de Estados em LabVIEW

Agora já conhecemos o "léxico" para a tradução de um código LabVIEW no diagrama de classes que o originou. Continuaremos agora com a descrição do modelamento do nosso sistema. Uma descrição e análise dos principais trechos de código será feita no Capítulo 6.

5.5 Modelo Físico – Diagramas de Componentes e de Implantação

Há dois tipos de diagramas que modelam a implementação física do sistema:

- o diagrama de componentes mostra a organização entre pacotes de código (arquivos de código fonte, bibliotecas, tabelas de banco de dados, etc.). A relação mais usada é a dependência, mostrando como um arquivo de código fonte depende de um outro que ele inclui ou como um executável depende de uma biblioteca (junção em tempo de compilação, conexão ou execução), mas também podem ser utilizadas relações de generalização, associação e realização. Os elementos de modelagem são os componentes e as interfaces. A UML reconhece cinco estereótipos de componentes: Executável; Biblioteca; Tabela; Documento e Arquivo.

- o diagrama de implantação mostra a estrutura da aplicação e a configuração dos nós de processamento em tempo de execução. Utilizado principalmente para sistemas embarcados, distribuídos ou cliente/servidor, pois permite avaliar conseqüências da distribuição e alocação de recursos.

Na Figura 24 podemos ver o diagrama de componentes completo para o nosso sistema simulador e na Figura 25 o diagrama de implantação com os principais blocos de software que são ativados em tempo de execução.

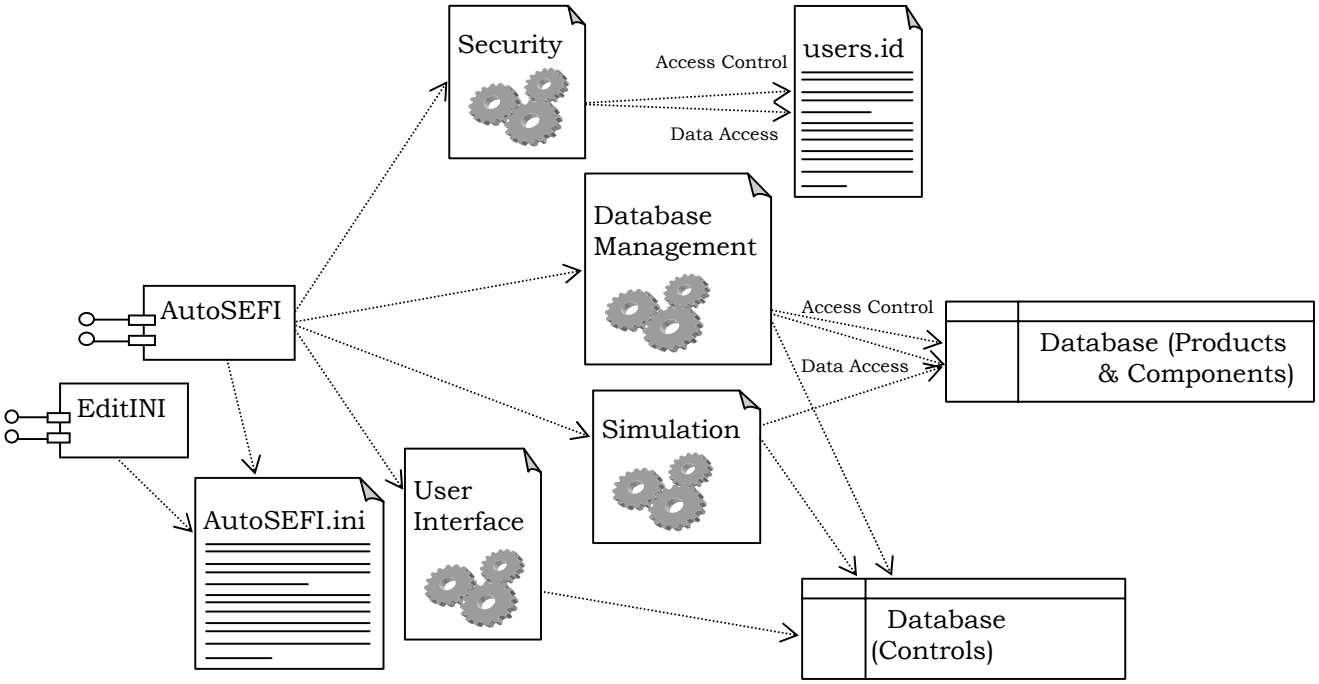


Figura 24: Diagrama de Componentes

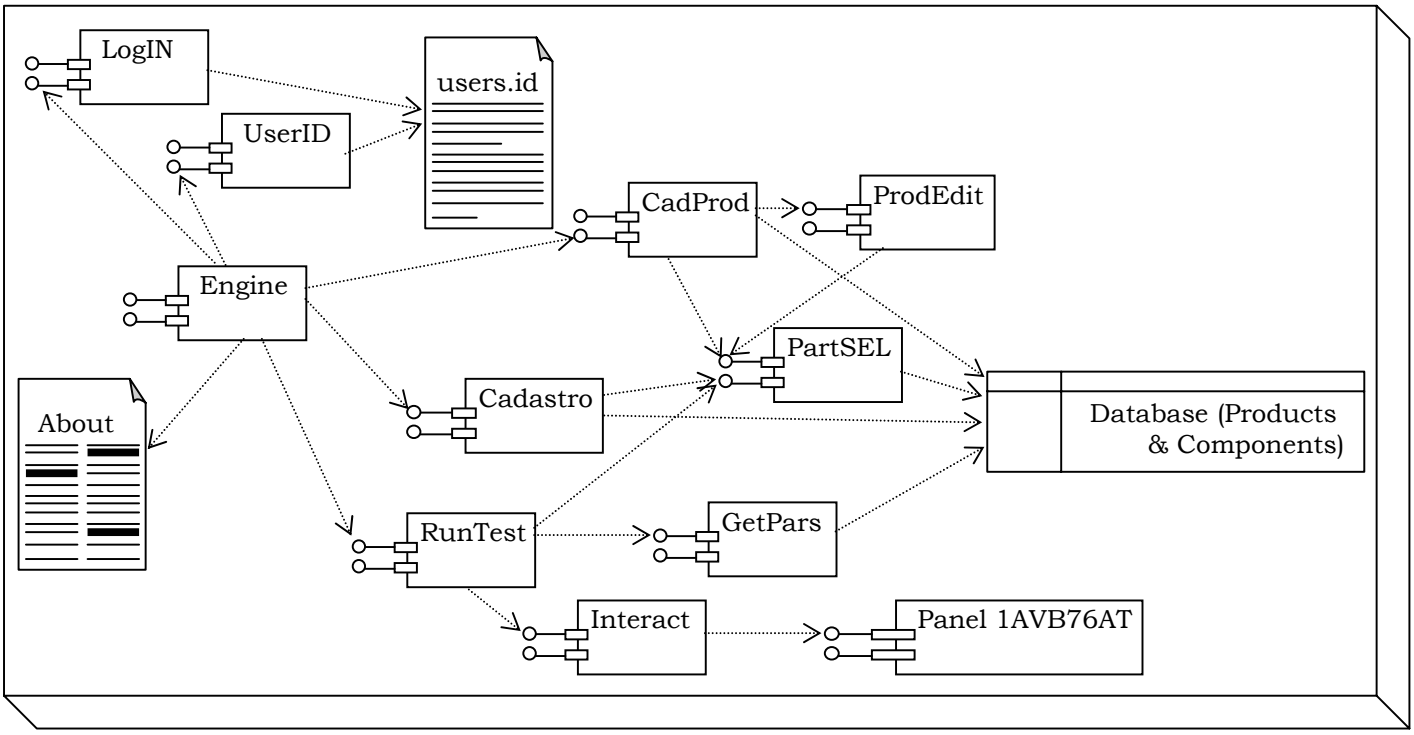


Figura 25: Diagrama de Implantação

5.6 Resumo da Aplicação da UML com LabVIEW 4.1

Antes de tudo, precisamos esclarecer que a programação em LabView é feita em duas vistas complementares: a janela de diagrama (código a ser executado) e a janela de painel frontal (interface gráfica com o usuário). Para cada "controle" ou "mostrador" posicionado no painel frontal é gerado um elemento de conexão na janela de diagrama para que ele possa ser utilizado (conectado) no resto do código.

Cada arquivo ou sub-rotina é chamado de VI (virtual instrument), VIs podem ser gravados como arquivos individuais com a extensão ".vi" ou agrupados em arquivos de bibliotecas com extensão ".llb". Esses arquivos funcionam como se fossem sub-diretórios, porém seu conteúdo só pode ser acessado pelo LabView. Com essas informações básicas, podemos seguir em frente.

O conjunto de todos os **diagramas de *Use Cases*** representa toda a funcionalidade a ser implementada no sistema e indica o que está disponível para quais atores. A descrição detalhada para cada *use case* não é padronizada na UML. Em este trabalho definimos um formulário a ser utilizado que pode ser visto no Apêndice A.

Estudando-se os *use cases* deve-se identificar toda referência a possíveis entidades que contenham informações geradas pelo ou necessárias para o sistema, bem como entidades que executem tarefas. Em seguida, devem ser identificadas relações de dependência entre essas entidades por interpretação do texto dos *use cases*. As entidades identificadas são chamadas de Classes e sua definição bem como a definição das relações de interdependência entre elas é a etapa mais crítica de todo desenvolvimento de sistema orientado a objeto. Essas definições são documentadas nos **diagramas de Classes**.

Caso pudéssemos haver utilizado a geração automática de código com a ferramenta Rational Rose, por exemplo para a linguagem C, as definições contidas nesses diagramas seriam automaticamente convertidas em declarações de classes e armazenadas em bibliotecas. Como o LabVIEW 4.1 não suportava declaração formal de classes, tivemos que criar manualmente para cada classe os chamados "controles" em LabVIEW, cada qual com suas respectivas propriedades conforme os diagramas de classes. Algumas das classes criadas foram armazenadas individualmente como arquivos ".ctl" (classes sem métodos, equivalentes a definições de tipo) e outras como arquivos ".vi" dentro de bibliotecas ".llb". Todos os três são formatos de arquivos do LabVIEW.

Como não há declaração formal de classes, também não há instanciamento. Em C, para instanciar uma *centralina* 1AVB76AT na classe ECU, teríamos algo como:

```
#include "NossaBiblioteca.h"
ECU centr1avb76at ;
```

Já em LabVIEW, precisamos sempre de um controle/mostrador posicionado no painel frontal para poder utilizá-lo no código. A abordagem mais direta seria considerar como "declaração de classe" um conector/mostrador copiado da nossa biblioteca e o "instanciamento" aconteceria quando esse conector/mostrador recebia algum valor concreto. A desvantagem óbvia é que no caso de se atualizar a definição de classe teríamos que buscar por todo o código aonde cada uma delas foi usada.

Uma oura abordagem que usamos em alguns casos foi criar um VI com diversas "declarações de classe". Depois fazíamos chamadas a esse VI solicitando como resposta uma das "declarações de classe" disponíveis, assim atualizações eram feitas apenas em um lugar.

Deixando um pouco o tema classes, vamos continuar com o modelo dinâmico. Estudando-se novamente os *use cases* devem-se identificar os processos a serem realizados. Dependendo do tipo de processo, pode ser mais adequado um modelamento através de **diagramas de Objetos** (para processos sequenciais) ou de **diagramas de Estados** (para processos orientados a eventos).

A tradução de diagramas de objetos para código LabVIEW é feita por simples interpretação visual. O diagrama de objetos já é representado por objetos trocando mensagens em uma seqüência linear de tempo, exatamente como a programação visual por LabVIEW!

A tradução de diagramas de estado já foi explicada em detalhe no Item 5.4.

Passando para o modelo fisico, temos o **diagrama de Componentes** que representa a organização do sistema em pacotes de código. A organização do nosso sistema será explicada em detalhe logo no início do Capítulo 6 a seguir. É recomendável acompanhar a leitura desses parágrafos olhando o diagrama representado na Figura 24: Diagrama de Componentes.

Por fim temos o **diagrama de Implantação**, que mostra os blocos de software em tempo de execução. A maioria dos blocos de software que podem ser vistos na Figura 25: Diagrama de Implantação serão explicados em detalhe no Item 6.2. É recomendável acompanhar a leitura do Item 6.2 identificando cada bloco de software nessa figura e olhando o código-fonte do nosso sistema.

Capítulo 6 – Realização do Software

A programação em LabView é feita em duas vistas complementares: a janela de diagrama (código a ser executado) e a janela de painel frontal (interface gráfica com o usuário).

Cada arquivo ou sub-rotina é chamado de VI (virtual instrument), VIs podem ser gravados como arquivos individuais com a extensão ".vi" ou agrupados em arquivos de bibliotecas com extensão ".llb". Esses arquivos funcionam como se fossem sub-diretórios, porém seu conteúdo só pode ser acessado pelo LabView.

Durante o desenvolvimento, o sistema simulador era composto dos seguintes arquivos em um diretório-base: autosefi.ini (basicamente um arquivo texto, contendo parâmetros de inicialização do sistema), Corefiles.llb, DataBaseManagement.llb, Security.llb, Simulation.llb, UserInterface.llb e Users.id (arquivo criptografado em formato proprietário que desenvolvemos, contendo toda a base de dados dos usuários cadastrados no sistema). Para a versão de distribuição, o arquivo CoreFiles.llb foi compilado gerando o AutoSEFI.exe e as llbs foram salvas com a opção 'Application Distribution', ou seja, os VIs não contém os diagramas. Complementando a organização do sistema, neste diretório-base existem os seguintes sub-diretórios: Controls e DataBases. Uma listagem dos VIs em cada biblioteca .llb pode ser encontrada no Apêndice B.

O sub-diretório "DataBases" contém as bases de dados que nada mais são do que arquivos com as extensões .cmp e .pdt (formato proprietário desenvolvido no escopo deste projeto) contendo as informações de configuração dos Componentes e Produtos respectivamente, organizados em estruturas de sub-diretórios conforme o esquema da base de dados "Example" na página seguinte.

O sub-diretório "Controls" contém arquivos de extensão ".ctl" (formato proprietário do LabView) com os seguintes controles usados nos diversos painéis frontais do sistema simulador: actuatori.ctl, actuatorii.ctl, configinfo.ctl, ecu.ctl, ECU1.ctl, idlespeed.ctl, mainstates.ctl, sensori.ctl, sensorii.ctl, type.ctl.

De forma análoga ao modelamento, iremos descrever nosso sistema começando pelo ponto de vista do usuário.

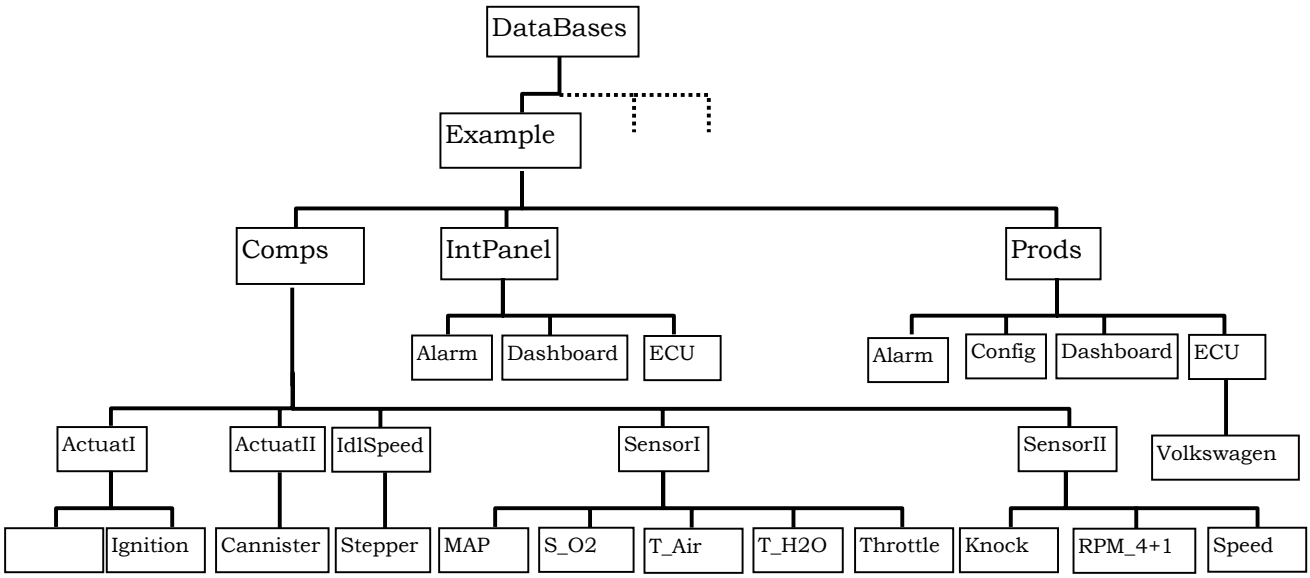


Figura 26: Estrutura de sub-diretórios para as bases de dados

6.1 Descrição da forma de utilização do Sistema

Instalação

Infelizmente não foi gerada uma versão de distribuição utilizando a interface de instalação padrão "Windows". Para transportarmos o sistema para uma outra máquina, utilizamos basicamente um arquivo compactado (.zip) com os arquivos AutoSEFI.exe e as outras llbs mencionadas acima. As estruturas de diretórios para a Base de Dados e os Controles ainda precisavam ser criadas manualmente dentro do diretório escolhido para descompactar os arquivos do sistema.

No Capítulo 9 mencionaremos uma maior facilidade de instalação do sistema simulador como requisito para futuras expansões do sistema.

Controle de Acesso ao Sistema

O Gerenciador de Usuários é o módulo do sistema simulador responsável pelo controle das informações cadastrais dos usuários autorizados a utilizá-lo. Este módulo mantém uma base de dados criptografada com um registro para o administrador do sistema e um para cada usuário. Cada registro possui campos para armazenar as seguintes informações: nome, login, senha e privilégios.

Para executar este módulo, devem ser fornecidos o login e a senha do administrador do sistema. Para a primeira utilização, devem ser utilizados o login e senha fornecidos com o manual do produto, e o administrador pode então substituí-los por outros mais a seu gosto para garantir a segurança do sistema. Em seguida o administrador deve cadastrar ao menos um usuário para que este possa utilizar os outros recursos do sistema, pois o administrador não tem privilégios para tal.

A interface do Gerenciador de Usuários está mostrada na Figura 27. A caixa da esquerda mostra os usuários atualmente registrados no sistema, sendo que o administrador é o que possui o símbolo '∅' ao seu lado. O formulário de registro à direita mostra os atributos do usuário selecionado. O administrador pode modificar, adicionar e remover usuários, desde que não haja repetição de logins. Os privilégios de utilização do sistema seguem o modelamento já discutido, e são definidos com base no(s) ator(es) escolhido(s) para cada usuário dentre:

- Técnico: só é capaz de executar testes em produtos;
- Engenheiro: além de executar testes, também é capaz de configurar produtos e componentes;
- Gerente: só é capaz de analisar relatórios de testes armazenados na Base de Dados.

Figura 27: Gerenciador de Usuários

Identificação do Usuário

Ao executar o arquivo AutoSEFI.exe o usuário é apresentado à tela mostrada na Figura 28 abaixo, onde deve informar seu login e senha de acesso.

Figura 28: Login

Engine

Após informar seu login, o usuário é apresentado à tela principal do sistema onde pode escolher que tipo de atividade deseja executar com o sistema. Apenas as atividades para as quais seu perfil está autorizado podem ser selecionadas.

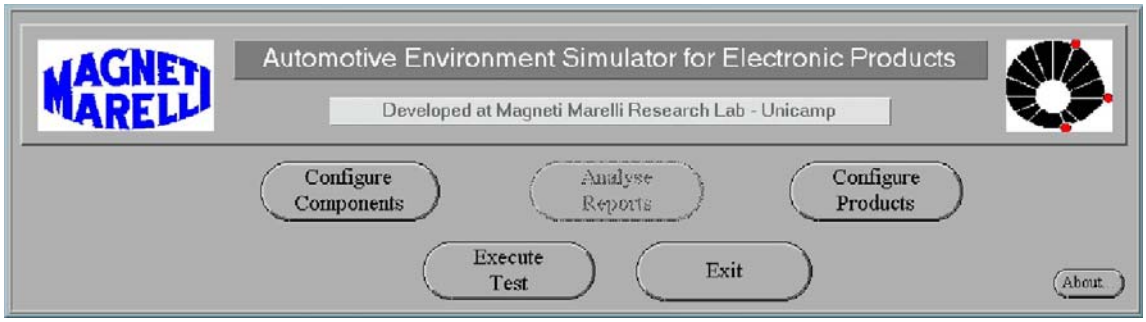


Figura 29: *Engine*

About

O botão de "About" mostra uma tela com informações sobre a versão atual do simulador.

Figura 30: *About*

Gerenciador de Componentes

Para executar uma simulação, o Técnico deve selecionar um Produto dentre os disponíveis na Base de Dados. Isso pressupõe que um Engenheiro já cadastrou ao menos um Produto para ser testado, e ainda, que os Componentes e Testes associados a esse produto também já foram cadastrados e estão disponíveis nessa mesma Base de Dados. O Gerenciador de Componentes é o módulo responsável pelo controle dos Componentes cadastrados na Base de Dados e é através dele que um Engenheiro pode adicionar, modificar ou remover componentes. Sua interface inicial pode ser vista na Figura 31, com o detalhe que os campos de atributos estariam todos com valores nulos.

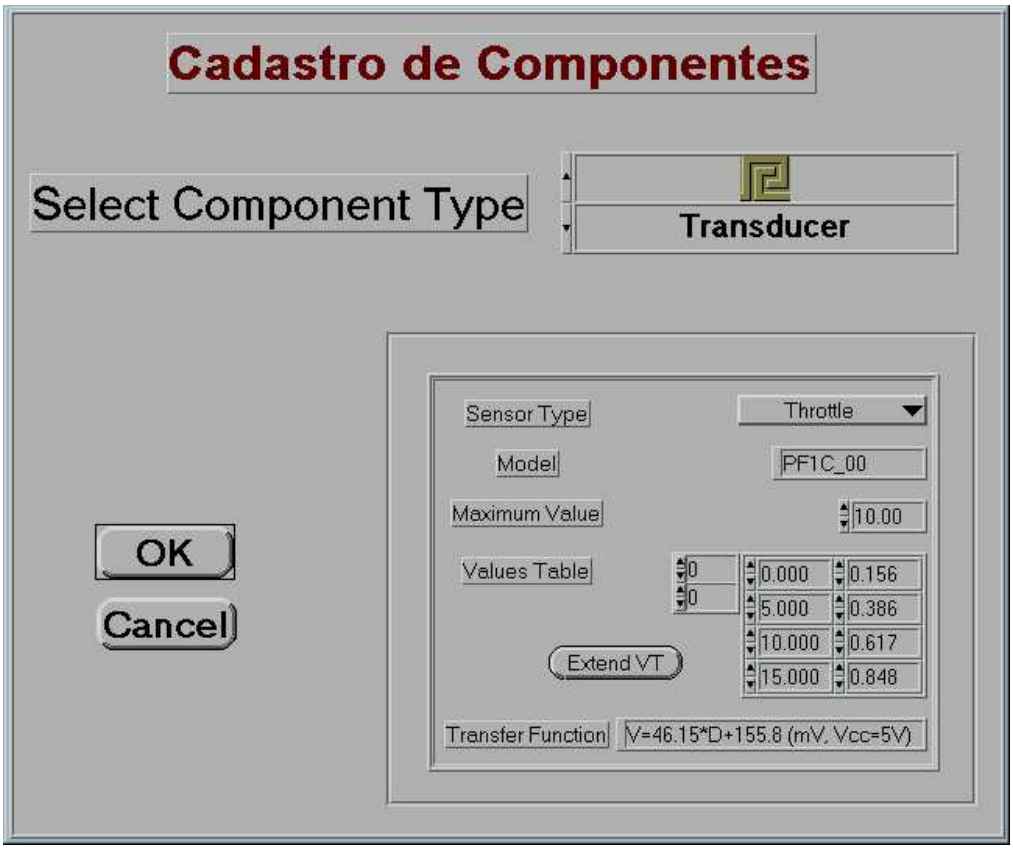


Figura 31: Cadastro de Componentes

O primeiro passo é selecionar o Tipo de Componente dentre os cinco disponíveis no sistema: Transdutores, Sensores de Freqüência, Atuadores de Potência, Atuadores de Freqüência e Atuadores de Marcha Lenta. A caixa abaixo do seletor mostra sempre os atributos pertinentes ao Tipo de Componente selecionado. Ao confirmar a seleção, é apresentada a janela de Seleção de Peça, onde são oferecidas as opções de adicionar, modificar ou remover um Componente do tipo previamente selecionado.

Esta janela, mostrada na Figura 32 na página seguinte, é utilizada para navegação na Base de Dados, somente entre componentes do tipo anteriormente selecionado. A lista da esquerda navega entre grupos de componentes, através de duplo-clique no nome do grupo para descer um nível ou em '..' para voltar para o grupo anterior (nível acima). A lista da direita mostra os componentes disponíveis para serem selecionados dentro do grupo selecionado à esquerda.

Importante: Os sub-grupos não são mostrados na lista da direita. O usuário deve selecionar um grupo para poder ver seus sub-grupos na lista da esquerda.

Pressionando "New", o usuário é solicitado a informar um nome para o novo componente ou grupo a ser criado, como pode ser visto na Figura 33.

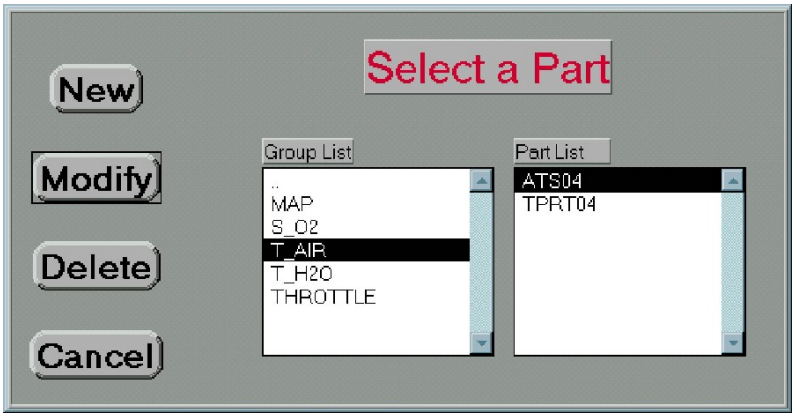


Figura 32: Seleção de Peça

Figura 33: Seleção de Nome

Com um duplo-clique em um componente ou pressionando "Modify" (ação *default* também associada com a tecla "Enter") o sistema retorna à tela da Figura 31, onde são mostrados os valores atuais dos atributos do Componente selecionado e o Engenheiro pode modificar os campos à sua conveniência. O botão de "Ok" foi substituído agora por um botão "Modify"". No caso de um novo componente, a tela é a mesma com a diferença que todos os campos estão preenchidos com valores nulos.

Segue-se um resumo dos atributos disponíveis para cada Tipo de Componente. A listagem está em inglês, pois faz parte da documentação original do sistema.

- *Transducer:*

(ring) Sensor Type: Indicates the kind of transducer for this component. It is used by the system to organize the Database.

(string) Model: The name to serve as a reference for the component in the Database.

(dbl) Maximum Value: The maximum value (in Volts) this signal can assume.

(array) Values Table: A look-up table with the conversion of physical to voltage unities for this component. The first column represents the physical values, and the second the voltages.
(string) Transfer Function: The transfer function that fits better the points in the Values Table. This information is not being used in the current version.

- *Frequency Actuator:*

(ring) Sensor Type: Indicates the kind of frequency actuator for this component. It is used by the system to organize the Database.
(string) Model: The name to serve as a reference for the component in the Database.
(dbl) Maximum Value: The maximum value (in Volts) this signal can assume.
(string) Amplitude Function: The transfer function relating the amplitude with the frequency of the signal. This information is not being used in the current version.

- *Power Actuator:*

(ring) Actuator Type: Indicates the kind of power actuator for this component. It is used by the system to organize the Database.
(string) Model: The name to serve as a reference for the component in the Database.
(dbl) Opening/Rise Time (usec): Time spent to open the injector or time to charge the ignition coil.
(dbl) Closing/Spark Time (usec): Time spent to close the injector or duration time of the spark produced by the ignition coil.

- *Frequency Actuator:*

(ring) Actuator Type: Indicates the kind of frequency actuator for this component. It is used by the system to organize the Database.
(string) Model: The name to serve as a reference for the component in the Database.
(dbl) Operating Frequency: The operating frequency for the duty cycle of the component. The first column represents the physical values, and the second the duty cycles in percentages.
(array) Values Table: A look-up table with the conversion of physical to duty cycle unities for this component.

- *Idle Speed Actuator:*

(ring) Control Type: Indicates the kind of idle speed control for this component. It is used by the system to organize the Database.
(string) Model: The name to serve as a reference for the component in the Database.
(dbl) Number of Steps/Operating Frequency: The maximum number of steps for Stepper Motors or the operating frequency of the duty cycle for DC Motors.
(array) Forward Step Phase Sequence: A table containing the valid ordered binary numbers representing the phase signals of a stepper motor.

Os tipos Transdutor e Atuador de Freqüência possuem ainda o método '*Extend VT*'. Este método é uma poderosa ferramenta para converter, ajustar ou adicionar dados ao conjunto de pontos fornecido na tabela de valores original do Componente (fornecida pelo respectivo fabricante), e é acionado com o pressionamento do botão homônimo. Sua interface pode ser vista na Figura 34, na página seguinte.

Este método executa uma interpolação *spline* cúbica nos valores presentes na matriz 'Input VT', adicionando os valores necessários para que o espaçamento entre valores seja igual ao valor selecionado no controle numérico 'Interval between Samples', que representará sua precisão na simulação das grandezas físicas associadas (primeira coluna). Adicionalmente, a melhor aproximação exponencial juntamente com seu erro quadrático médio em relação aos valores iniciais também é mostrada, permitindo que o Engenheiro substitua seus valores originais por esta função de transferência se assim o desejar.

Finalmente, muitos transdutores como NTCs, PTCs, etc. representam variações em uma grandeza física através de variações em sua resistência. Em dispositivos de eletrônica automotiva, a circuitaria de entrada desses transdutores usualmente consiste de um resistor de *pull-up* ligado a uma tensão Vcc, e o divisor resistivo entre o transdutor e este resistor fornece um nível de tensão que é interpretado pelo dispositivo. Como nossas placas DAQ são adequadas para a geração de níveis de tensão e não resistências, o Engenheiro pode fornecer os valores de resistência disponíveis com o transdutor e os valores do resistor de *pull-up* e da tensão Vcc para que o método '*Extend VT*' calcule os valores apropriados.

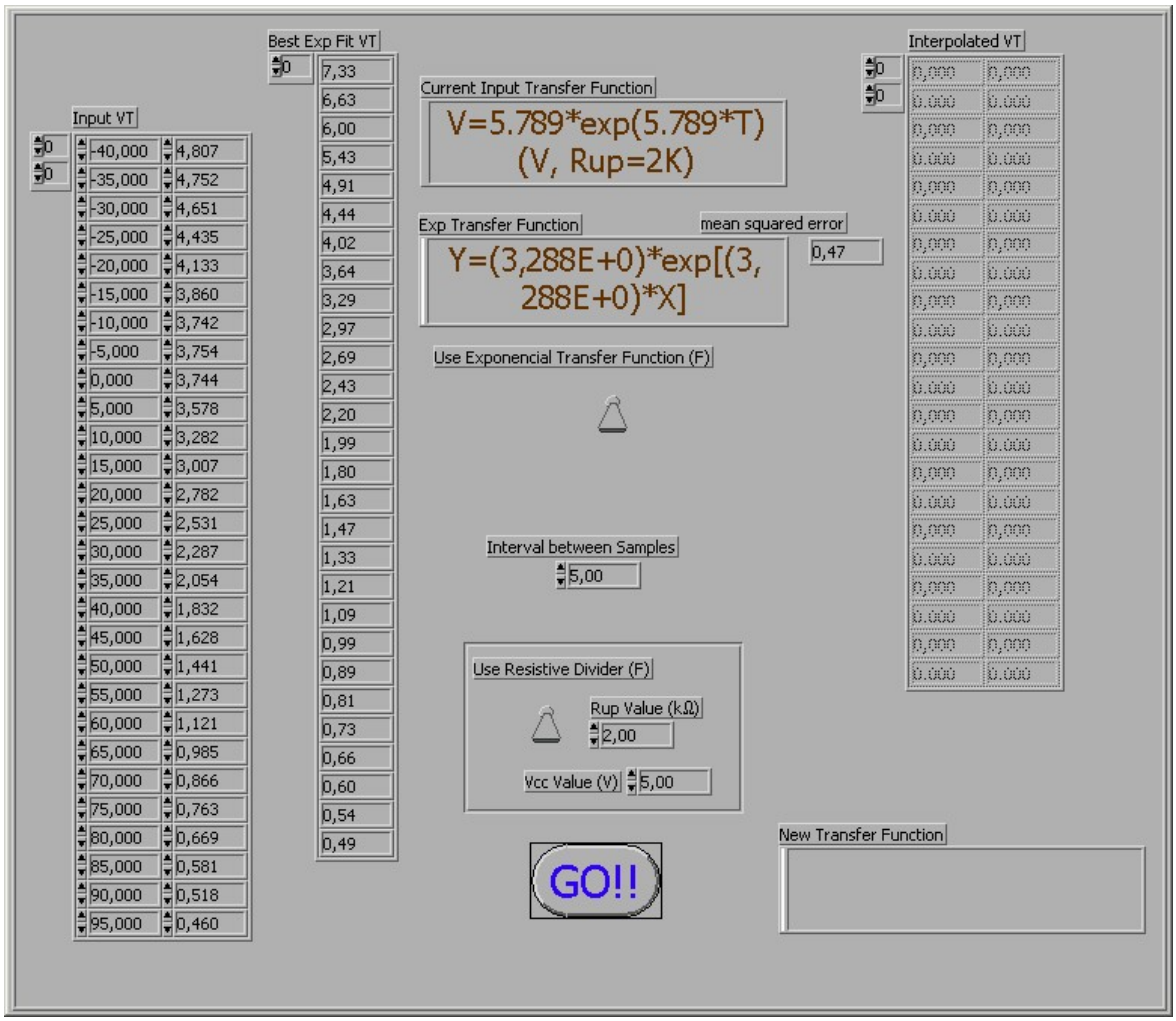


Figura 34: Ferramenta *Extend VT*

Gerenciador de Produtos

Depois que um Engenheiro já cadastrou Componentes na Base de Dados, ele pode configurar um produto. O Gerenciador de Produtos é o módulo através do qual um Engenheiro pode adicionar, modificar ou remover produtos da Base de Dados.

O primeiro passo é selecionar o tipo de produto. Nesta versão atual o nosso sistema como um todo suporta apenas *centralinas*, mas a interface inicial de seleção que pode ser vista na Figura 35 abaixo já mostra os atributos e métodos associados a outros tipos de produtos à medida que o seletor de produto é acionado.

Figura 35: Seleção de tipo de Produto

Da mesma forma que com o Gerenciador de Componentes, o sistema apresenta a mesma janela de Seleção de Peça mostrada na Figura 32. A funcionalidade é análoga à descrita anteriormente, com a diferença que os campos na Figura 35, agora com o botão "Modify" substituindo o botão "Ok", não serão preenchidos manualmente. O Gerenciador de Produtos tem uma ferramenta própria para permitir uma visão melhor dos componentes que estão sendo associados ao produto em questão. Esta ferramenta é ativada ao se pressionar o botão "Modify" e sua janela, com os componentes associados ao grupo "Transdutores", pode ser vista na. Figura 36.

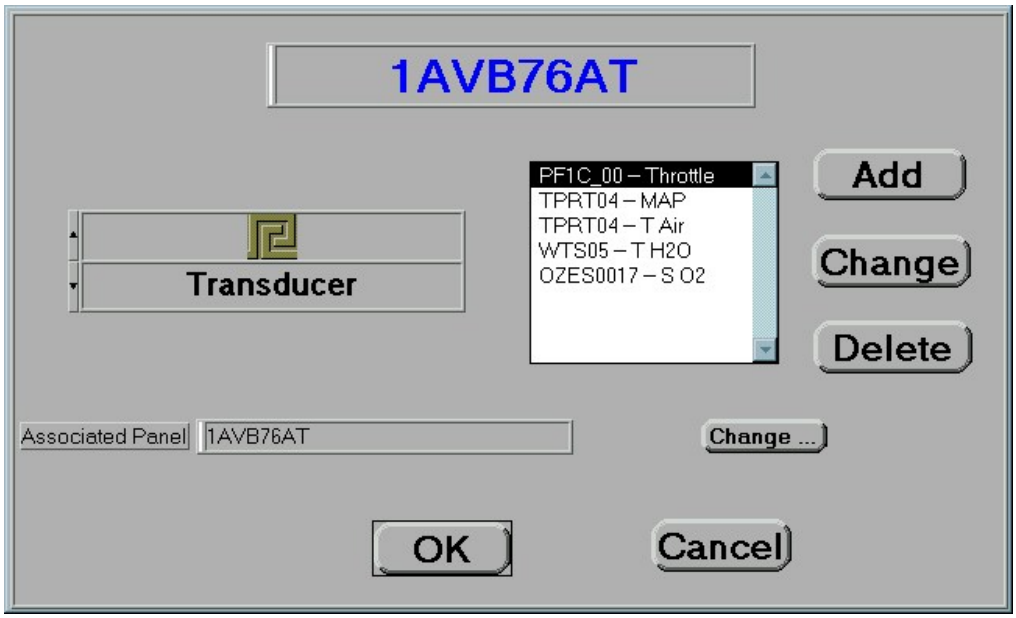


Figura 36: Configuração de Produto

O modelo (nome) do produto na parte superior central da janela. O painel de controle selecionado é mostrado no campo mais inferior e pode ser substituído pressionando-se o botão "Change...". Mais uma vez a seleção será feita utilizando-se a janela de seleção de peça já mostrada na Figura 32.

Durante a edição de um produto, o Engenheiro associa os componentes de acordo com a subdivisão em nove tipos adotada. A lista central mostra todos os componentes – do tipo selecionado á sua esquerda – que já estão associados com este produto. Espaços vazios poderão aparecer entre os nomes dos componentes durante uma sessão onde diversos componentes sejam adicionados e/ou substituídos e devem ser desconsiderados. O usuário pode adicionar e/ou substituir componentes da lista pressionando o botão correspondente, sempre através da já mencionada janela de seleção de peça. Pressionando o botão "Delete" ativa-se uma janela de confirmação antes da remoção definitiva do componente selecionado.

O sistema vai alocando os recursos disponíveis em seu módulo de condicionamento de sinais à medida que o Engenheiro vai associando esses componentes aos pinos do Produto. Essa tabela de alocação dá origem ao relatório de conexões para o chicote que unirá o Produto ao Módulo de Condicionamento.

Segue-se um resumo dos atributos disponíveis para cada o Tipo de Produto ECU (*centralina*). A listagem está em inglês, pois faz parte da documentação original do sistema.

- ECU:

(string) Model: The name to serve as a reference for the component in the Database.
(array) Transducers: Array of strings containing the relative paths of all transducers associated with this product.
(array) Frequency Sensors: Array of strings containing the relative paths of all frequency sensors associated with this product.
(array) Power Actuators: Array of strings containing the relative paths of all power actuators associated with this product.
(array) Frequency Actuators: Array of strings containing the relative paths of all frequency actuators associated with this product.
(string) Idle Speed Actuator: Contains the relative path of the idle speed actuator associated with this product.
(string) Associated Panel: Contains the relative path of the user interface panel associated with this product.

Executar Teste (Simulação)

O usuário inicia um teste selecionando um produto dentre os disponíveis na Base de Dados, através da mesma janela de seleção de peça diversas vezes mencionada. O sistema então adquire os dados dos componentes associados a este produto e configura as placas DAQ, definindo os limites de tensão, tipo de conexão e taxa de aquisição para cada sinal. Em seguida, o sistema apresenta o painel frontal correspondente. Este painel possui controles correspondendo a todos os sinais de entrada requeridos pelo Produto, mostradores e gráficos correspondendo às informações relevantes sobre a atuação do produto sobre suas cargas (como as formas de onda da corrente nos bicos injetores e bobinas de ignição), e interruptores com a capacidade de simular a desconexão de sensores e cargas. Após a inicialização da interface visual, o painel monitora somente a tensão e bateria e as Entradas e Saídas Digitais, até que o Técnico ligue o interruptor “Key-On”. O sistema passa então a simular um cenário de partida de um veículo e passa a interagir continuamente com o Técnico, simulando estaticamente o ambiente correspondente ao estado dos controles no painel frontal. A simulação é interrompida desligando-se o interruptor de “Key-On” ou pressionando-se o botão “End Test”.

O sistema pode simular a desconexão de todos os transdutores e sensores de freqüência, colocando-se a saída correspondente nas placas DAQ em um estado de alta impedância. A sonda lambda possui um modo adicional de simulação que será descrito no Item 6.2 – Transdutores.

Com o botão “Hold Output” pressionado, o Técnico pode alterar os valores de vários sinais sem alterar o ambiente que está sendo simulado. Estes só serão atualizados simultaneamente quando o botão for liberado.

O botão “Run Script” permite que o Técnico execute um *script* dentre os disponíveis na Base de Dados, e, portanto previamente cadastrado por um Engenheiro. *Scripts* são cenários dinâmicos que representam todos os valores a serem assumidos pelos sinais de entrada da *centralina* e um limite esperado para seus sinais de saída, ao longo do tempo. O Engenheiro pode cadastrar uma seqüência de sinais variando-os a seu gosto ou converter os dados adquiridos de um veículo em um percurso real, simulando esse percurso dinamicamente para a UUT (Unity Under Test).

A estratégia de teste *closed-loop* não pode ser implementada devido à total inexistência de dados sobre o comportamento dinâmico de um veículo. Para futuras implementações desta modalidade de teste, dependeremos fundamentalmente de maior colaboração da Magneti Marelli e integração com a equipe desenvolvedora do sistema.

O painel frontal da *centralina* modelo 1AVB76AT pode ser visto na Figura 37 a seguir.

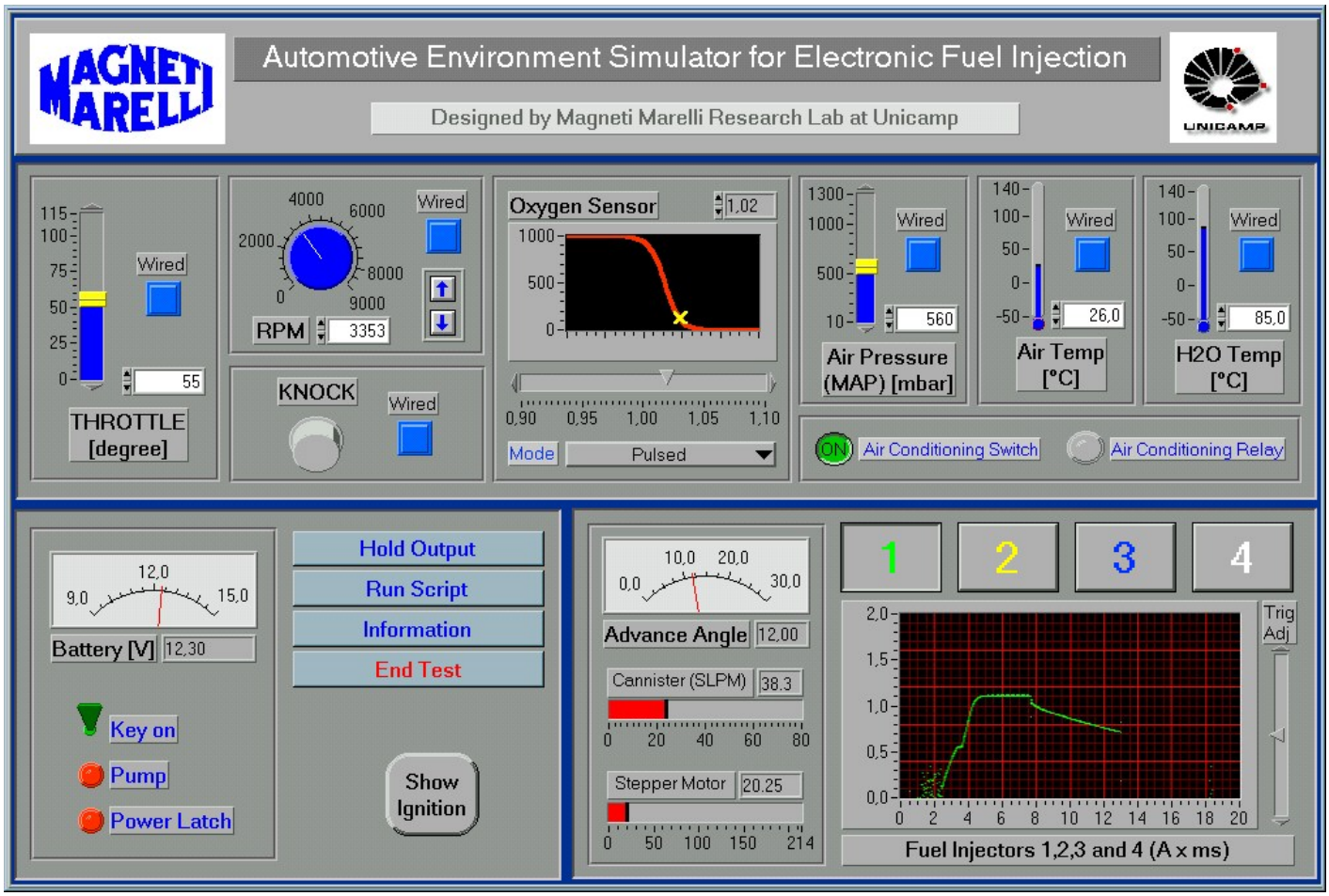


Figura 37: Painel Frontal da *Centralina* 1AVB76AT

Documentação

A documentação dos resultados de um teste é um aspecto muito importante do processo de verificação. A habilidade de gravar e analisar a operação do software de controle da ECU para estímulos externos conhecidos facilita a verificação do software e do sistema. O nível de documentação será dependente dos requisitos do sistema em particular e geralmente recaem em uma de três categorias: inspeção visual, traçados gráficos ou listagens numéricas. Procedimentos de teste que requerem apenas uma confirmação do funcionamento ou não da ECU recaem na primeira categoria. Na segunda categoria, os traçados gráficos são a principal ferramenta de documentação utilizada para demonstrar a conformidade do software da ECU com os requisitos do sistema, pois fornecem uma descrição precisa das operações internas da ECU e do ambiente externo através da sincronização temporal de todos os fluxos de dados. Na terceira categoria as informações gráficas podem ser convertidas em listagens numéricas para análise mais aprofundada dos valores.

O simulador foi desenvolvido com a capacidade de gerar documentação nas três formas descritas acima, no entanto as características de cada documentação não foram especificadas pela Magneti Marelli. Apenas para efeito de validação do nosso sistema, foram feitos alguns testes com a geração de arquivos contendo dados *time-stamped*, cujas rotinas não foram incorporadas na versão atual do software.

6.2 Descrição do código-fonte

Como o LabVIEW é um ambiente de programação visual, ou seja, não há código digitado mas sim "desenhado", descreveremos alguns dos blocos de Software mais importantes com auxílio de *screen shots* das janelas de diagrama de cada VI.

Desde já nos desculpamos pelo fato que as figuras a seguir serão em geral ilegíveis. Infelizmente este trabalho escrito não permite a utilização de recursos como barras de rolagem e por ser um código "desenhado", alguns dos blocos de Software só seriam legíveis em formato impresso com a utilização de papel tamanho A1 ou A0. Para contornar esse problema, disponibilizamos junto com essa dissertação todo o código-fonte gravado em CD. Recomendamos que o leitor acompanhe esse Capítulo visualizando simultaneamente o código-fonte em seu computador.

No nosso primeiro exemplo vamos estudar o código do "Partsel.vi", responsável pela janela de seleção de peça e navegação pela Base de Dados já vista na Figura 32. Ele mostra que a nossa preocupação com a documentação foi grande principalmente nos blocos de Software responsáveis pelo gerenciamento da Base de Dados. A inclusão de comentários em um código LabVIEW (o equivalente ao símbolo ' em *Visual Basic* ou "/* ... */" ou "//" em C) é feita através de caixas de texto explicando cada trecho do código, como pode ser verificado nas Figuras 38 e 39 na página a seguir. Os textos em vermelho incluídos nessas figuras são apenas ampliações dos textos originais para que estes fiquem legíveis, somente por esta razão preferimos mantermo-los em inglês.

Após este primeiro exemplo, nos concentraremos na análise do código vinculado às atividades de simulação que não está tão bem comentado como o relacionado à Base de Dados.

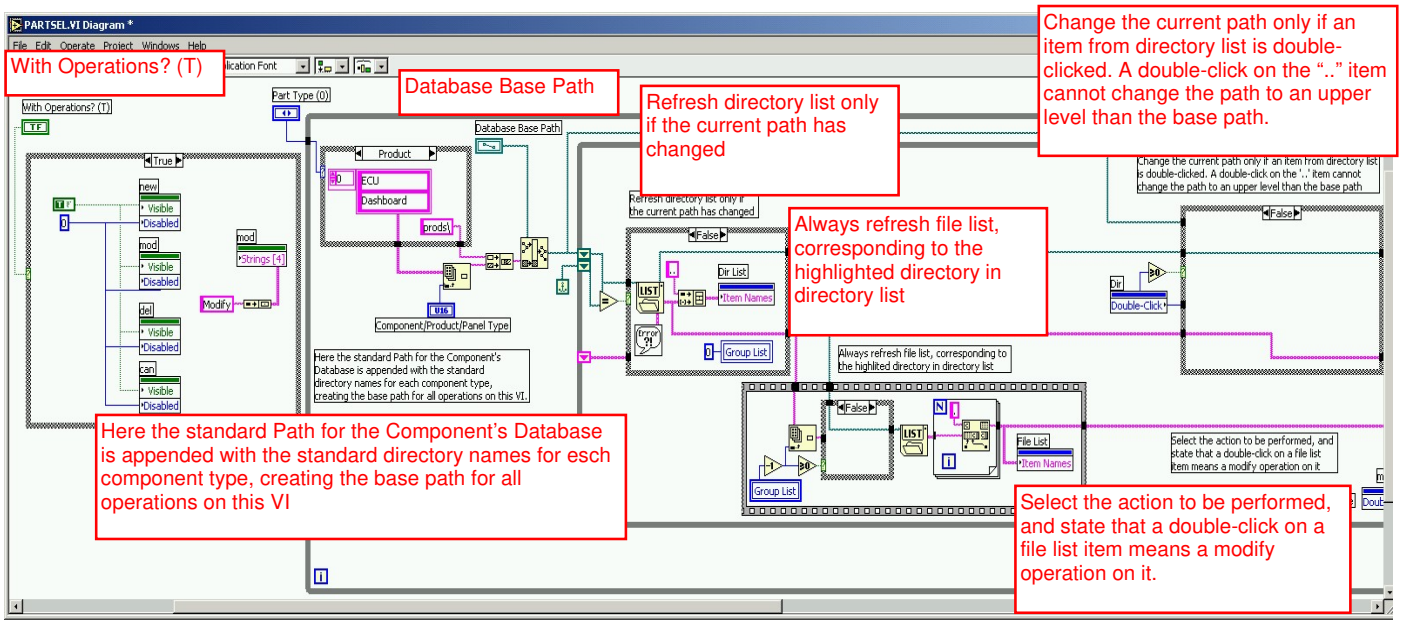


Figura 38: Partsel.vi (lado esquerdo)

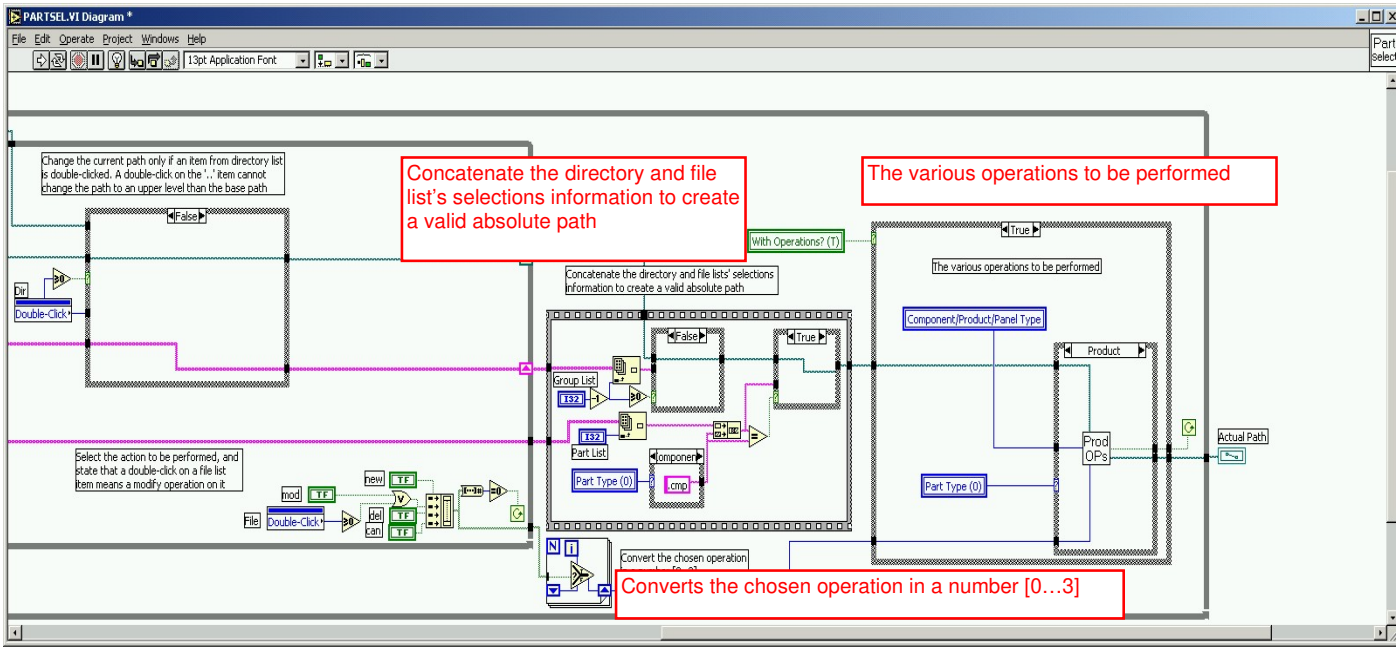


Figura 39: Partsel.vi (lado direito)

Passando agora ao código relacionado com a simulação, comecemos com o Runtest.vi. Esse VI implementa o diagrama de estados mostrado anteriormente na Figura 21. Podemos ver os estados “Initialization” e “Configuration” nas Figuras 40 e 41 a seguir.

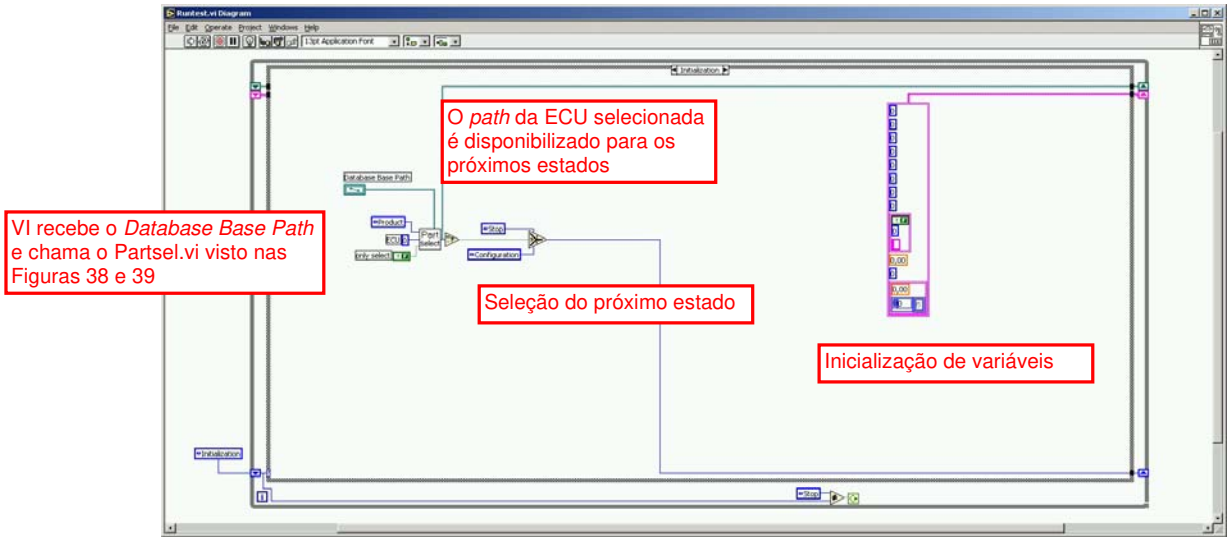


Figura 40: Runtest.vi (Initialization)

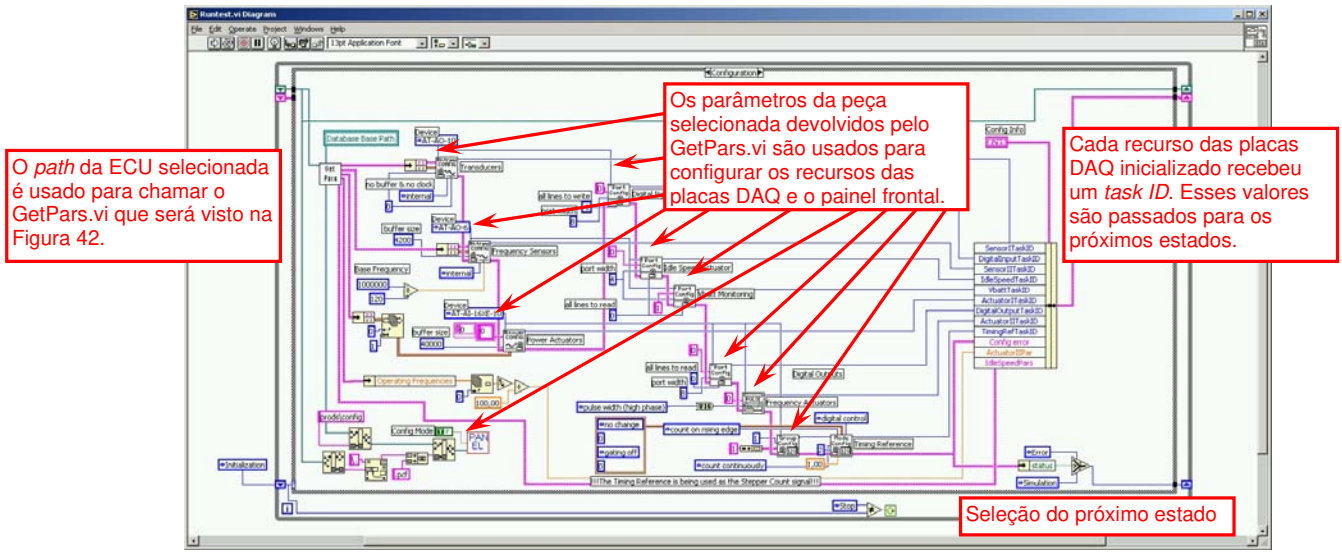


Figura 41: Runtest.vi (Configuration)

Como vimos acima, o estado “Configuration” chama a execução de GetPars.vi enviando como parâmetro o *path* da ECU selecionada. Vejamos abaixo como ele busca na Base de Dados as informações de cada um dos componentes associados à ela.

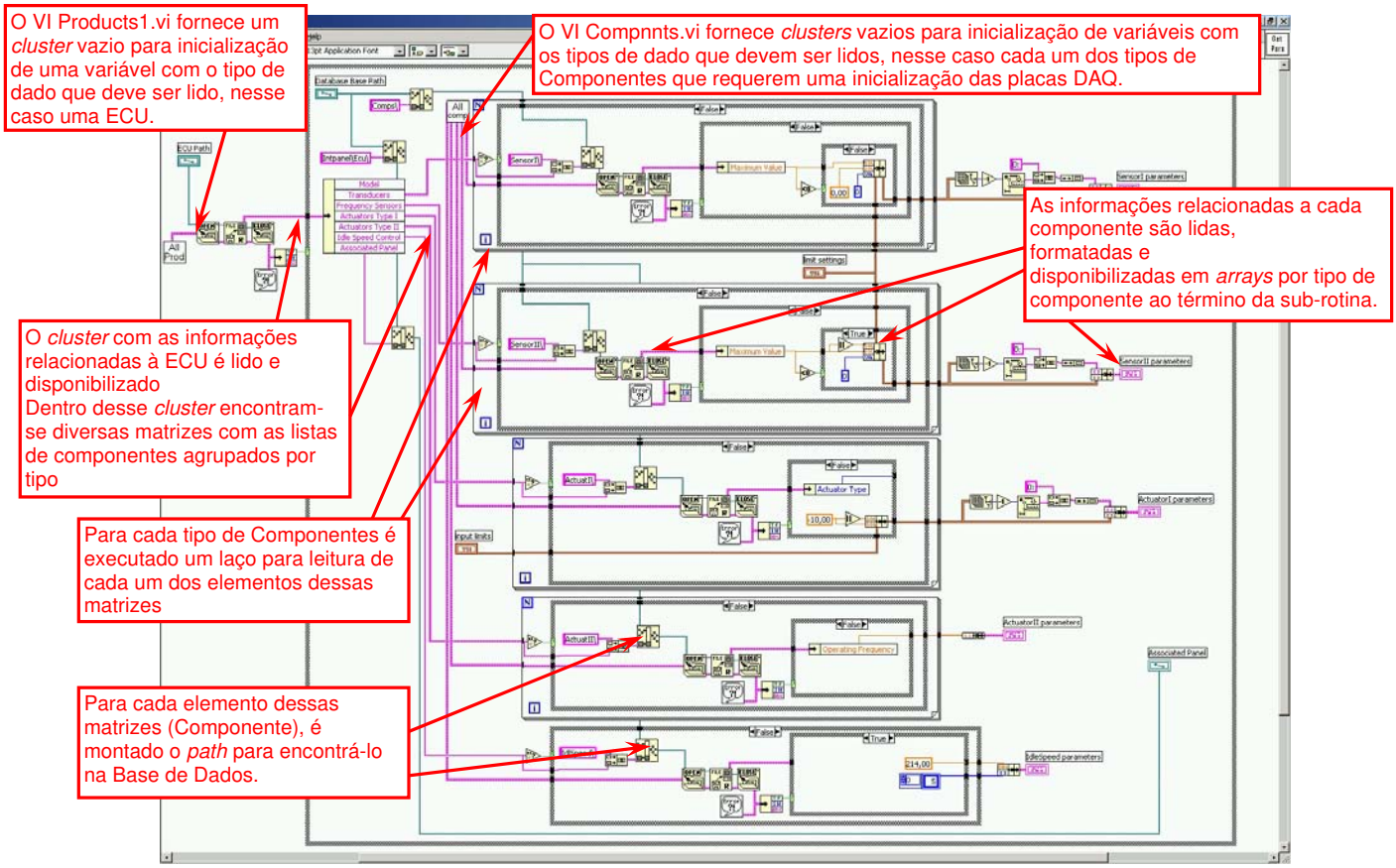


Figura 42: GetPars.vi

Voltando ao Runtest.vi, passemos então á descrição do estado “Simulation”. Como vimos no diagrama de classes da Figura 22, não devemos realizar as tarefas de aquisição, processamento e geração de sinais de forma seqüenciada. Para compatibilizar os recursos disponíveis no nosso sistema com a velocidade de simulação requerida, o processo de simulação foi dividido em subprocessos sendo executados de forma concorrente, cada qual com sua própria temporização. Dessa forma a maioria das atividades de geração e aquisição foram implementadas com temporização em hardware, com a utilização de *buffers* localizados nas próprias placas DAQ sendo a fonte/destino dos dados da simulação. A recuperação e fornecimento dos dados de/para os *buffers* é realizada de forma assíncrona, temporizada em software de acordo com as características dinâmicas e grau de importância de cada tipo de sinais para o sistema. Vejamos como isso funciona na Figura 43 a seguir.

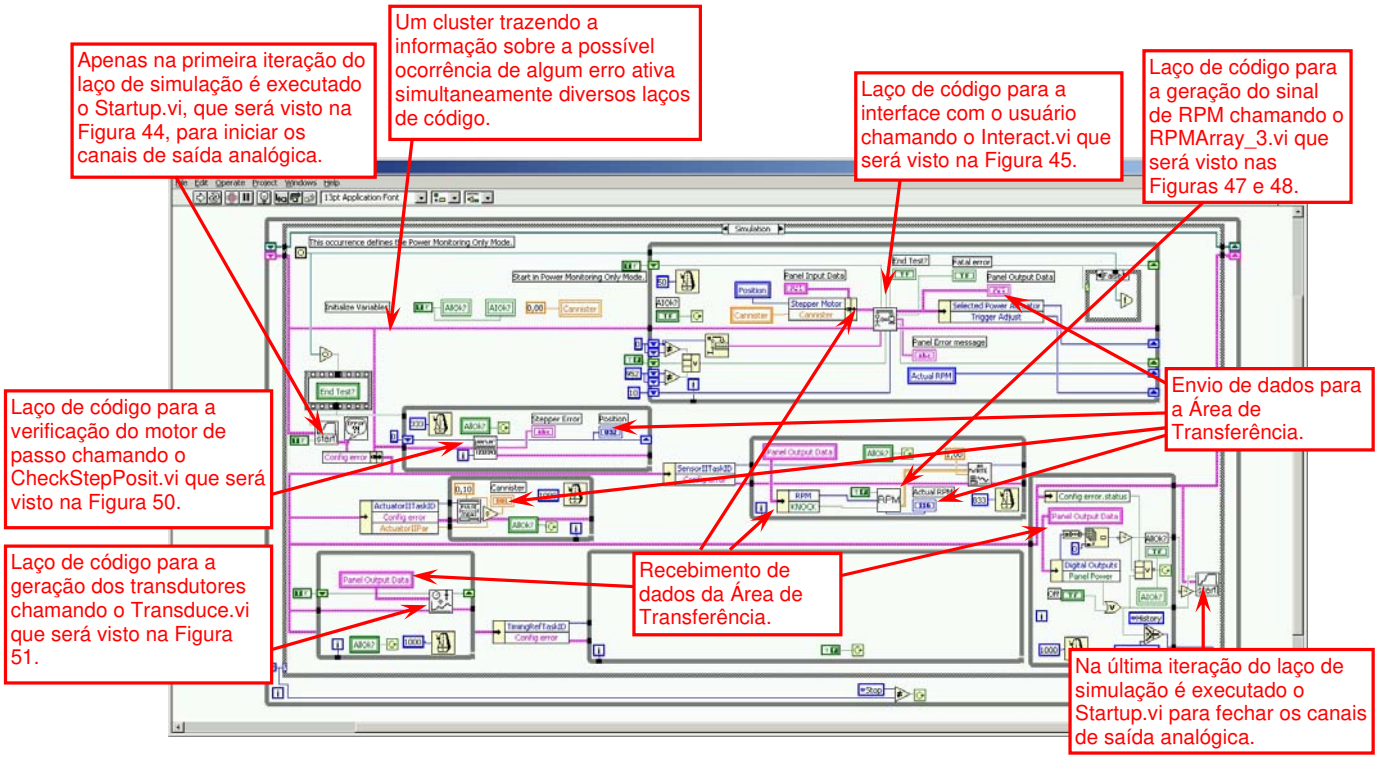


Figura 43: Runtest.vi (Simulation)

No Runtest.vi acima já encontramos a aquisição de sinais dos componentes do tipo:

Atuadores de Freqüência:

As medições necessárias para se avaliar os comandos da ECU para os atuadores de potência são geralmente de freqüência e/ou *duty cycle*. Os contadores/temporizadores disponíveis nas placas DAQ já possuem rotinas prontas para estas funções, bem como para diversas outras de medição de tempos como duração do estado lógico alto ou baixo, etc. e não tivemos maiores problemas com sua implementação.

Os atuadores de freqüência controlados por ECUs são geralmente válvulas eletromagnéticas para controle de vazão de algum fluxo. Os dois principais componentes que se encaixam nessa descrição são as válvulas cannister e EGR. A válvula cannister direciona os vapores do tanque provenientes de evaporamento de combustível para que eles sejam consumidos no motor.

Nossa única limitação era o fato de nossas placas DAQ possuírem somente dois contadores/temporizadores, um dos quais estava sendo utilizado para o motor de passo. Para a versão de bancada desenvolvida e para o modelo de *centralina* utilizado isso não foi empecilho, pois só tínhamos a necessidade de simular uma válvula cannister. Esse problema deve ser solucionado na versão final com a utilização de outras placas DAQ, conforme será discutido nas conclusões desse trabalho.

Como vimos, as funções de cada laço de código estão encapsuladas em sub-rotinas, principalmente para facilitar a visualização. No caso dos atuadores de freqüência pudemos utilizar algumas já existentes no LabVIEW sem necessidade de modificação, mas vamos observar como funciona cada uma das demais começando pelo Startup.vi.

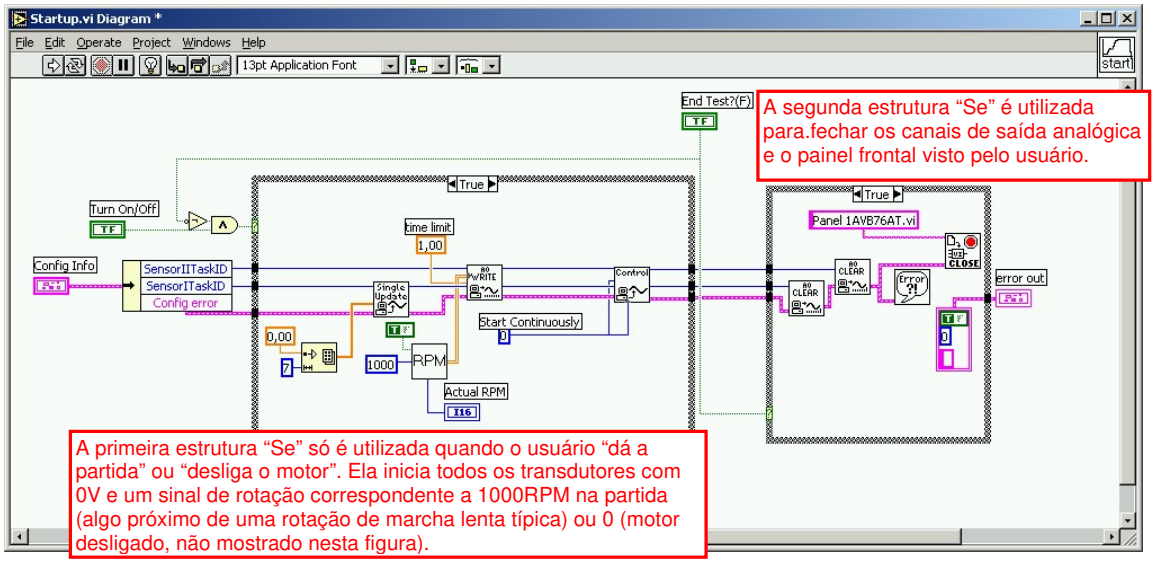


Figura 44: Startup.vi

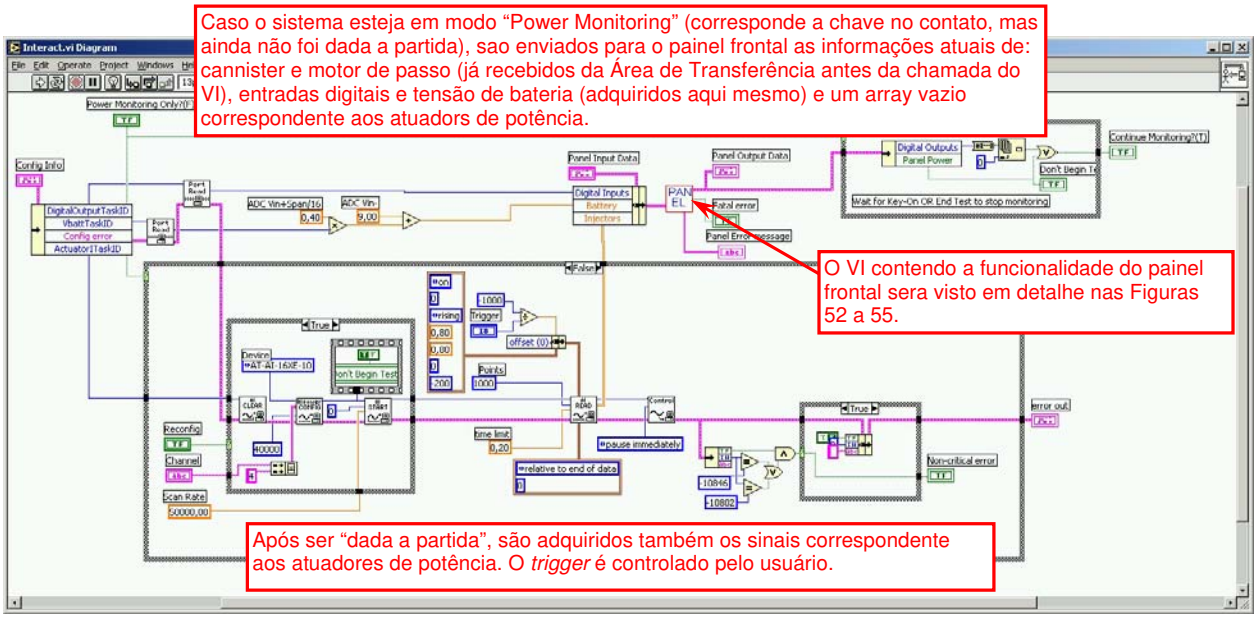


Figura 45: Interact.vi

No Interact.vi mostrado acima, encontramos as atividades de aquisição de dados das entradas digitais e da tensão de bateria. A tensão de bateria é amostrada por um conversor A/D no módulo de condicionamento de sinais que será visto no Capítulo 7 a seguir e, portanto é lida aqui como um sinal digital de 8 bits posteriormente escalado. Além destes tipos de componentes, também são adquiridos aqui os:

Atuadores de Potência:

Os atuadores de potência se dividem em dois tipos: bicos injetores e bobinas de ignição. Em ambos os casos as informações de interesse são obtidas através da monitoração da forma de onda da corrente, sua duração e instante de ativação, sempre em relação ao sinal do sensor de rotação.

Para garantir o melhor desempenho com nosso equipamento atual, foi estabelecido que só seriam adquiridos os sinais do bico injetor atualmente selecionado e da bobina de ignição. A aquisição é feita em blocos de 1000 pontos para cada sinal a uma taxa de 50KS/s. O *buffer* de aquisição possui 40000 posições. O instante de *trigger* é definido por um nível de 0,8V numa borda de subida no sinal do bico injetor e o número de pontos de pré-*trigger* é definido pelo Técnico através de controle existente no painel frontal.

Como o tempo de abertura do bico injetor está quase sempre abaixo de 20ms, esse é o tamanho da "janela" de visualização da sua forma de onda no painel frontal, no entanto o Técnico pode alterá-lo à vontade para visualizar um espaço de tempo maior ou um detalhe da forma de onda, utilizando o controle de *trigger* para posicionar a forma de onda na "janela" de visualização.

Para evitar paradas desnecessárias do sistema, os erros de sobre-escrita no *buffer* e/ou de tentativa de leitura sem dados ainda disponíveis, estão sendo ignorados e o gráfico da forma de onda simplesmente aguarda os próximos dados válidos antes de ser atualizado.

Como as placas DAQ utilizadas não permitem a utilização de múltiplos eventos de *trigger*, a verificação dos ângulos de avanço e dwell apresentou dificuldades quase que insuperáveis. Foram tentadas abordagens com a utilização de ocorrências (um recurso da linguagem LabVIEW, através do qual utilizávamos o temporizador interno do PC) e até com a aquisição do sinal de rotação recém-gerado junto com os sinais de corrente, para que rotinas de software analisassem o conteúdo do *buffer* de aquisição para extrair as informações de temporização. Essas alternativas foram descartadas para a versão final do sistema por serem extremamente custosas em termos de processamento, ou imprecisas. A solução mais adequada será melhor discutida na conclusão desse trabalho.

Antes de passarmos para o próximo VI, queremos esclarecer que na verdade, a geração em si dos sinais dos sensores de freqüência acontece no Runtest.vi e pode ser vista na Figura 43, mas como o mais importante é a geração das formas de onda que devem ser simuladas, deixamos para explicá-las aqui junto com o RPMArray_3.vi.

Sensores de Freqüência:

Os sensores de freqüência utilizados por ECUs são geralmente utilizados para medição da rotação do motor e velocidade do veículo e detecção de detonação. A detonação é reconhecida através de vibrações do bloco do motor em uma determinada freqüência que ocorrem próximo do PMS do cilindro detonante e é medida com um sensor baseado geralmente em cerâmicas piezoelétricas. O sinal produzido por esses sensores só é analisado nos instantes próximos a cada PMS e é filtrado para que se procure apenas a freqüência característica da detonação para cada motor.

Os sensores de rotação são geralmente indutivos e detectam flutuações no campo magnético induzidas por dentes de uma engrenagem montada junto ao rotor em questão. Eles podem ser de dois tipos: relutância magnética e efeito Hall. A principal diferença entre eles, sob um ponto de vista externo, é que sensores de relutância variam a amplitude do sinal de saída com a rotação, enquanto que sensores Hall têm o sinal de saída com a mesma amplitude da sua tensão de alimentação.

A simulação dos sinais de ambos os sensores de freqüência foi feita com canais de saída analógica convencionais. Nesta primeira versão não foi de nosso interesse verificar os circuitos de condicionamento de sinais internos à ECU, portanto os sinais simulados serão todos com amplitude de 5V. O sinal correspondente ao sensor de detonação será de forma senóide na freqüência característica de cada motor e o sinal do sensor de rotação obedeceu ao respectivo quadro de sinais da *centralina* sendo simulada. A Figura 46 mostra o quadro de sinais para *centralinas* da família IAW1AV.

A utilização de canais de saída analógica trouxe conseqüências adversas que só foram percebidas quando da implementação das rotinas de software para simular o sensor de rotação. Os sinais gerados devem refletir quase que instantaneamente as ações do Técnico no painel frontal e não podem ser interrompidos para evitar funcionamento anormal da ECU, principalmente o sinal do sensor de rotação. Isso implicou que deveríamos trabalhar com uma taxa de atualização e um tamanho de *buffer* fixos devido ao longo tempo necessário para reconfigurar esses parâmetros, durante o qual a placa não estaria gerando nenhum sinal.

O *buffer* deve ser utilizado quando o sistema não consegue disponibilizar os dados a serem gerados para a placa DAQ em tempo hábil para acompanhar a taxa de atualização utilizada, o que ocorre no nosso caso, pois o processo de montar matrizes de pontos representando cada forma de onda é relativamente lento. Como as placas DAQ esvaziam o *buffer* bem mais rápido do que conseguimos enchê-lo, devemos utilizar um *buffer* circular. Pelo mesmo motivo, o *buffer* **sempre** deve conter formas de onda completas, para que elas sejam geradas continuamente mantendo a referência de temporização da ECU (dente mais longo ou dentes faltantes).

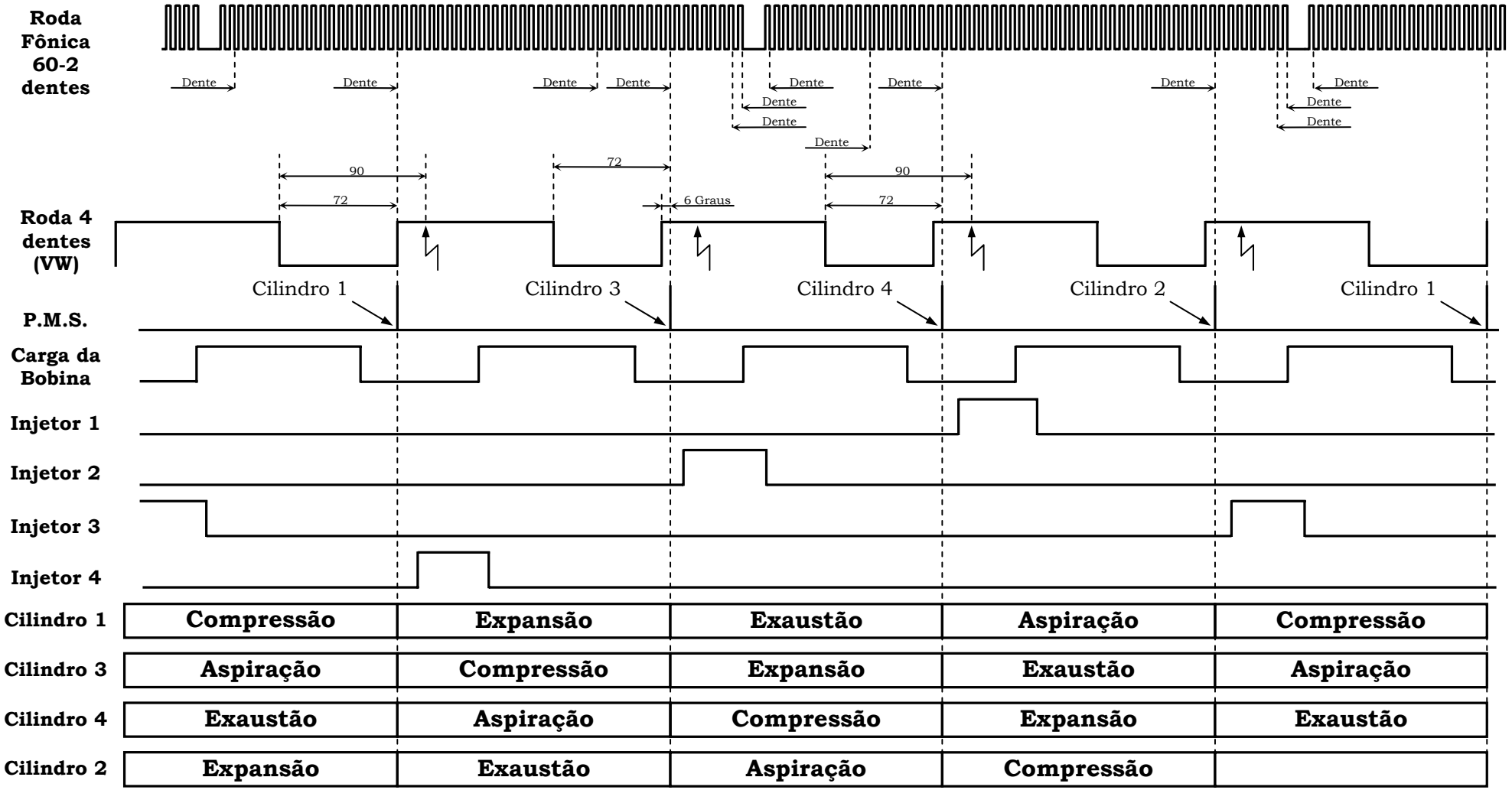


Figura 46: Quadro de sinais para *centralinas* da família IAW

Agora, uma taxa de atualização fixa implica em um intervalo de tempo fixo entre dois pontos gerados. Cada valor de rotação corresponde a um intervalo de tempo no qual deve ser gerada a forma de onda correspondente a uma volta da árvore de manivelas, de acordo com o '*quadro segnali*' utilizado. Portanto, cada valor de rotação implica em número fixo de pontos para representar a forma de onda correspondente. Finalmente, um tamanho de *buffer* fixo implica em que somente algumas formas de onda são capazes de preencher totalmente o *buffer*, sem sobrar ou faltar pontos. Essas formas de onda são aquelas para as quais o tamanho do *buffer* é múltiplo do número fixo de pontos que a representa!

Com essas considerações, passamos a buscar os valores de taxa de atualização e tamanho de *buffer* que nos proporcionasse o maior número de formas de onda possíveis de serem geradas adequadamente. Para garantir o reconhecimento do PMS do cilindro 1, necessitamos de uma resolução mínima de 6º para a roda de 4 dentes. Como um ciclo da forma de onda dessa roda equivale a 720º (2 voltas no eixo da árvore de manivelas, conforme a Figura 46), necessitamos de no mínimo 120 pontos para representá-la. Como nossa taxa de atualização é fixa, quanto mais pontos são utilizados para uma forma de onda mais tempo ela leva para ser gerada, portanto corresponde a uma rotação menor. Isso significa que os 120 pontos correspondem a duas voltas do eixo, na máxima rotação a ser simulada pelo sistema. Considerando que a máxima rotação atingida pelos motores utilizados atualmente em veículos particulares é algo em torno de 8000 RPM, a taxa de atualização pode ser dada por:

$$T_{at} = \frac{RPM_{máx}}{60} * \frac{P_{mín}}{2} \cong 8000\, pontos / seg$$

Como a taxa de atualização é obtida através da divisão da freqüência-base de clock da placa DAQ que é de 1MHz, utilizamos um divisor de 120 para uma taxa de 8333,33 pts/s. Recalculando a expressão anterior, a máxima rotação a ser simulada pelo sistema será de 8333,33 RPM.

Após diversas iterações, optamos por um *buffer* de 4200 pontos, o qual permitirá a simulação das rotações mostradas na Tabela 1, na página seguinte.

Para esta primeira versão, também optamos por não simular a detonação para cada cilindro, mas somente um efeito de detonação que fosse simultâneo para todos os cilindros. Como os dados a serem simulados são colocados previamente no *buffer* circular e são ininterruptamente gerados pela placa DAQ, não tínhamos mecanismos eficientes de sincronizar a simulação da detonação com o sinal de rotação. A solução mais adequada envolve a modificação dos conceitos de simulação desses sinais, com substituição de placas DAQ e será melhor discutida na conclusão deste trabalho.

Pontos por Forma de Onda	Rotação	Pontos por Forma de Onda	Rotação
120	8333,33	350	2857,14
140	7142,86	420	2380,95
150	6666,67	525	1904,76
168	5952,38	600	1666,67
175	5714,29	700	1428,57
200	5000,00	840	1190,48
210	4761,90	1050	952,38
280	3571,43	1400	714,29
300	3333,33	-----	-----

Tabela 1: Valores Válidos de Rotação

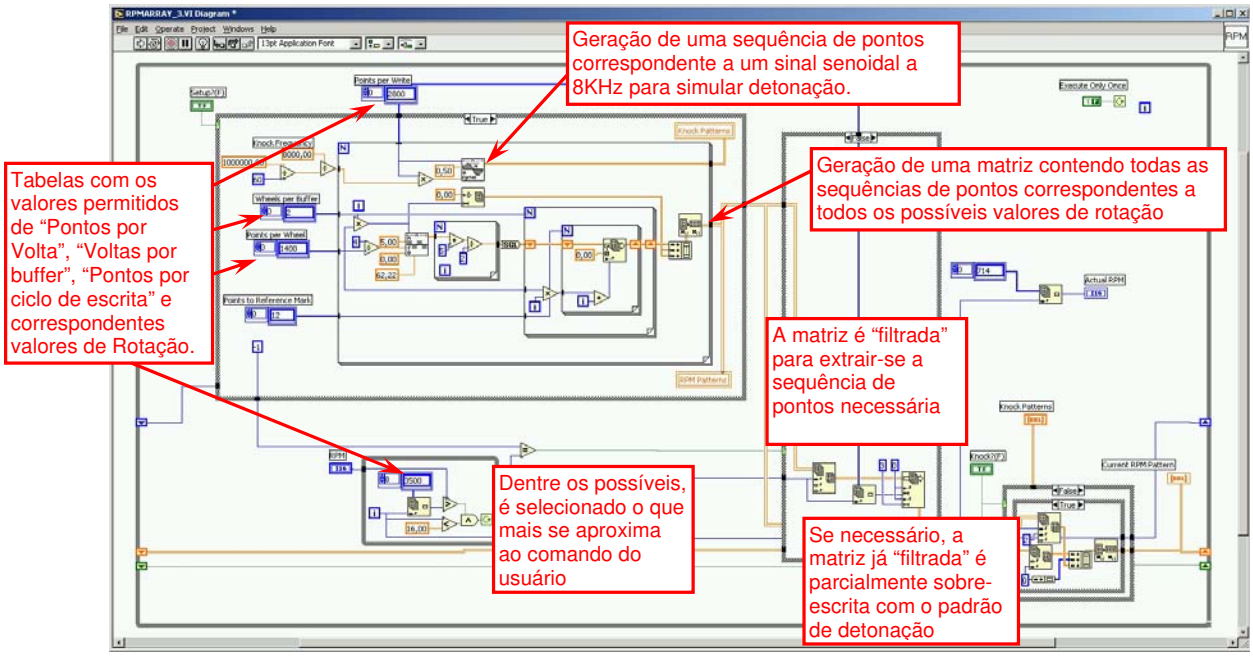


Figura 47: RPMArray_3.vi

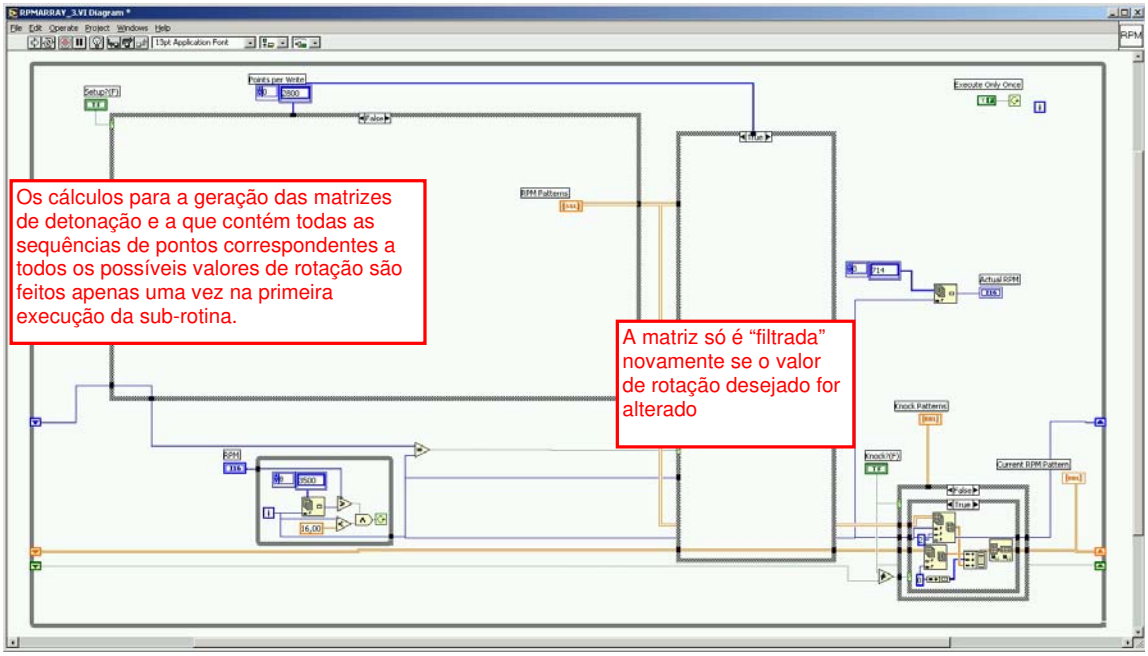


Figura 48: RPMArray_3.vi (otimização de processamento)

O próximo a ser analisado é o CheckStepPosit.vi, usado para aquisição do sinal do:

Atuador de Marcha Lenta:

O atuador de marcha lenta é o dispositivo que controla o fluxo de ar em um duto de *by-pass* da borboleta, através da movimentação de um pino que obstrui gradativamente esse duto. É através deste duto que o motor recebe ar para permanecer em funcionamento, quando a borboleta está totalmente fechada (motorista sem o pé no acelerador). O atuador de marcha lenta é portanto quem determina a rotação do motor quando o veículo está parado. Em motores carburados, esse pino é um parafuso que permanece em uma posição fixa, cujo ajuste é feito manualmente e depende do "ouvido" de quem ajusta.

Em veículos com sistemas eletrônicos, esses atuadores são geralmente de dois tipos: válvula eletromagnética e motor de passo. Para válvulas eletromagnéticas, o controle do curso do pino é feito através de *duty cycle*, neste caso utilizamos uma entrada de um dos contadores/temporizadores disponíveis em nossas placas DAQ, os quais podem realizar essa medição automaticamente. Como neste caso o deslocamento do pino não é uniforme, a ECU requer a existência de um sensor de posição para determinar sua posição a cada instante e poder determinar a estratégia adequada para o gerenciamento da marcha lenta. Esse sensor não é necessário quando da utilização de motores de passo, conforme veremos a seguir.

O motor de passo funciona através da polarização de duas bobinas posicionadas perpendicularmente entre si com valores iguais de tensão. Invertendo a polaridade das bobinas alternada e seqüencialmente, é gerado um campo magnético no estator cujo sentido varia de +/- 45º a cada inversão de polaridade, atraindo o campo magnético do rotor. Esse procedimento resulta num movimento do eixo em sentido horário ou anti-horário, com apenas 4 posições possíveis de parada do rotor. Para converter esse movimento circular em linear, o eixo gira sobre uma rosca sem fim, cujo comprimento associado ao diâmetro do eixo determina o número máximo de passos e o curso total do motor. Se denominarmos os terminais de uma bobina de '1' e '3' e os da outra bobina de '2' e '4', teremos um diagrama de sinais similar ao mostrado na Figura 49 para um motor de passo em movimento.

Se considerarmos ainda que o estado lógico do sinal de cada terminal é um digito binário, podemos estabelecer que só existem quatro combinações possíveis formando quatro números binários que se alternam. Esses números podem ser lidos na vertical na mesma Figura 49. Inicialmente o sistema simulador estava utilizando quatro entradas lógicas para avaliar continuamente o estado dos quatro terminais do motor de passo. À medida que os números binários adquiridos se sucediam, eles eram comparados com o diagrama de fases para se determinar o sentido de rotação do motor. No entanto a aquisição desses números através de uma porta digital de 4 bits tinha que ser feita um a um, pois nossas placas DAQ não ofereciam a possibilidade de se estabelecer um taxa de aquisição contínua para sinais digitais. Os atrasos gerados com a configuração da porta a cada aquisição estavam ocasionando a perda de passos e, portanto o sistema estava indicando erroneamente a posição atual do motor de passo. Atualmente essa porta digital está sendo usada apenas para detectar estados não válidos que podem representar problemas de mal-contato ou defeito neste dispositivo e a contagem dos passos teve que ser feita conforme a descrição que se segue.

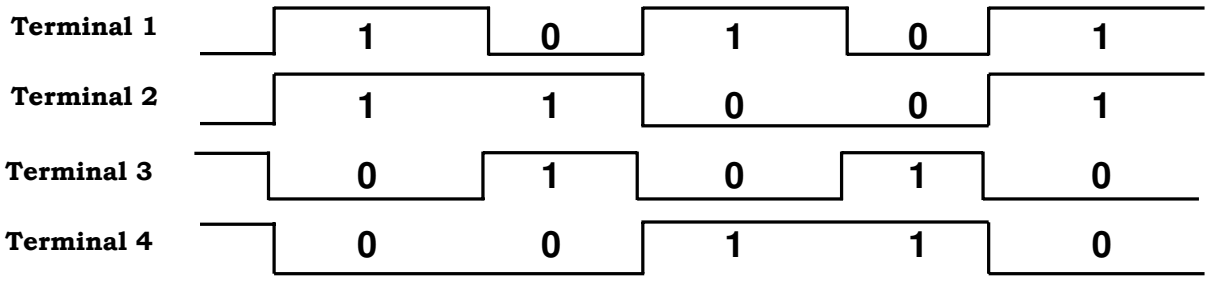


Figura 49: Motor de Passo em Movimento

Como os sinais em ‘3’ e ‘4’ são sempre o inverso dos sinais ’1’ e ‘2’ respectivamente, pois representam o outro terminal da mesma bobina, podemos lidar apenas com os dois últimos. Esse par de sinais é similar aos fornecidos por codificadores de quadratura (*quadrature encoders*), portanto utilizaremos seus conceitos para aquisição das informações de deslocamento do motor de passo. Codificadores são dispositivos que convertem deslocamentos lineares ou circulares em sinais ou pulsos digitais. Nosso interesse recai sobre os codificadores incrementais, que geram um pulso para cada passo incremental, fornecendo maior resolução a um menor custo. No caso de um movimento circular onde por meio de uma “trilha de código” é gerado um pulso a cada volta de um eixo, por exemplo, a freqüência do sinal gerado indica a velocidade de deslocamento, porém não é suficiente para determinar sua direção. Para tanto necessitamos de um codificador de dois canais ou de quadratura, com duas “trilhas de código” no eixo, usualmente defasadas em 90º. Por exemplo, se a borda de subida (ou de descida) do canal ‘A’ preceder a equivalente do canal ‘B’, pode-se determinar se o movimento é no sentido horário ou anti-horário, de acordo com a definição dos canais.

Os contadores/temporizadores disponíveis nas placas DAQ têm uma série de facilidades, dentre as quais a habilidade de contar bordas no sinal conectado à sua entrada SOURCE, de forma incremental ou decremental de acordo com o estado lógico de um sinal digital conectado à sua entrada UP_DOWN. Observando novamente a na Figura 49, podemos observar que não haveria nenhum problema aparente em se conectar diretamente os sinais ‘1’ e ‘2’ a essas entradas, pois as bordas de ‘1’ sempre ocorrem num determinado estado de ‘2’ quando em movimento num sentido e no estado inverso quando o movimento é no sentido oposto. No entanto, vibrações no eixo, ruído e/ou *jitter* no sinal poderiam gerar pulsos espúrios contados indevidamente. Para garantir a confiabilidade das nossas medições, nós utilizamos o conversor de clock de quadratura (*quadrature clock converter*) LS7084, da LSI Computer Systems Inc., que possui filtros passa-baixa para eliminar os efeitos de ruído e *jitter* e executa a contagem das bordas num dos canais sempre verificando se também houveram transiçõees no outro para evitar a contagem indevida. Esse CI fornece duas saídas: um sinal que pulsa uma vez para cada contagem e um sinal com o sentido do movimento; ambos podendo ser conectados diretamente às entradas SOURCE e UP_DOWN.

Como pudemos observar no diagrama de estados da Simulação mostrado na Figura 22, a monitoração do motor de passo é feita por uma sub-máquina de estado independente, que inicia suas atividades imediatamente após o evento de *key-on*, pois nesse instante a ECU comanda o motor de passo para retrair-se um número de passos superior ao número máximo possível. Isso faz com que ele gire várias vezes em falso na rosca sem fim, garantindo que o motor se encontra em sua posição inicial e fornecendo uma referência de posição confiável, uma vez que a partir daqui cada movimento é anotado para que se conheça a posição exata do motor a cada instante.

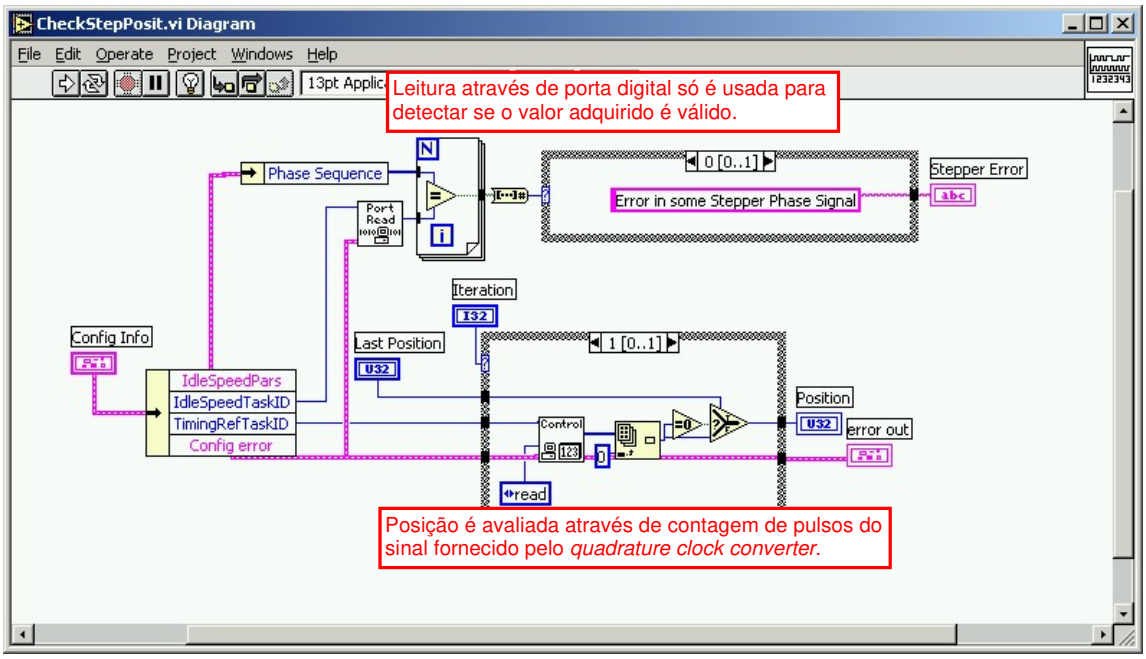


Figura 50: CheckStepPosit.vi

Chegamos então ao último VI mencionado na Figura 43. Esse VI, mostrado na Figura 51 a seguir é responsável pela geração dos sinais correspondentes aos:

Transdutores:

Ao contrário dos sensores de freqüência, os transdutores representam o estado atual de uma grandeza física através de um valor "estático" de uma grandeza elétrica, é claro que este valor varia tão rápido quanto varie a grandeza física correspondente no ambiente. O nosso sistema se propõe justamente a simular um ambiente, e no início do trabalho essa simulação era estática tal qual os simuladores manuais atuais, ou seja, o simulador só respondia à atuação dos comandos do Técnico no painel frontal. Como nenhuma pessoa consegue modificar os controles que representam as grandezas físicas do ambiente muito rapidamente (comparando-se com a velocidade de processamento de um PC), a atualização dos valores dos transdutores está sendo feita somente quando necessário.

No entanto, percebemos que esta condição não é suficiente para atender fielmente as simulações dinâmicas que nos propomos a realizar. Para a simulação de scripts essa estratégia apresenta uma limitação na resolução temporal disponível – há um tempo mínimo permitido (que depende da velocidade de processamento do PC) entre a atualização de dois valores consecutivos. Quando da implementação futura de uma simulação realimentada através de modelamentos de motores, essa limitação torna-se proibitiva, pois não podemos dizer a um modelo que a pressão do ar no coletor de admissão só pode ser alterada daqui a 1ms, por exemplo.

Esse problema só será resolvido com a implementação de uma geração de sinais contínua a uma alta taxa de atualização, e com a utilização de um ou mais *buffers* de saída para os valores dos transdutores a serem simulados.

Mesmo com essa limitação, foi implementado um comportamento especial para o sinal da sonda lambda. A sonda lambda mede a quantidade de oxigênio nos gases de escape e serve como parâmetro para que a ECU verifique se a titulação da mistura ar-combustível estava de acordo com o valor que ela esperava. Supondo que o Técnico posicionasse o controle da sonda para um valor que indique "mistura pobre", a ECU passaria a atuar fornecendo mais combustível (aumentando o tempo de abertura dos bicos injetores, por exemplo) e como não haveria uma resposta adequada da sonda, ela permaneceria injetando combustível em demasia até que o Técnico modificasse esse valor! Apesar de não termos conhecimento profundo quanto à resposta de nenhum motor, é razoável considerar que, uma ECU que esteja funcionando corretamente **pelo menos neste aspecto**, irá aumentar a quantidade de combustível e essa atuação será refletida na sonda lambda. Para tentar atender a gregos e troianos, além das simulações normal do valor e de desconexão que seguem os mesmo princípios dos outros transdutores, foi implementado uma simulação chamada de "pulsada", onde o valor simulado para a sonda lambda fica alternando entre o valor indicado no controle do painel frontal e o valor correspondente a uma mistura na relação estequiométrica. O Técnico seleciona o tipo de simulação desejada no painel frontal.

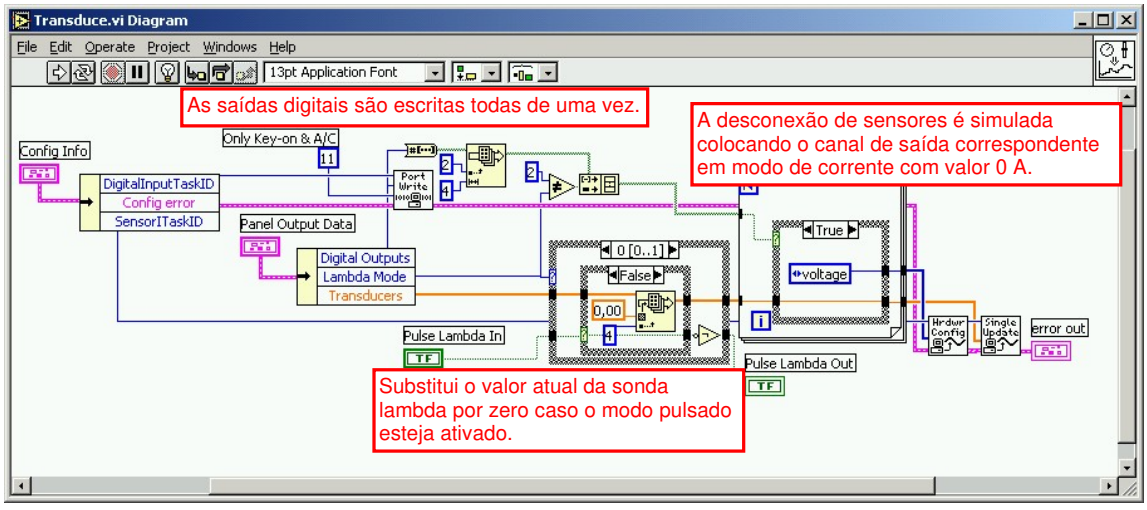


Figura 51: Transduce.vi

Finalmente chegamos ao painel frontal, já mencionado na Figura 41, sendo chamado pelo Runtest.vi e na Figura 45, chamado pelo Interact.vi. No primeiro caso, o painel frontal entra em modo de inicialização como pode ser visto abaixo.

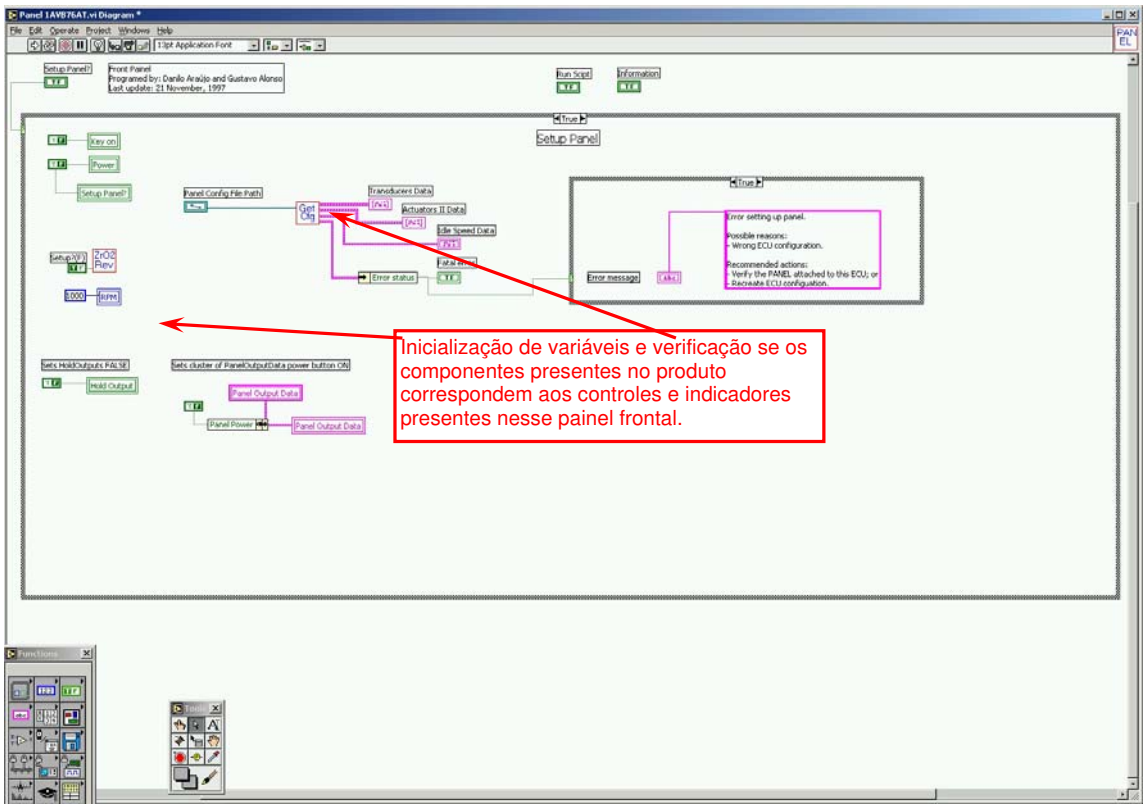


Figura 52: Panel 1AVB76AT.vi (Modo de inicialização)

Já as seguintes figuras mostram o painel frontal em operação normal, ou seja, quando é chamado pelo Interact.vi. Para facilitar a interpretação do código e evitar conflito de sincronização entre as ações do usuário, utilizamos uma estrutura de "seqüência" para que atividades similares sejam realizadas mais ou menos em bloco. Dessa forma decidimos iniciar com uma atualização dos controles no painel frontal, ou seja, disponibilizar visualmente para o usuário os últimos valores já adquiridos pelas placas DAQ e que se encontram na Área de Transferência. Essas atividades podem ser vistas na Figura 53. Em seguida executamos as modificações do usuário que podem ser realizadas dentro deste mesmo vi, como veremos na Figura 54. Por fim disponibilizamos as ações do usuário na Área de Transferência para que as outras respectivas sub-rotinas atualizem a simulação dos sensores, como veremos na Figura 55.

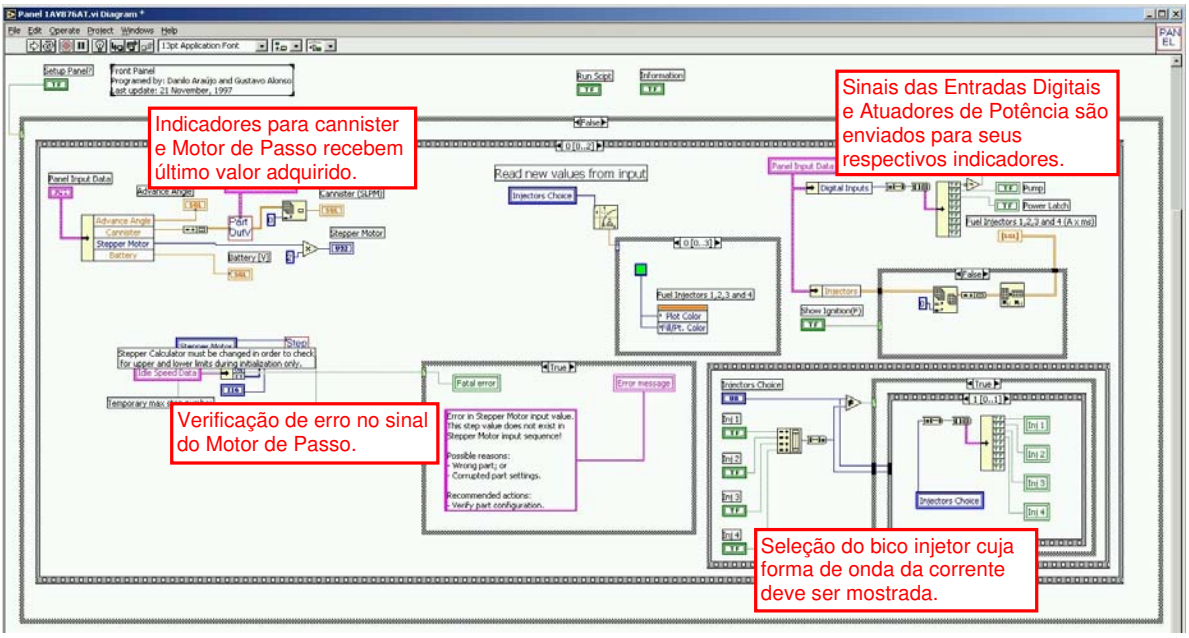


Figura 53: Panel 1AVB76AT.vi (Mostra respostas da ECU no painel)

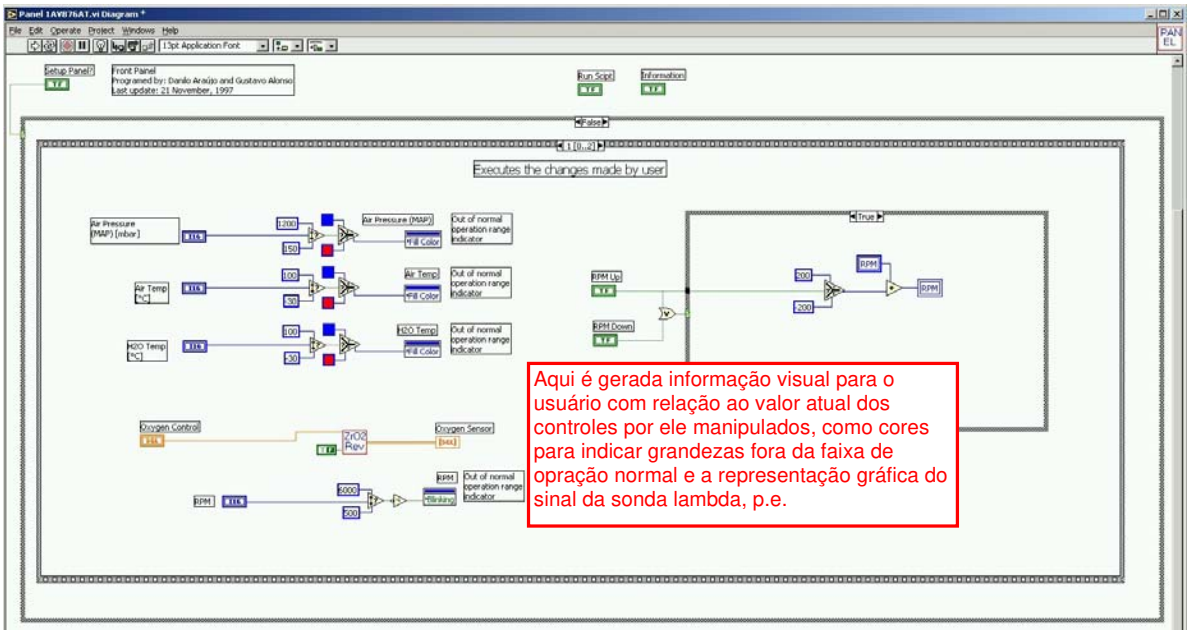


Figura 54: Panel 1AVB76AT.vi (Executa modificações do usuário)

As funcionalidades “Hold Output” e “Run Script” mencionadas no item 7.1 – Executar Teste (Simulação), são realizadas no trecho de código mostrado na Figura 55 abaixo.

No modo “Hold Output”, ao invés de enviar o valor atual dos indicadores no painel, deixamos a Área de Transferência inalterada e a simulação continuará funcionando com o último status enviado.

No modo “Run Script”, o princípio é o mesmo, mas enviamos valores oriundos de um arquivo de *script*. Essa flexibilidade é uma conseqüência direta da utilização da Área de Transferência como interface entre os comandos executados no painel e o Bloco de Software responsável pela simulação.

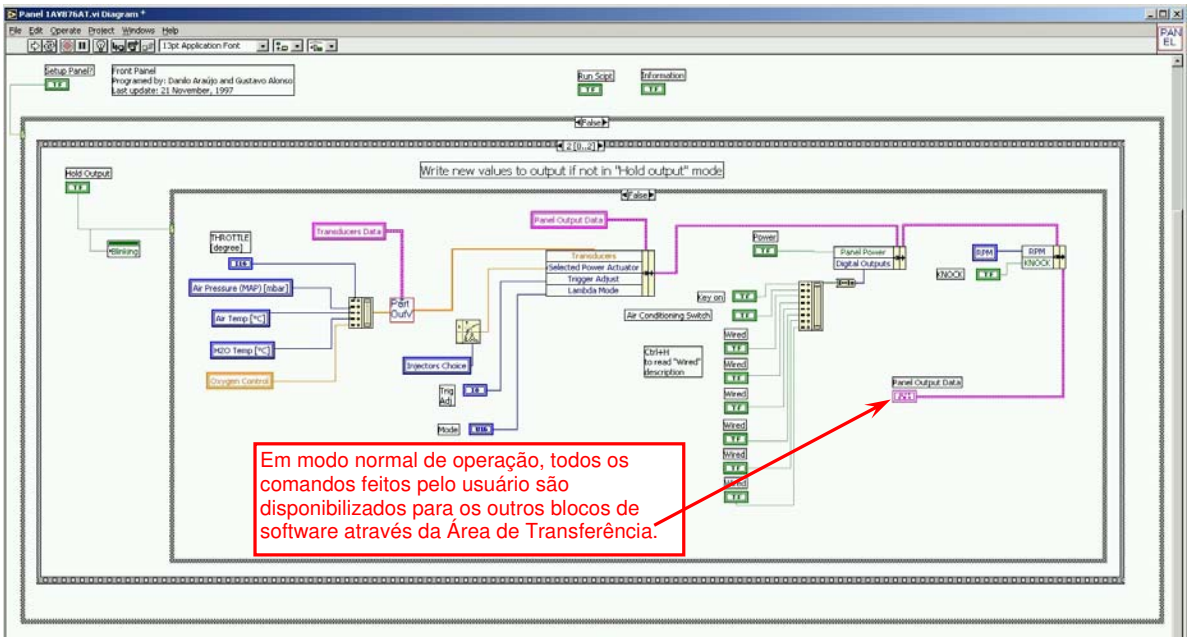


Figura 55: Panel 1AVB76AT.vi (Disponibiliza ações do usuário)

Capítulo 7 – Realização do Hardware

7.1 Módulo de Condicionamento de Sinais

O módulo de condicionamento de sinais é o elemento de interface entre o produto sob teste e as placas DAQ instaladas na plataforma utilizada, seja ela uma versão bancada ou portátil. Este módulo tem uma configuração mínima que deve ser respeitada a todo custo, com poucos elementos que poderiam ser considerados opcionais. Portanto não existe a possibilidade de se obterem vantagens significativas de tamanho ou peso que justificassem o desenvolvimento de múltiplas configurações desse módulo, para um mesmo tipo de produto a ser testado. O aproveitamento de um único módulo para diversos produtos é uma possibilidade ainda não descartada por completo, devido à grande similaridade e redundância dos diversos sinais utilizados por produtos de eletrônica embarcada. Esse aproveitamento poderia ser feito com o desenvolvimento de uma placa de circuito impresso que proveja diversas funcionalidades, com somente algumas delas sendo utilizadas para cada tipo de produto. Para minimizar o número de pinos nos conectores de entrada e saída do módulo, seriam utilizados jumpers para direcionar cada sinal a seu circuito de tratamento correspondente. Esse módulo deve possuir conectores padrão para as placas e um único conector para a *centralina* sob teste, conforme discutido no Capítulo 3.

Tendo em vista, mais uma vez, a aplicação de uma *centralina* como produto-piloto de testes para efeito de validação do simulador, será desenvolvido um módulo de condicionamento que atenda somente a seus requisitos. Utilizando a notação padronizada no modelamento do simulador, podemos classificar os sinais da *centralina* em nove grupos distintos, com diferentes requisitos: Transdutores, Sensores de Freqüência, Atuadores de Potência, Atuadores em Freqüência, Atuadores de Marcha Lenta, Entradas On-Off, Saídas On-Off, Serial e Alimentação/Referência. Vamos agora explorar as características de cada grupo e agrupar características semelhantes para determinar os tipos de circuitos de condicionamento adequados a cada um dos sinais.

Alimentação/Referência:

Este será o primeiro grupo a ser tratado por ser obviamente o mais fundamental para o funcionamento da *centralina* e de toda a circuitaria presente no módulo. Como veremos mais adiante, estarão presentes no módulo não somente circuitaria mas também várias cargas automotivas reais. E uma característica da maioria dessas cargas é que sua alimentação é realizada localmente. Assim o controle da *centralina* se resume em chavear a carga para a massa, permitindo ou não a passagem de corrente e portanto sua atuação. Isso implica na existência de um barramento com a tensão da bateria capaz de alimentá-las. Para tanto iremos utilizar uma bateria automotiva com capacidade de 40Ah a uma tensão nominal de 12V conectada ao módulo, que poderá ser substituída futuramente por uma fonte de tensão DC variável de 9-15V com capacidade de fornecimento de 10A. Por enquanto assume-se que o uso do simulador seja restrito a locais aonde haja uma bateria disponível.

Em um ambiente real, a alimentação desse barramento está condicionada à ativação do relé "Pump" pela *centralina*, após a ação de *key-on* ter sido executada pelo usuário. A *centralina* também gerencia sua própria alimentação através do relé "Power Latch". Para verificar essa funcionalidade e garantir às *centralinas* um *shut-down* apropriado, essa montagem será implementada com dois relés e uma chave *On-Off*. Esses relés poderão ser isolados através de jumpers, caso em que a chave *On-Off* alimentará diretamente o barramento e a *centralina*. Esta montagem pode ser observada na Figura 56. Nesta figura podemos observar as conexões das bobina e chaves dos relés “Power Latch” e “Pump” e da dupla chave *On-Off*, identificadas com os números 1, 2 e 3, respectivamente.

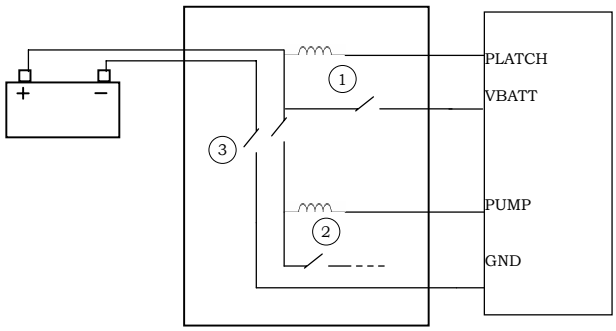


Figura 56: Conexão dos Relés Power Latch, Pump e Chave Key-On

Ao contrário das cargas reais, a circuitaria de condicionamento de sinais exige tensões de referência muito mais “limpas” e constantes, imunes ao ruído presente inevitavelmente na bateria e às flutuações provocadas por essas cargas. Essas tensões, de +12, -12 serão fornecidas pela fonte de alimentação do PC, através de conector específico e a tensão de +5 VDC seria fornecida pela própria *centralina* através dos pinos de alimentação de sensores V_HALL, V_APS e V_FARF. No entanto a *centralina* não foi capaz de fornecer a corrente necessária (cerca de 1,5mA) e também tivemos que utilizar a fonte de alimentação do PC.

Pelas mesmas razões, o pólo negativo da bateria (*massa telaio*) não pode ser utilizado como referência de terra para a circuitaria e para os sinais a serem gerados e/ou adquiridos pelas placas DAQ, mesmo sendo ele o terra de alimentação da *centralina*. Para tanto devemos utilizar a mesma referência de terra da instrumentação de precisão interna à *centralina* (*massa segnali*). Essa referência está disponível através dos sinais GND_SENS e GND_KNOCK e será conectada a todos os sinais de terra das placas DAQ. O sinal erroneamente chamado de GND_LAMBDA (na verdade ele corresponde ao terminal negativo da sonda lambda) será conectado também a essa referência por corresponder ao “terminal negativo” do sinal simulado pelo sistema. Os sinais SH_KNOCK e SH_LAMBDA (que correspondem à blindagem dos cabos do sensor de detonação, da sonda lambda e, internamente, ao *Tuner Box* ou *massa radiofrequenza*), também serão conectados a essa referência.

Atuadores:

Os atuadores se dividem em três grupos: de potência, em freqüência e de marcha lenta. Sua principal característica em comum é que, na maioria dos casos, o sistema requer a existência da carga real comandada pela *centralina*. As saídas On-Off também podem opcionalmente estar conectadas a um relé automotivo ou a um resistor de pull-up.

Cargas Reais:

A presença física de algumas cargas reais torna-se imprescindível quando suas características, em geral indutivas e/ou variáveis, tornam impossível sua simulação ao nível de software. Testes da capacidade de fornecimento de corrente pelos drivers do produto, curto-circuito à massa e à bateria, imunidade do sistema a *spikes* de tensão e outros, só podem ser realizados em situações pseudo-reais, utilizando as mesmas cargas que geram não-linearidades em um ambiente automotivo.

A conexão dessas cargas com a eletrônica será feita com os mesmos conectores utilizados nos veículos em produção. Dessa forma, para cada modelo sob teste, somente as cargas reais pertinentes a ele necessitam estar presentes. Estarão disponíveis no módulo de condicionamento de sinais para uma *centralina* os conectores para as seguintes cargas reais:

- 4 bicos injetores MPI
- 1 bico injetor SPI *
- 3 bobinas de ignição **
- 1 motor de passo
- 1 motor DC *
- 1 válvula cannister
- 1 válvula EGR *
- 4 relés automotivos **

* Disponível em futuras versões ** Nem todos disponíveis nesta versão

Cada grupo de atuadores requer um tipo de circuitaria para condicionar seus sinais de interesse: a corrente na carga, para os atuadores de potência; o duty cycle, para os atuadores em freqüência e a presença ou ausência de tensão em cada fase do atuador de marcha lenta.

Atuadores de Potência:

Nossa sonda de corrente será um resistor de valor conhecido conectado em série com a carga cuja corrente deve ser medida, sobre o qual medimos a tensão entre seus terminais. O amperímetro ideal apresenta resistência nula, caso limite onde sua presença não influencia a corrente a ser medida; mas para fins práticos basta utilizarmos uma resistência muito menor do que a impedância no resto da malha.

Os atuadores de potência em questão são bicos injetores e bobinas de ignição. Estas cargas possuem impedâncias mínimas típicas da ordem de 10 e 1Ω, portanto iremos utilizar sondas de 0.1 e 0.024Ω (3W-1% de precisão), cuja interferência pode ser considerada desprezível. Apesar dessas correntes poderem chegar até 7.5A, elas só existem durante o tempo de atuação de cada carga - tempo de injeção ou tempo de Dwell -, nunca superiores a 20 ms por ciclo motor; o que corresponde a uma dissipação máxima de potência muito pequena nos resistores (0.75W * 20ms = 15mJ), não representando perigo para o mesmo.

Cada sinal de tensão adquirido deverá portanto ser inferior a 1V, porém muito ruidoso. Esses sinais serão amplificados por amplificadores diferenciais de instrumentação que têm como funções: melhorar a resolução do sinal a ser adquirido pelas placas DAQ, eliminar a tensão de modo comum e protegê-las contra possíveis transientes de tensão. A tensão máxima suportada por cada canal analógico nas placas DAQ, quando desligadas, é de 15V. Portanto para garantir a segurança das placas, a alimentação dos amplificadores será de 12V e o ganho dos amplificadores para cada tipo de carga deve permitir total excursão do sinal na saída sem nunca chegar à saturação, em condições normais de operação.

Como exemplo, quando a *centralina* corta a atuação das cargas, a energia armazenada nessas bobinas pode gerar picos de tensão reversa da ordem de até 300V com duração de 100μs. E por estarem todas alimentadas através do mesmo barramento, esse ruído interfere em todas as cargas. Para evitar que esses transientes danifiquem os amplificadores diferenciais, cada carga terá um par de diodos de *free-wheeling* conectados em paralelo e conduzindo para o barramento de alimentação. O diodo 1N4148 tem uma resposta rápida logo no início do transiente, mas não suporta essa corrente por muito tempo. Logo a seguir o 1N4007 entra em condução e descarrega a maior parte da carga da respectiva bobina. A Figura 57 mostra o esquemático de um bloco de condicionamento composto pela sonda de corrente, um amplificador diferencial de ganho 20 e os diodos de proteção. O módulo de condicionamento tem seis conjuntos como este disponíveis.

Freqüência e Marcha Lenta:

Sob o ponto de vista do sistema, sinais que representam presença ou ausência de tensão e duty cycle pertencem ao domínio digital. O único inconveniente é que, no caso automotivo, o nível lógico alto é representado pela tensão de bateria e o nível lógico baixo pela massa. Conforme discutido anteriormente, ambas são referências muito ruidosas e a tensão de bateria ainda pode variar de 9 a 15V! Para adquirirmos esses sinais através das placas DAQ, precisamos convertê-los para níveis TTL. E é de fácil constatação que um simples divisor de tensão praticamente resolve esse problema. Ainda são necessários dois diodos para limitar os valores máximo e mínimo possíveis aos níveis aceitáveis pelas placas DAQ, e portanto, uma referência de 5V e um terra.

O circuito conectado em paralelo com cada uma dessas cargas pode ser observado na Figura 58, e o módulo de condicionamento tem dez circuitos como este disponíveis. O mesmo circuito deverá ser utilizado para adquirir os sinais TACHOM - uma repetição via hardware do sinal de rotação fornecido pelo sistema simulador (quadro de sinais) e CONSUM – consumo de combustível geralmente codificado em ciclo ativo, quando estes estiverem presentes na *centralina*.

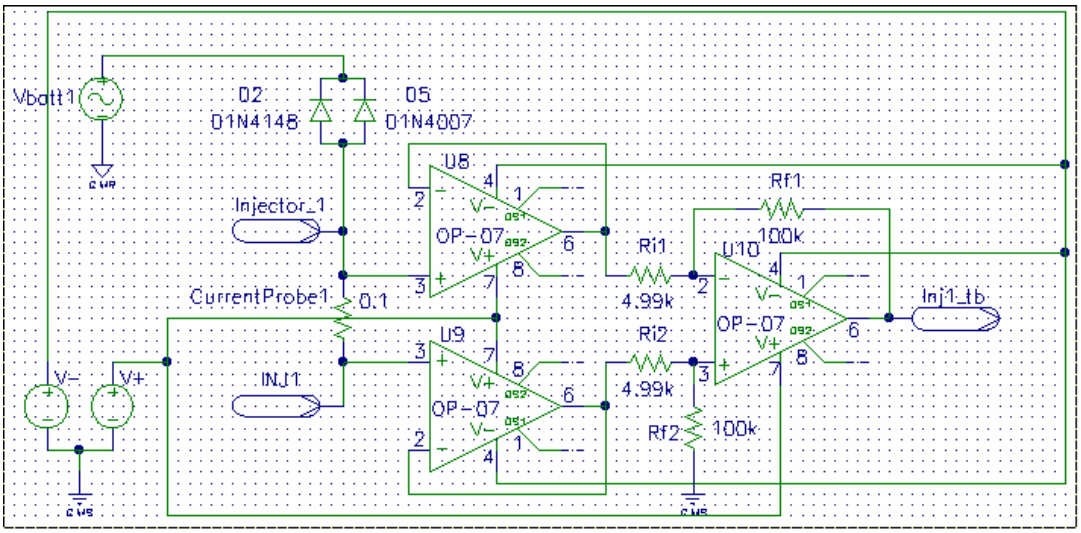
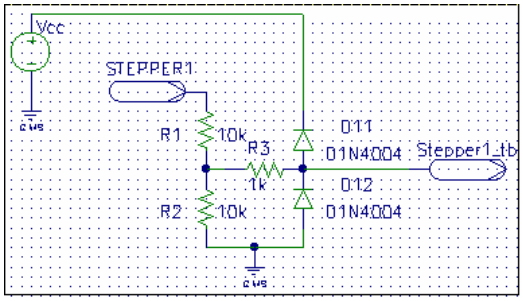
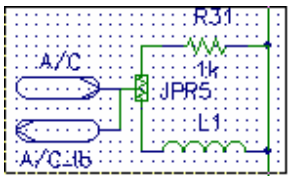


Figura 57: Bloco Amplificador de Instrumentação





Figuras 58 e 59: Bloco Divisor de Tensão e Opção de relé para Saídas On-Off

Saídas On-Off:

As saídas On-Off em geral também funcionam com alimentação local, mas como as cargas reais neste caso são os relés e não a carga realmente acionada, essa alimentação não precisa ser necessariamente a tensão de bateria. Assim podemos alimentar os relés diretamente com a nossa referência de 5V, dispensando o condicionamento do sinal. Além disso, esses sinais não possuem funções de diagnóstico pela *centralina* e sua característica indutiva é indiferente, portanto podemos substituir os relés por resistores! Essa opção será feita através de jumpers, conforme a Figura 59. O módulo de condicionamento tem cinco circuitos como este disponíveis.

Caso hajam saídas On-Off alimentadas pela *centralina*, deverão ser utilizados circuitos de condicionamento idênticos aos mostrados pela Figura 59.

Serial:

A comunicação serial com a *centralina* é realizada através das linhas seriais K e L. Dependendo da aplicação e com relação à *centralina*, a linha K pode ser bidirecional ou de saída e a linha L pode ser de entrada ou não estar em uso. O único ponto em comum é que, como tudo no ambiente automotivo, o nível lógico alto é representado pela tensão de bateria e o nível lógico baixo pela massa. Por sua vez, a interface serial do computador opera em padrão RS-232.

Ao invés de projetar uma circuitaria que realizasse essa conversão, separasse os sinais de entrada e saída na linha K e provesse todas as proteções para garantir a integridade do computador, optamos por usar transceivers disponíveis no mercado, fazendo a conversão em duas etapas. Primeiro utilizamos o CI patenteado Marelli TY93051 que desloca os níveis das linhas K e L para níveis TTL. Em seguida utilizamos o chip MAX232 que converte níveis TTL para RS-232. A Figura 60 mostra o esquemático do circuito já com os conectores de ponte (JPR1 - JPR4) para fazer a opção de tipo de aplicação.

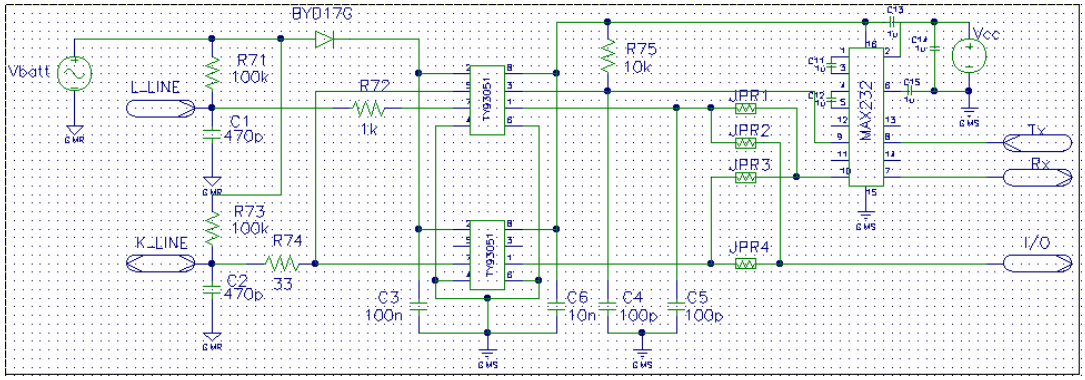


Figura 60: Interface Serial

Entradas On-Off:

As entradas On-Off se classificam conforme o sinal interpretado quando ela não está conectada (flutuante), ou seja, se possuem um resistor de Pull-Up ou um de Pull-Down. E como tudo o mais, o nível alto é a tensão de bateria. A Figura 61 mostra circuitos deslocadores de nível para operar com uma entrada em Pull-up e com uma entrada em Pull-Down, respectivamente. O módulo de condicionamento tem cinco circuitos de cada tipo disponíveis.

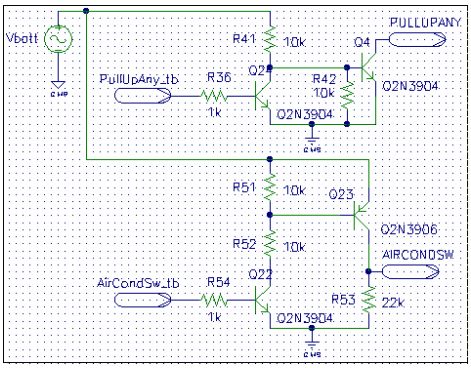


Figura 61: Deslocadores de Nível para Entradas em Pull-Up e Pull-Down

Transdutores e Sensores:

Como a impedância de entrada desses sinais na *centralina* é muito alta, poderíamos conectá-los diretamente das placas DAQ. No entanto, para facilitar a conexão, esses sinais serão redirecionados dentro do módulo de forma a termos somente um conector entre o sistema e a *centralina* (os conectores entre o módulo e o computador são considerados como pertencentes ao sistema, pois independem do produto sob teste).

Para compreendermos melhor a realização final do sistema na Figura 62 abaixo, repetimos alguns elementos da Figura 10.

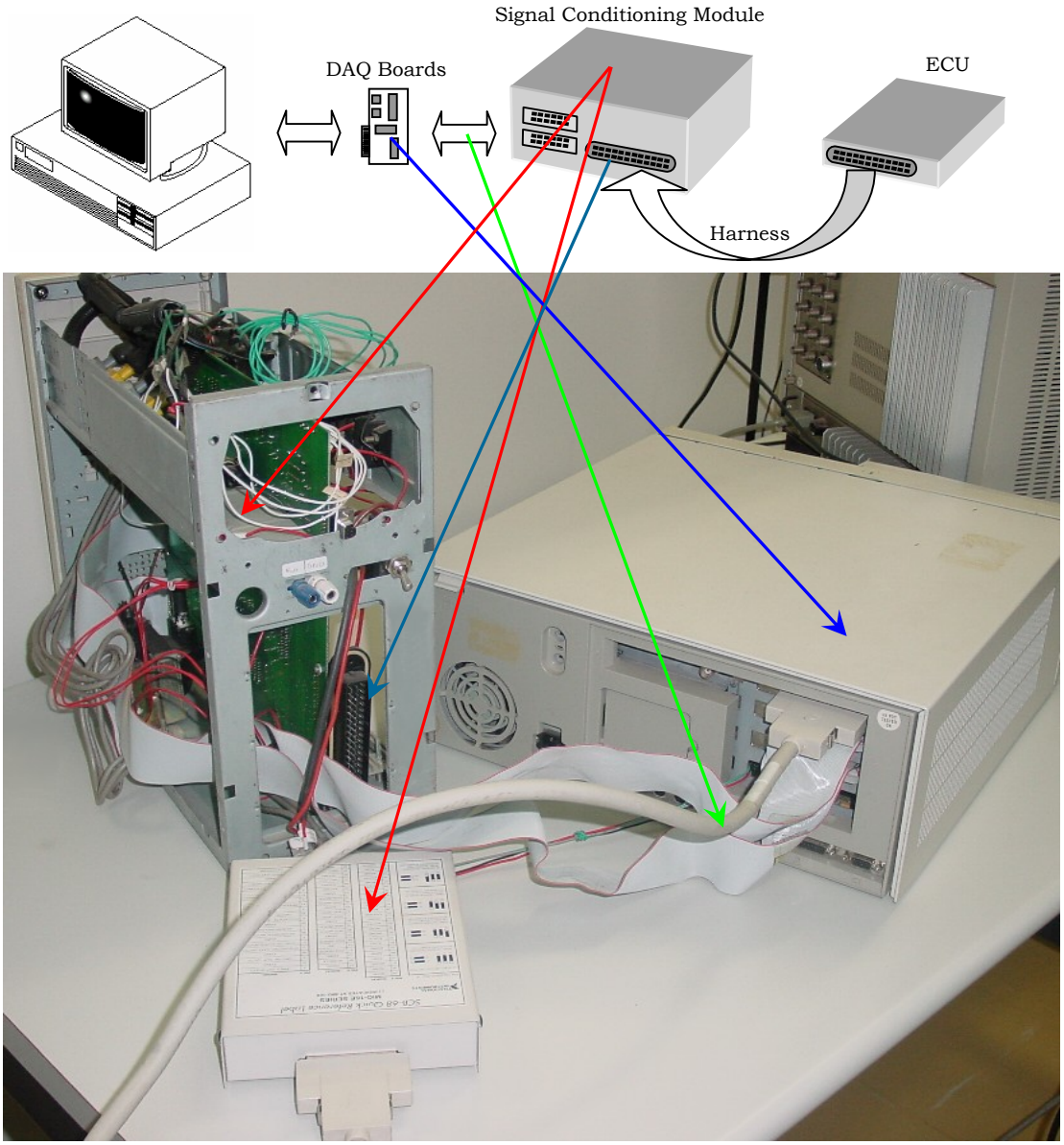


Figura 62: Módulo de Interface do AutoSEFI

As Figuras 63 e 64 a seguir, mostram mais alguns detalhes desse módulo.

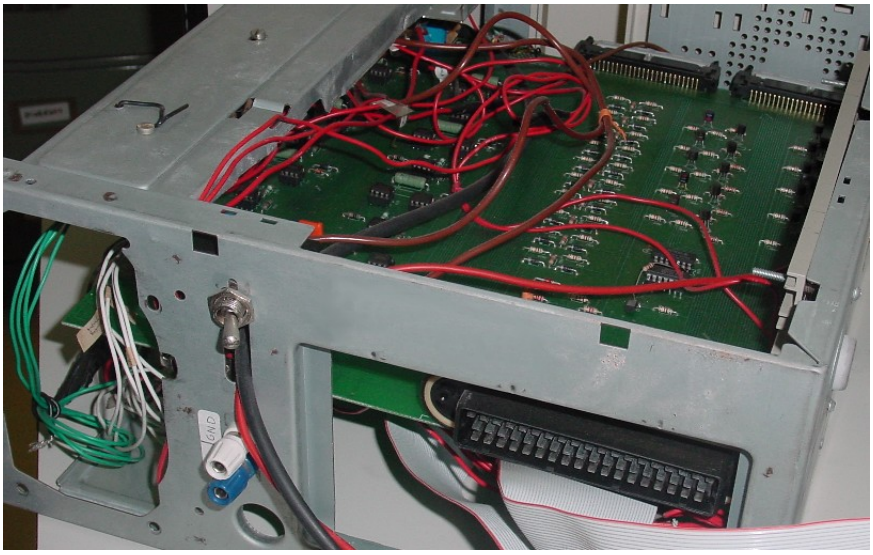


Figura 63: Plugues de conexão da bateria e chave "Key-on"

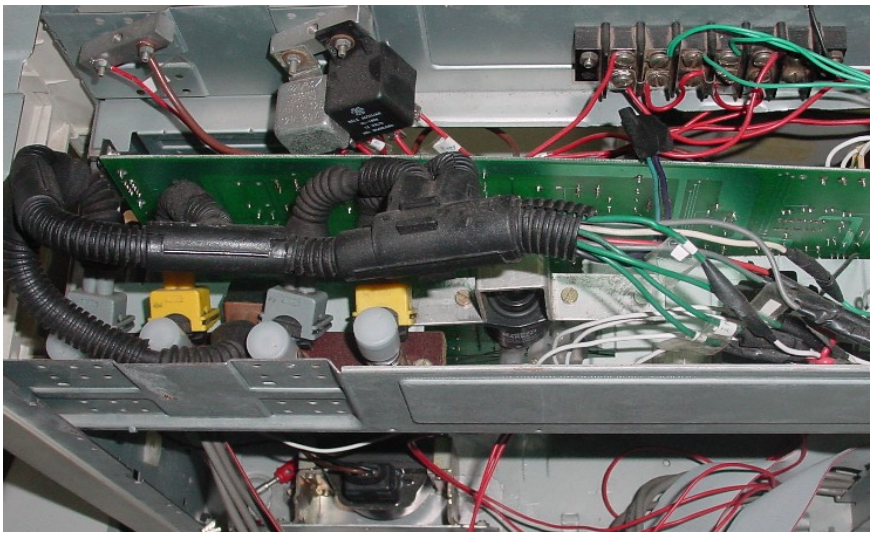


Figura 64: Cargas reais e relés de "Power Latch" e "Pump"

Capítulo 8 – Avaliação do Sistema

8.1 Parâmetros dos Componentes-Exemplo e Produto-Exemplo

Para a validação do sistema, cadastramos no sistema diversos Componentes utilizados pela Magneti Marelli, bem como um de seus Produtos. O Produto-Exemplo utilizado foi a *centralina* modelo 1AVB76AT, fornecida para motores da Volkswagen. Essa *centralina* utiliza os seguintes componentes:

- Sensor comb. de pressão e temperatura do ar no coletor de admissão modelo TPRT04

  - Função de Transferência: V=0.0429*P-0.3929 (V, P=mbar)

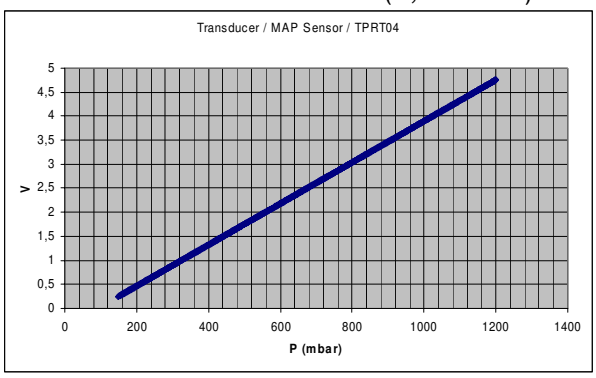


Figura 65: Sensor MAP

  - Função de Transferência (best fit): V=5.789*exp(5.789*T) (V, $R_{Pull\text{-}Up}$=2K)

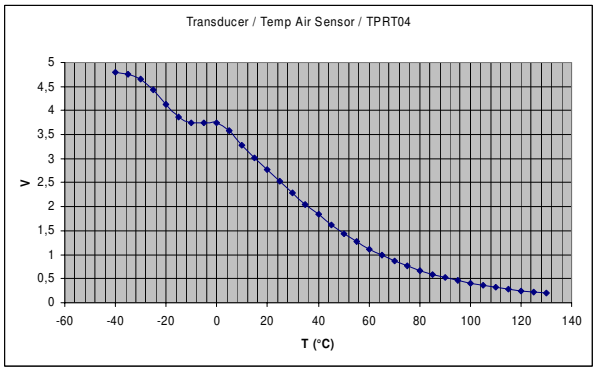


Figura 66: Sensor T Air

- Sensor de temperatura do líquido de arrefecimento modelo WTS05
  - Função de Transferência (best fit): V=10.91*exp(10.91*T) (V, $R_{Pull\text{-}Up}$=2K)

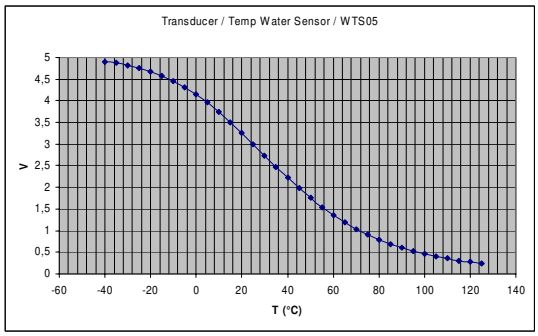


Figura 67: Sensor T H²O

- Sensor de posição da borboleta modelo PF1C00
  - Função de Transferência: V=46.15*D+155.8 (mV, Vcc=5V)

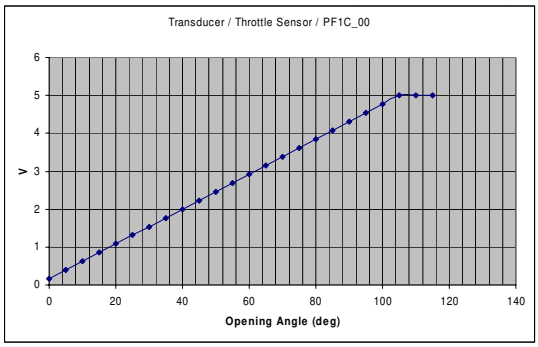


Figura 68: Sensor *Throttle*

- Sonda Lambda modelo OZES0017

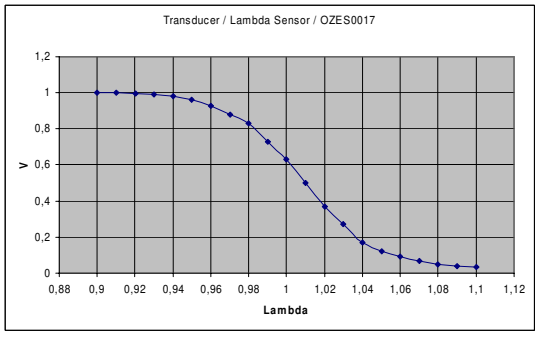


Figura 69: Sonda Lambda

- Sensor de detonação modelo KNE02
  - Amplitude do sinal: V=16.917*R+319.832 (mV, 2-4<R<10 kHz)

- Sensor de rotação modelo Sen8G
  - Amplitude do sinal: V= 6*R (mV)

- Bobina de ignição modelo BAE504
  - Tempo de carga: 25 µs
  - Tempo de faísca: 1300 µs

- Bicos Injetores modelo IWM500
  - Tempo de abertura: 1000 µs
  - Tempo de fechamento: 54,25 µs

- Válvula Cannister modelo 213W8571
  - Freqüência de operação: 16 Hz

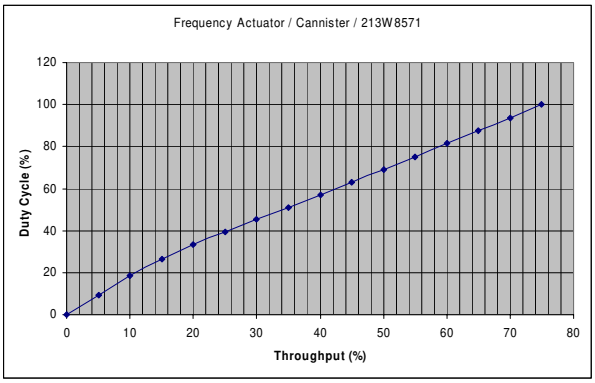


Figura 70: Cannister

- Motor de Passo modelo IBA
  - Número de passos: 214
  - Seqüência de fases: 0101, 1001, 1010, 0110

8.2 Resultados Obtidos

Devido à ausência de informações detalhadas sobre os algoritmos internos de cálculo da *centralina* utilizada, não pudemos realizar uma verificação funcional do sistema de forma quantitativa. Em uma verificação desse tipo, forneceríamos um conjunto de valores dos sensores para o qual conheceríamos a resposta exata prevista da *centralina* e poderíamos comparar nossos valores adquiridos de tempo de injeção, p.e., com o valor esperado.

Uma grande parte da informação que tínhamos disponível (alguns diagramas de bloco com a estrutura básica do Software e um programa de teste de fim de linha) também está disponível no CD que acompanha este trabalho. Como não conhecíamos nenhuma resposta exata prevista, tivemos que nos contentar com verificações qualitativas. Enumeramos aqui algumas delas:

1- Acionamento do relé de "*power latch*" (veja Figura 56: Conexão dos Relés Power Latch, Pump e Chave Key-On) –

AÇÃO: ao se acionar a chave de contato (em nosso caso a dupla chave *On-Off*), a *centralina* recebe a tensão de bateria pelo pino "PUMP" e "desperta".

REAÇÃO: Sua primeira ação é acionar o relé de "power latch" para garantir sua alimentação da tensão de bateria. Em seguida ela "puxa" o sinal do pino "PUMP" para massa o que aciona o relé de pré-pressurização da linha de combustível. Logo em seguida ela deveria comandar o motor de passo por alguns segundos para garantir que ele se encontra em fim de curso e calibrar sua posição inicial.

2- Entrada em modo de marcha lenta –

AÇÃO: borboleta em posição de abertura mínima E rotação inferior a valor de calibração

REAÇÃO: motor de passo é acionado para permitir entrada de ar. A posição final do motor de passo depende do valor da pressão do ar no coletor de admissão.

3- Entrada em modo de plena carga –

AÇÃO: borboleta toda aberta OU pressão de coletor superior a um limite de calibração

REAÇÃO: tempo de injeção máximo (permitido para a rotação atual)

4- Tempo de abertura e fechamento dos bicos – medição simples

5- Entrada em modo de desaceleração –

AÇÃO: rotação superior a valor de calibração E transitório negativo da borboleta

REAÇÃO: redução gradativa do tempo de injeção (freio motor)

6- Saída de modo de desaceleração –

AÇÃO: rotação inferior a valor de calibração E desaceleração por mais de n PMS (valor de calibração) E empobrecimento da mistura menor que limite de calibração

REAÇÃO: redução gradativa do tempo de injeção (freio motor)

7- Entrada em modo de *Cut-Off* –
AÇÃO: borboleta em mínimo por mais de m PMS (v. calib.) E T_{H2O} superior a um limite de calibração E rotação superior a um limite de calibração E rotação retornou maior que limite de calibração após o último *Cut-Off*
REAÇÃO: tempo de injeção reduzido a zero (freio motor)

Também convém mencionar aqui algumas das limitações encontradas com os equipamentos utilizados. De todas as informações relevantes sobre as cargas controladas pela *centralina*, a única não adquirida de forma satisfatória foi o ângulo de avanço da ignição. Essa informação requer a sincronização da aquisição e o armazenamento do tempo de ocorrência de alguns eventos em diversos sinais. No entanto, todas as placas DAQ de custo acessível só permitem a configuração de uma única condição de *trigger* para todos os canais a serem adquiridos, característica também comum à maioria dos osciloscópios disponíveis no mercado. Algumas abordagens tentadas foram o uso de ocorrências DAQ e retirada condicional de dados do *buffer* de aquisição, com diversos graus de sucesso. No entanto preferimos omitir a apresentação dessa informação até que possamos garantir sua acuidade na próxima versão do simulador.

Outra limitação do simulador é o fato de só podermos gerar o *quadro segnali* para 16 valores possíveis de rotação do motor. Essa limitação deve-se à taxa de atualização das placas de saídas analógicas ser limitada a submúltiplos de seu clock interno. Como também não poderíamos alterar o tamanho do *buffer* de geração *on-the-fly*, sob pena de interromper a geração do sinal de rotação por um tempo inaceitável para a *centralina*, determinamos um tamanho de *buffer* ótimo para satisfazer os seguintes requisitos, por ordem de prioridade:

- Taxa de atualização que garantisse precisão na forma de onda gerada, para todo o alcance de rotações utilizado pela *centralina* (aprox. 400-7000 RPM);
- Tamanho mínimo para preencher o *buffer* em poucas iterações quando devemos variar o sinal simulado, pois não podemos reescrevê-lo de uma vez durante a geração do mesmo, nem chavear *buffers* (limitação das placas de custo acessível).
- Tamanho grande o suficiente para permitir o maior número possível de rotações, cujas formas de onda de quadro segnali possuam comprimento submúltiplo desse tamanho.

Ambos os problemas devem ser solucionados na próxima versão do simulador, que irá substituir a placa AT-AO-06 pela PC-TIO-10. Esta nova placa é somente digital, possuindo 10 contadores/temporizadores e 8 linhas de entrada/saída, e será usada para gerar o sinal de rotação com precisão de décimos de RPM e sinais para sincronizar a aquisição dos Atuadores de Potência, implementando a medição do ângulo de avanço. Tais abordagens já foram exaustivamente estudadas e foram consideradas viáveis e satisfatórias pelo cliente e pelo fornecedor dos equipamentos.

Com esses resultados concluímos que a versão bancada do sistema simulador foi concluída com êxito para os seus propósitos de acompanhar um trabalho de Mestrado: demonstrar a utilização de novas tecnologias para sistemas de aquisição de dados e o desenvolvimento de uma metodologia para a execução em linguagem de programação gráfica de um sistema modelado com técnicas de orientação a objeto.

Capítulo 9 – Conclusões

É com muito prazer que temos hoje a confirmação das principais decisões de projeto tomadas, pois nesse meio-tempo a linguagem UML e a utilização de placas DAQ se estabeleceram definitivamente em todos os sistemas de teste utilizados pela indústria automotiva mundial.

Outras decisões confirmadas foram algumas das nossas técnicas de programação desenvolvidas para LabVIEW 4.1, como por exemplo:

- a metodologia usada na "tradução" de diagramas de classe em código LabVIEW – hoje o fabricante do mesmo já disponibiliza um pacote de ferramentas ("*State Diagram Toolkit*", disponível apenas a partir do LabVIEW 7.0) para geração automática de código a partir de um diagrama de estados. O código gerado apresenta a mesma estrutura que descrevemos no Item 5.4.
- novas funções com conceitos de orientação a objeto – a partir do LabVIEW6i foram disponiblizados os chamados "*control references*" e "*property nodes*" aumentando as possibilidades de encapsulamento de código e poliformismo.
- a adaptação do LabVIEW para programação orientada a objeto – uma empresa sueca desenvolveu um pacote de ferramentas ("*GOOP Inheritance Toolkit*") que provê suporte para todas as características de programação orientada a objeto; declarações formais de classe, criação e destruição de objetos, relações de herança, etc. Um tutorial e material informativo estão disponíveis no CD que acompanha essa dissertação.
- a adaptação do LabVIEW para programação orientada a objeto – outra empresa alemã desenvolveu outro pacote de ferramentas ("ObjectVIEW") que poderia ser considerado algo como um G++ (G, para linguagem gráfica). No momento ele só está disponível em alemão, mesmo assim incluímos material informativo no CD que acompanha essa dissertação.

9.1 Orientação para Futuras Expansões do Projeto

Nossa idéia inicial era expandir o sistema para possibilitar também testes de outros componentes, em especial chegamos a preparar protótipos para instrumentos combinados e sistemas de alarme por ultra-som. Mas o que em muitos momentos foi um inconveniente agora passa a ser uma grande vantagem. O grande lapso de tempo entre a execução prática desse projeto em '97 – '98 e a defesa desta tese em 2004 por motivos profissionais, nos proporciona uma perspectiva histórica que facilita em muito a análise que se segue.

Falando um pouco do presente, a Figura 71 abaixo mostra o crescimento do número de unidades eletrônicas conectadas em rede através de diversos barramentos (CAN em 100 e 500kbits/s, LIN, etc.) em veículos Volkswagen. O aumento da complexidade desses sistemas eletrônicos automotivos resultou em uma nova definição das competências essenciais para as montadoras e os fornecedores de componentes eletrônicos.

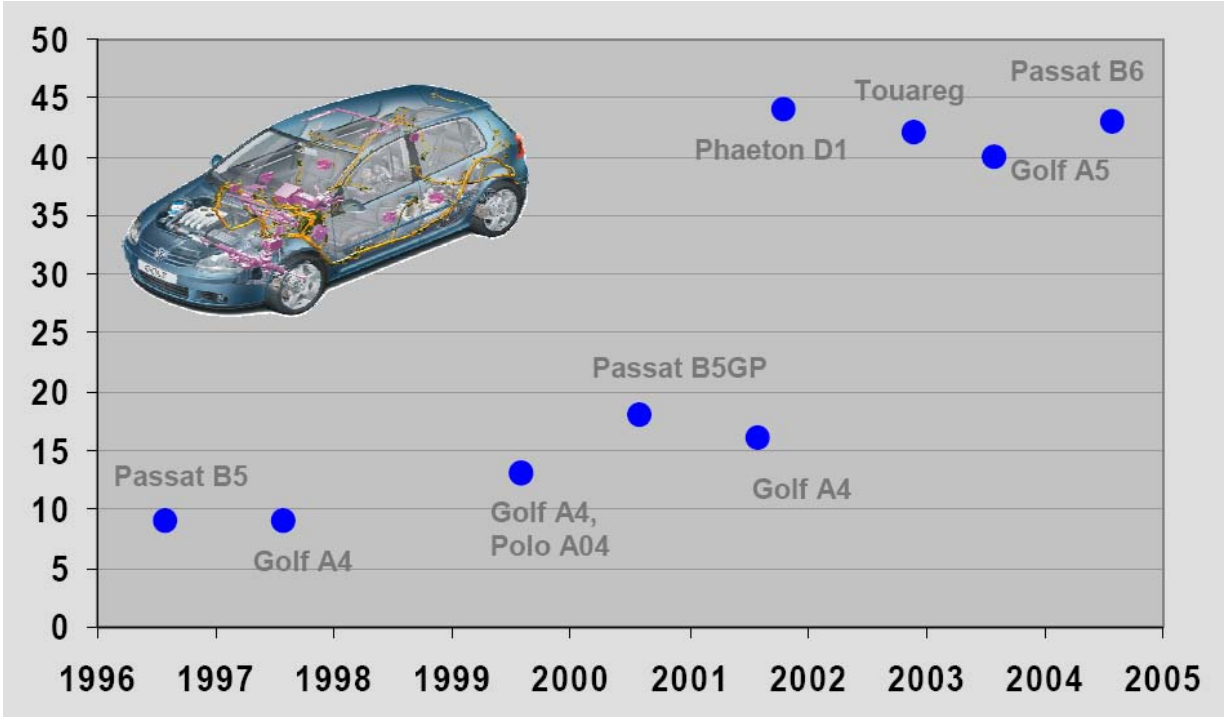


Figura 71: Unidades eletrônicas conectadas em rede[21]

O processo de desenvolvimento utilizado hoje em dia pela Volkswagen se baseia no chamado “Modelo V” que pode ser visto na Figura 72 abaixo.

Esse modelo descreve os passos incrementais que devem ser tomados na definição de sistemas eletrônicos complexos e a dualidade que existe entre especificação e correspondente verificação, em diferentes níveis de abstração. Na prática, ele representa a divisão de responsabilidades e tarefas entre os departamentos de engenharia da montadora e os fornecedores. Vale também ressaltar a similaridade entre esse modelo e o escalonamento de níveis de testes mencionado no Item 1.2.

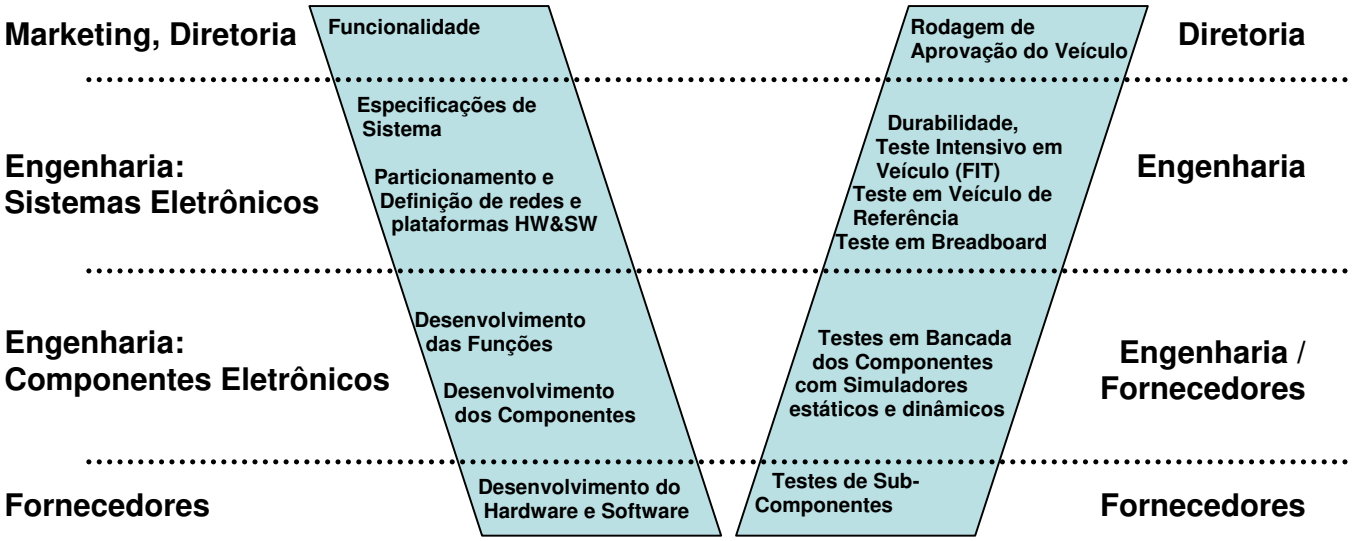


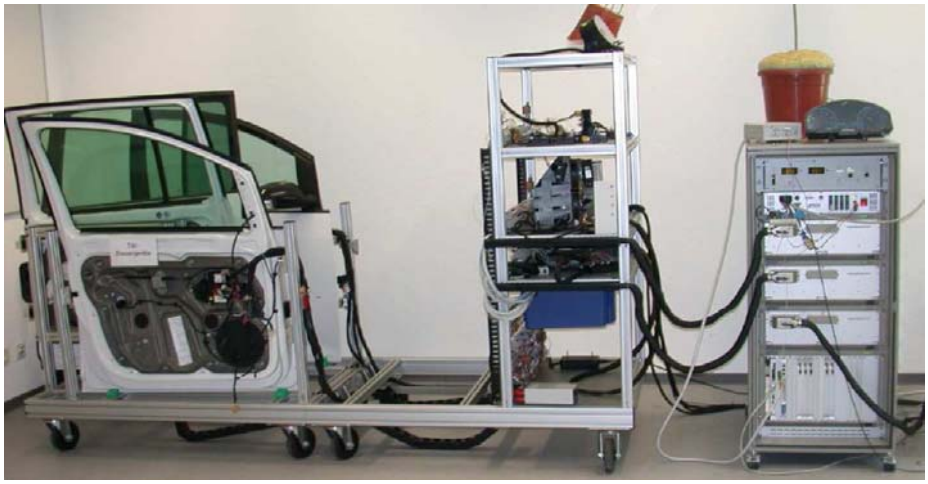
Figura 72: Modelo V

Sobre os testes de sub-componentes falaremos um pouco mais no final desse trabalho. Os testes em bancada visam garantir que cada componente individualmente está livre de falhas (Figura 73).

Os testes em *Breadboard* verificam a comunicação e interação entre unidades de controle, para distintas configurações de rede (Figura 74). Um de seus resultados é a geração de uma lista com as versões de hardware e software “liberadas” para cada componente, antes da montagem de protótipos de veículos completos.

Os testes em Veículo de Referência verificam o comportamento das redes em um ambiente veicular real (Figuras 75 e 76), onde a influência da bateria e do cabeamento, aterramento, corrente de *stand-by* etc. também estão presentes. Mesmo aqui são utilizados freqüentemente simuladores para os sinais provenientes dos sistemas de motor e chassis.

Por fim, são realizados ainda testes de durabilidade e o teste FIT que verificam a funcionalidade do sistema sob o ponto de vista do cliente.



Figuras 73 e 74: Bancada de testes para componentes e *Breadboard* para sistemas de conforto[21]

Figuras 75 e 76: Veículo de Referência[21]

O objetivo dessa discussão foi ressaltar melhor o papel que nosso sistema simulador desempenha no ambiente automotivo atual. Para poder desempenhar sua função de garantir que cada componente individualmente está livre de falhas, temos dois tipos de necesidades em aberto: mais recursos para simular os distintos ambientes aonde esse componente será montado; mais recursos para simular novos sensores/atuadores.

Com referência ao primeiro tipo de necessidade, identificamos como recurso absolutamente necessário à incorporação de interfaces para diversos tipos de barramentos automotivos: CAN (alta e baixa velocidade) é indispensável; LIN é altamente desejável; Flex Ray e TTP são recomendáveis para garantir a utilização do sistema a médio-prazo; MOST e Fire Wire se tornam interessantes caso haja interesse em expandir o sistema para teste de sistemas de *Infotainment*.

Para simular distintos ambientes e situações, temos necessidade de expandir o módulo de geração de scripts e percursos para atender a distintos tipos de testes. Especificamos aqui os tipos de teste desejados e suas respectivas funcionalidades:

- Automático
  - ⇒ Dado um set de dados adquiridos em campo, o usuário seleciona um trecho do percurso descrito e inicia o teste. O sistema simulará o comportamento do veículo para o dispositivo que está sendo testado, fornecendo-lhe os sinais correspondentes em todos os pinos que requisitam entrada de dados. Simultaneamente, ele irá também adquirir os sinais fornecidos pelo dispositivo e armazená-los em arquivo. Paralelamente a essa transferência de dados através da pinagem do dispositivo via sinais elétricos não tratados, alguns dispositivos provém a possibilidade de se obter os valores realmente compreendidos e/ou pretendidos pelo dispositivo via serial. Isto é, o sistema fará a monitoração dos dados enviados e verificação da validade dos dados recebidos consultando diretamente os registradores internos do dispositivo periodicamente. O usuário escolhe quais sinais e como quer visualisá-los: p.e. duty cycle; gráfico real time com os sinais esperado, adquirido e/ou pretendido; etc. Após o término do teste, o sistema avalia a discrepância da resposta de cada sinal e reporta ao usuário os sinais que violaram uma determinada tolerância predefinida. Usuários das áreas de Qualidade e Assistência Técnica podem personalizar seus report files a fim de evidenciar algum comportamento repetitivo e anômalo de algum(ns) sinal(is). Dependendo da discrepância de cada sinal o sistema também proverá um diagnóstico simplificado e sugestões de soluções básicas para problemas simples.
- Semi-Automático
  - ⇒ O usuário deverá definir um estado inicial manualmente ou escolhido de um set de dados. O sistema proverá uma interface com visualização de todos os parâmetros modificáveis, que estarão sendo enviados continuamente ao dispositivo em teste. O sistema procederá à atualização dos dados em dois possíveis modos: com ou sem realimentação. O primeiro efetuará cálculos de correlação entre variações nos diferentes parâmetros de entrada, a fim de determinar o próximo estado válido para o motor associado ao dispositivo. A determinação destas funções de correlação é configurável pelo usuário para permitir sua adaptação a vários motores e atualização à medida que modelamentos mais precisos forem sendo desenvolvidos. O segundo é apenas uma implementação dos testes paramétricos realizados atualmente. O sistema reportará apenas a comparação entre os sinais enviado e compreendido, visto que o usuário varia os parâmetros buscando algum comportamento particular do dispositivo e não existe uma resposta padrão esperada.

- End-Of-Line (EOL)
  ⇒ O usuário pode cadastrar uma seqüência de testes EOL e associá-los a cada dispositivo. Para isso é necessário especificar a string de ativação do modo EOL do dispositivo (se existir). No entanto, devido a limitações de hardware, sempre haverão alguns testes executados na linha de montagem que não poderão ser realizados (sobretensão, etc.). Isso não é considerado um problema, visto que o teste EOL é apenas mais uma ferramenta que se aproveita dos recursos do sistema e não sua finalidade principal.

Para falar do segundo tipo de necessidade (novos sensores/atuadores), devemos observar um pouco os últimos desenvolvimentos na área de motores. A Figura 77 mostra um panorama dos desenvolvimentos que já estão em andamento na Siemens/VDO para cumprimento dos níveis de emissões máximo prescritos para EUA e Europa nos próximos anos.

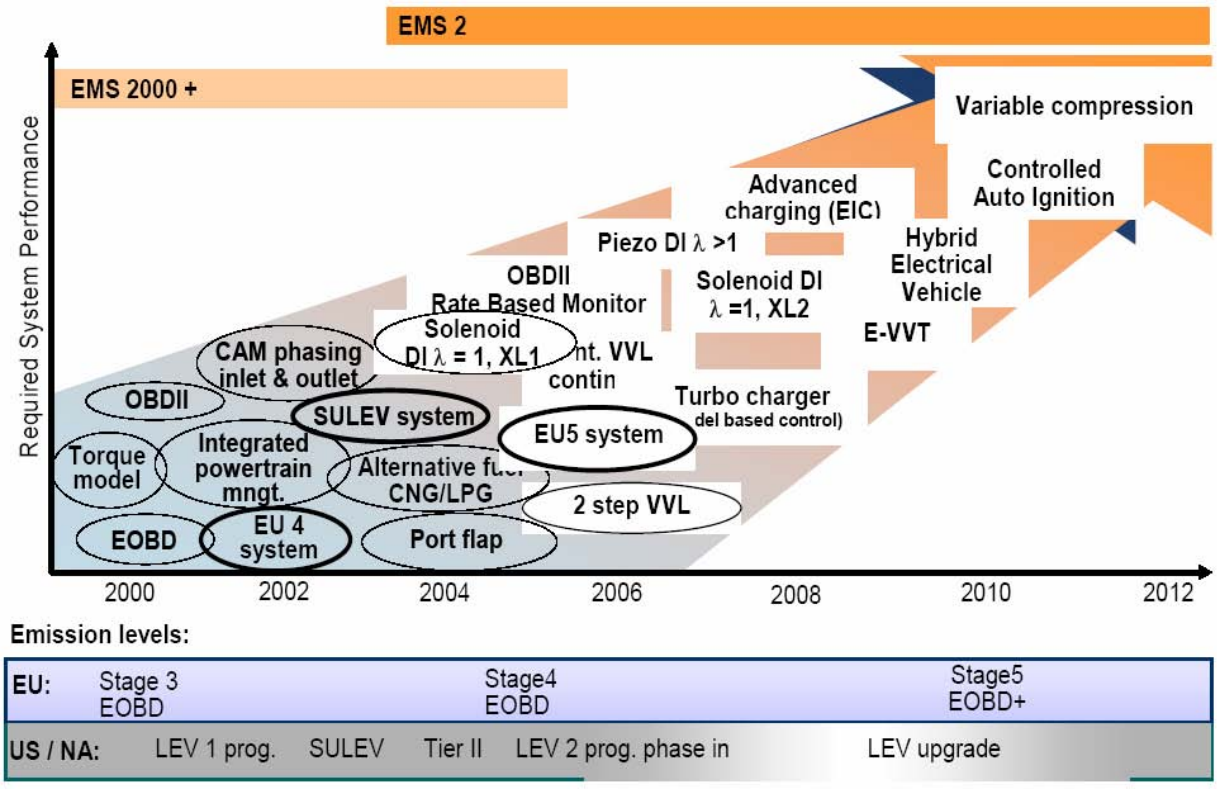


Figura 77: Níveis de emissão nos EUA/Europa e tecnologias para esse fim[22]

Para sermos mais precisos, a Figura 78 abaixo mostra componentes necessários para atingir níveis PZEV (partial zero-emission vehicle) e SULEV (super-ultra-low-emission vehicle). Nossa conclusão ao observar esse sistema é que se torna necessário concentrar futuros desenvolvimentos de nosso simulador em uma única direção. A incrível variedade de tipos de sensores e atuadores torna virtualmente impossível conceber algo que seja adaptável a qualquer tipo de sistema automotivo.

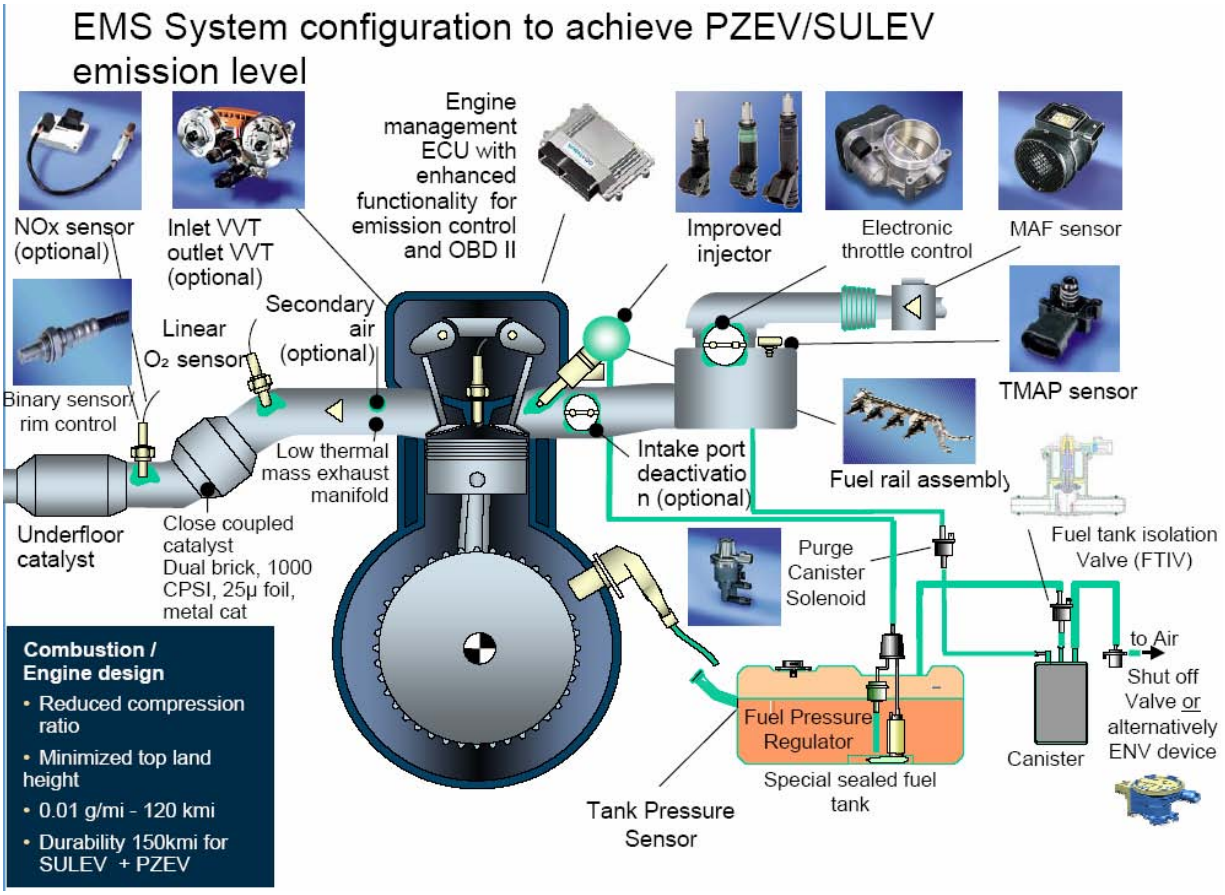


Figura 78: Componentes típicos de sistemas PZEV & SULEV[22]

Finalmente podemos mencionar algo sobre os testes de sub-componentes mencionados na Figura 72 como atividades sob responsabilidade dos fornecedores. A indústria automobilística vive hoje em ambiente de extrema pressão para redução dos tempos e custos de desenvolvimento de novos produtos.

A única solução possível para atender a esses objetivos aparentemente divergentes, tendo em vista ainda o aumento da complexidade já mencionado, é uma maior padronização dos sistemas utilizados.

Essa solução já está em andamento e se chama AutOSAr (Automotive Open System Architecture). Não iremos entrar em mais detalhes além do que se pode ver na Figura 79 abaixo, mas a mensagem é que não devemos nos concentrar no teste de subcomponentes de hardware e software porque no futuro os fornecedores de componentes automotivos já os receberão pré-especificados e validados.

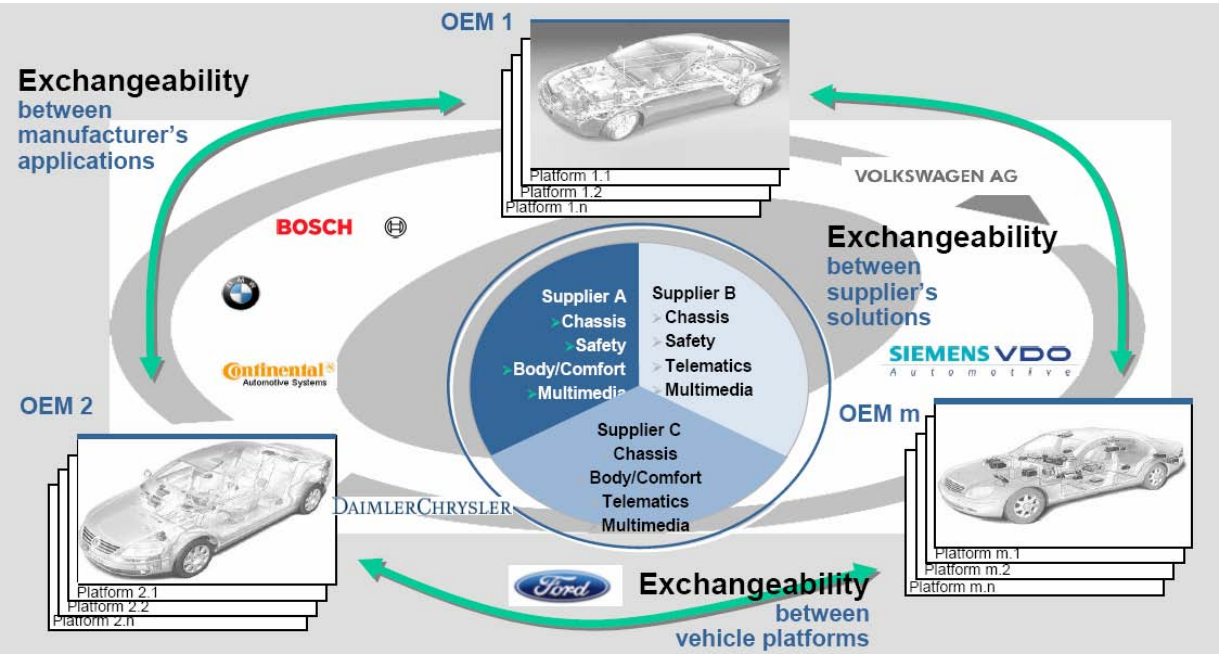


Figura 79: Autosar[21]

Acrescentando alguns pontos já mencionados no decorrer deste trabalho, resumimos por fim nossa recomendação para as futuras expansões do sistema nos seguintes pontos:

- Prover interfaces para redes automotivas CAN. LIN, etc.
- Expandir e melhorar o módulo de scripts para permitir diferentes tipos de testes automatizados
- Expandir e padronizar as formas de documentação dos resultados dos testes
- Priorizar oficialmente as *centralinas* como foco de futuros desenvolvimentos e adquirir recursos de hardware para simular os sinais necessários para os futuros motores em questão
- Facilitar e automatizar a instalação do sistema
- Importar o código para a última versão do LabVIEW disponível – o sistema foi criado na versão 4.1 e hoje já está disponível a versão 7.0

9.2 Trabalhos Acadêmicos

Como resultado deste trabalho foram apresentados os seguintes *papers* em eventos internacionais:

- "Automotive Simulator for Electronic Fuel Injections" – SAEBrasil99 / São Paulo
- "Low-Cost Virtual-Instrumentation-Based Simulator for Electronic Fuel Injections" – ISATA2000 / Dublin

Referências Bibliográficas

[1] Butler, K. R. and Wagner, J. R., A Strategy to Demonstrate the Compliance of Automotive Controller Software to Systems Requirements Using Simulation Technology, SAE Transactions – Journal of Engines, Vol.103, Section 3, pp. 776-786

[2] S. Alles, C. Swick, M. Hoffman, S. Mahmud and F. Lin (1995) The Hardware Design of a Real Time HITL for Traction Assist Simulation, IEEE Transactions on Vehicular Technology, Vol. 44, N0. 3, pp. 668-682

[3] IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE Standard 610.12, 1990

[4] ISO 11452-1...5, Prüfverfahren für Komponenten;

mais especialmente a emissão conjunta DIN ISO 11452-4 (Ausgabe: 2000-03)

e ainda ISO 7637-1, Methods of test

e DIN 40 839-3, Elektromagnetische Verträglichkeit (EMV) in Straßenfahrzeugen

[5] J. Wagner, R. Brunts, K. Kaster, D. Eagan and D. Anthony, (1999) A Vision for Automotive Electronics Hardware-In-The-Loop Testing, International Journal of Vehicle Design, Vol. 22, Nos. 1-2, pp. 14-28

[6] Luis Alejandro Cortés y Antonio García, Proceso de Codiseño Hardware-Software, IV Workshop Iberchip, Mar del Plata – Argentina, Março 1998

[7] Giovanni de Micheli, Computer-Aided Hardware-Software Codesign, IEEE Micro Vol.14 Nº 4, Agosto 1994

[8] A. Kalavade and E. A. Lee, A Hardware-Software Codesign Methodology for DSP Applications, IEEE Design & Test, Vol. 10 Nº 3, pp. 16-28, September 1993

[9] J. K. Adams, D. E. Thomas, The Design of Mixed Hardware/Software Systems, 33rd Design Automation Conference, 1996

[10] K. Newton, W. Steeds and T. K. Garret, The Motor Vehicle, Butterworths, 11th ed. 1989

[11] F. F. Schmidt, translated by R. W. Stuart Mitchell and J. Horne, The Internal Combustion Engine, Chapman and Hall, 1965

[12] Automotive Engineering, Vol.9 Nº 5, Outubro 1984

[13] Automotive Handbook, Robert Bosch GmbH, 4th ed. October 1996

[14] Sistemas de Injeção e Ignição Eletrônicas, ISVOR-FIAT, Agosto 1991

[15] Wagner, J.R. and Furry, J. S. (1992), A Real Time Simulation Environment for the Verification of Automotive Electronic Controller Software, International Journal of Vehicle Design, Vol. 13, No. 4, pp. 365-377

[16] Data acquisition Fundamentals, Signal Conditioning Fundamentals for PC-Based Data acquisition systems & Field Wiring and Noise Considerations for Analog Signals, AN 007, 025 & 048, available at http://www.natinst.com/appnotes.nsf, National Instruments, 1998

[17] James Rumbaugh et al, Object-Oriented Modeling and Design, Prentice Hall, 1990

[18] Derek Coleman et al, Object-Oriented Development: The Fusion Method, Prentice Hall, 1994

[19] Ivar Jacobson et al, Object-Oriented Software Engineering, Addison-Wesley, 1992

[20] UML Summary, UML Notation Guide, UML Semantics & UML Extensions for Objectory Process for Software Engineering, Documentation Version1.1 from UML Resources, available at http://www.rational.com/uml, updated September 1997

[21] Dr. Karl-Thomas Neumann, Prozesse für die Software Entwicklung im Volkswagen Konzern, Kongress „Fortschritte in der Automobil-Elektronik“, Ludwigsburg 2004

[22] Dr. Klaus Egger (Siemens VDO), Trends and Potentials for future Developments for Engine and Powertrain, „Fortschritte in der Automobil-Elektronik“, Ludwigsburg 2004

Apêndice A – Alguns Use Cases

USE CASE 1	Modificar_Produto	
Goal in Context	Configurador deseja alterar os componentes associados a um produto da Base de Dados	
Scope & Level	Configuração de Produto, Subfunção	
Preconditions	Produto existe na Base de Dados	
Success End Condition	Produto é alterado na Base de Dados, conforme informações fornecidas pelo Configurador	
Failed End Condition	Produto permanece inalterado na Base de Dados	
Primary, Secondary Actors	Configurador	
Trigger	Requisição de modificação em produto	
DESCRIPTION	**Step**	**Action**
	1	O Configurador requisita modificação em produto.
	2	O sistema adquire as informações relativas ao produto da Base de Dados.
	3	O sistema apresenta uma tabela, associando cada pino do conector do produto a um componente, e um campo com a interface de testes do produto.
	4	O Configurador pode associar novos, remover ou substituir componentes para cada pino; bem como associar nova interface de testes.
	5	O Configurador confirma a modificação.
	6	O sistema fornece a lista de conexões do produto com o AutoSEFI.
EXTENSIONS	**Step**	**Branching Action**
	5a	Não há interface de testes associada ao produto: 5a1. O sistema retorna ao passo 4.
	5b	O Configurador desiste da modificação: 5b1. O sistema não altera as informações do produto na Base de Dados.
VARIATIONS		**Branching Action**
	1	O sistema criou um novo produto por requisição do Configurador
	6	O Configurador pode visualizar ou imprimir a lista.

RELATED INFORMATION	1. Modificar_Produto
Priority:	low
Performance	10 min
Frequency	1/mês
Channels to actors	interativo
OPEN ISSUES	Definir atributos do produto Conferir recursos do AutoSEFI
Due Date	13/01/1999
...any other management information...	
Superordinates	Configurar_Produtos (#4), Adicionar_Produto (#2)
Subordinates	Selecionar_Parte (#3),

USE CASE 2	Adicionar_Produto	
Goal in Context	Configurador deseja acrescentar um novo produto à Base de Dados	
Scope & Level	Configuração de Produto, Subfunção	
Preconditions		
Success End Condition	Novo Produto é acrescentado à Base de Dados, conforme informações fornecidas pelo Configurador	
Failed End Condition	Base de Dados permanece inalterada	
Primary, Secondary Actors	Configurador	
Trigger	Requisição de criação de novo produto	
DESCRIPTION	**Step**	**Action**
	1	O Configurador requisita criação de novo produto
	2	O Configurador fornece nome e pinagem do novo produto
	3	O sistema cria um novo produto, sem nenhum componente nem interface de testes associados
	4	O Configurador modifica esse novo produto
EXTENSIONS	**Step**	**Branching Action**
	4a	O Configurador não modificou o novo produto: 4a1. O sistema remove o novo produto da Base de Dados.

RELATED INFORMATION	2. Adicionar_Produto
Priority:	Low
Performance	10 min
Frequency	1/mês
Channels to actors	Interativo
OPEN ISSUES	Apresentação da Pinagem
Due Date	05/03/1999
...any other management information...	
Superordinates	Configurar_Produtos (#4)
Subordinates	Selecionar_Parte (#3)

USE CASE 11	Executar Simulação em Produto	
Goal in Context	Técnico deseja simular um ambiente automotivo para um produto da Base de Dados	
Scope & Level	Resumo	
Preconditions	Produto existe na Base de Dados	
Success End Condition	A simulação foi executada e os resultados foram apresentados em forma visual ou impressa.	
Failed End Condition	Nao ocorreu simulação ou os resutados não foram apresentados ao Técnico	
Primary, Secondary Actors	Técnico	
Trigger	Requisição de Execução de Simulação	
DESCRIPTION	**Step**	**Action**
Inicialização	1	O Técnico se identifica para o sistema
	2	O Técnico seleciona um produto e especifica um tipo de simulação.
Configuração	3	O sistema adquire as informações sobre o produto da base de dados.
	4	O sistema configura as placas DAQ e inicializa a interface visual para o modo Simulação.
Simulação	5	O sistema apresenta o Painel Frontal para o Técnico.
Simulação (cont.)	6	O sistema simula um ambiente correspondente aos instantes que se seguem à partida de um veículo, fazendo com que o produto atue como se estivesse em condições normais de operação.
	7	O sistema executa concorrentemente os seguintes processos:
	7.1	O sistema atualiza as informações relativas ao estado atual das cargas controladas pelo produto e as disponibiliza na Área de Transferência.
	7.2	O Painel Frontal apresenta as informações presentes na Área de Transferência nos mostradores e gráficos e disponibiliza o estado atual dos controles e interruptores na Área de Transferência.
	7.3	O sistema checa o estado dos controles e interruptores presente na Área de Transferência e simula esse ambiente para o produto.
	7.4	O Técnico pode, a qualquer momento, requisitar ao sistema a mudança para os modos Hold Outputs e Run Script.

	7.5	O Técnico pode, a qualquer momento, interromper a simulação através de interruptores representando os eventos de parada do motor e Key-off.
Histórico	8	O Técnico requisita ao sistema informações sobre a simulação.
	9	O Técnico termina a execução do sistema.
EXTENSIONS	**Step**	**Branching Action**
Hold Outputs	7a	O sistema armazena o estado atual dos controles e interruptores e simula continuamente esse ambiente para o produto.
		O Técnico pode alterar os valores de todos os controles, respeitando seus limites de variação instantânea, sem influir no ambiente que está sendo simulado para o produto.
		Após executar as modificações desejadas repesentando o novo ambiente para o produto, o Técnico solicita ao sistema o retorno para o modo Simulação.
Run Script	7b	O sistema armazena o estado atual dos controles e interruptores e simula continuamente esse ambiente para o produto.
		O sistema desabilita o Painel Frontal e apresenta a interface de seleção de scripts para o Técnico.
		O Técnico seleciona o script a ser simulado dentre os disponíveis na base de dados, ou edita um novo script, podendo armazená-lo na base de dados.
		O sistema apresenta o Painel Frontal, com os controles e interruptores desabilitados.
		O sistema executa o script, simulando o percurso (ambiente variante no tempo) representado no mesmo.
		O Técnico pode selecionar um novo script, retornar ao modo Simulação ou interromper a simulação e mudar para o modo Histórico.
VARIATIONS		**Branching Action**
	8	O Técnico decide se esses resultados devem ser armazenados na base de dados e/ou apresentados em forma visual ou impressa.
	9	O Técnico pode executar uma nova simulação.

RELATED INFORMATION	11. Executar Simulação em Produto
Priority:	high
Performance	muitas horas
Frequency	várias vezes por dia
Channels to actors	Interativo
OPEN ISSUES	Definir formato de script
Due Date	13/01/1999
...any other management information...	
Superordinates	
Subordinates	Checar Controles e Interruptores (#12)

USE CASE 12	Checar Controles e Interruptores	
Goal in Context	O sistema executa as ações tomadas pelo Técnico durante uma simulação	
Scope & Level	Simulação de Produto, Subfunção	
Preconditions	Uma simulação está em andamento	
Success End Condition	O sistema compreendeu as ações tomadas pelo Técnico e reagiu de acordo.	
Failed End Condition	As ações tomadas pelo Técnico foram ignoradas pelo sistema.	
Primary, Secondary Actors	Técnico	
Trigger	Simulação foi iniciada	
DESCRIPTION	**Step**	**Action**
	1	O sistema executa concorrentemente os processos de 2 a 4:
Sensores de Frequência	2.1	O Gerador de Sinais Frequenciais requisita para a Área de Transferência um Container de Informações com o estado dos controles e interruptores referentes aos Sensores de Frequência.
	2.2	O Gerador de Sinais Frequenciais cria as formas de onda correspondentes às informações recebidas.
	2.3	O Gerador de Sinais Frequenciais as reproduz nos canais de saída analógica apropriados.
Transdutores	3.1	O Gerador de Tensões requisita para a Área de Transferência um Container de Informações com o estado dos controles e interruptores referentes aos Transdutores.
	2.2	O Gerador de Tensões reproduz as informações recebidas nos canais de saída analógica apropriados.
Sonda Lambda	3.1	O Gerador de Sonda Lambda requisita para a Área de Transferência um Container de Informações com o estado do controle referente ao valor da sonda.
	3.2	O Gerador de Sonda Lambda calcula o duty cycle e a frequência do sinal de sonda lambda correspondentes às informações recebidas, para simula uma variação realimentada entre o valor recebido e lambda unitário.
	3.3	O Gerador de Sonda Lambda reproduz o sinal nos canais de saída analógica apropriados.

Entradas On-Off	4.1	O Gerador de Sinais On-Off requisita para a Área de Transferência um Container de Informações com o estado dos controles referentes às Entradas On-Off e aos interruptores referentes às cargas.
	4.2	O Gerador de Sinais On-Off aciona as saídas digitais correspondentes.
EXTENSIONS	**Step**	**Branching Action**
VARIATIONS		**Branching Action**

RELATED INFORMATION	12. Checar Controles e Interruptores
Priority:	high
Performance	muitas horas
Frequency	várias vezes por dia
Channels to actors	Interativo
OPEN ISSUES	
Due Date	13/01/1999
...any other management information...	
Superordinates	Executar Simulação em Produto (#11)
Subordinates	

USE CASE 13	Atualizar Mostradores e Gráficos	
Goal in Context	O sistema verifica as respostas do produto para o qual está sendo executada uma simulação e as disponibiliza na Área de Transferência.	
Scope & Level	Simulação de Produto, Subfunção	
Preconditions	Uma simulação está em andamento	
Success End Condition	O sistema compreendeu as respostas do produto e as disponibilizou na Área de Transferência.	
Failed End Condition	O estado dos mostradores e gráficos na Área de Transferência não corresponde ao estado atual do produto.	
Primary, Secondary Actors	Técnico	
Trigger	Simulação foi iniciada	
DESCRIPTION	**Step**	**Action**
	1	O sistema executa concorrentemente os processos de 2 a 4:
Atuadores de Potência I (Injetores)	2.1	O sistema adquire dos canais de entrada analógica apropriados a forma de onda da corrente no Atuador de Potência selecionado pelo Técnico no Painel Frontal.
	2.2	O sistema disponibiliza os dados adquiridos na Área de Transferência.
Atuadores de Potência II (Ignição)	3.1	O sistema adquire dos canais de entrada analógica apropriados a forma de onda da corrente no Atuador de Potência e extrai as informações sobre o Tempo de Dwell e o Ângulo de Avanço da ignição.
	3.2	O sistema disponibiliza os dados adquiridos na Área de Transferência.
Atuador de Marcha Lenta	4.1	O sistema adquire os sinais das fases e checa sua polaridade.
	4.2	O sistema acompanha duas fases do Atuador de Marcha Lenta a fim de determinar sua posição exata e a disponibiliza na Área de Transferência.
Atuadores em Frequência	5.1	O sistema adquire a frequência e o duty cycle dos sinais frequenciais provenientes dos Atuadores em Frequência.
	5.2	O sistema disponibiliza essas informações na Área de Transferência.
Saídas On-Off	6	O sistema adquire os estados das Saídas On-Off e os disponibiliza na Área de Transferência.

Sincronismo	7	O sistema utiliza o sinal correspondente ao "Quadro Segnali" a fim de estabelecer uma referência temporal para a simulação.
EXTENSIONS	**Step**	**Branching Action**
VARIATIONS		**Branching Action**

RELATED INFORMATION	13. Atualizar Mostradores e Gráficos
Priority:	high
Performance	muitas horas
Frequency	várias vezes por dia
Channels to actors	Interativo
OPEN ISSUES	
Due Date	13/01/1999
...any other management information...	
Superordinates	Executar Simulação em Produto (#11)
Subordinates	

USE CASE 14	Apresentar no Painel Frontal	
Goal in Context	O sistema verifica as ações tomadas pelo Técnico durante uma simulação e as disponibiliza na Área de Transferência, bem como verifica na na Área de Transferência as respostas do produto para o qual está sendo executada uma simulação e as apresenta para o Técnico	
Scope & Level	Simulação de Produto, Subfunção	
Preconditions	Uma simulação está em andamento	
Success End Condition	O sistema apresentou o estado atual do produto para o Técnico e irá reagir de acordo as ações tomadas pelo Técnico.	
Failed End Condition	O estado dos mostradores e gráficos não corresponde ao estado atual do produto e/ou as ações tomadas pelo Técnico serão ignoradas pelo sistema.	
Primary, Secondary Actors	Técnico	
Trigger	Simulação foi iniciada	
DESCRIPTION	**Step**	**Action**
	1	O sistema fornece ao Painel Frontal as informações disponíveis na Área de Transferência.
	2	O Painel Frontal atualiza as informações nos mostradores e gráficos.
	3	O Painel Frontal checa requisições de mudança do modo de simulação e eventos de parada do motor e Key-off e as processa.
	4	O Painel Frontal checa a atuação do Técnico em todos os controles e interruptores e atualiza as informações da Área de Transferência.
EXTENSIONS	**Step**	**Branching Action**
VARIATIONS		**Branching Action**

RELATED INFORMATION	14. Apresentar no Painel Frontal
Priority:	high
Performance	muitas horas
Frequency	várias vezes por dia
Channels to actors	Interativo
Due Date	13/01/1999
Superordinates	Executar Simulação em Produto (#11)

USE CASE 15	Adquirir Informações sobre Produto	
Goal in Context	Disponibilizar na Área de Transferência as informações sobre o produto selecionado disponíveis na Base de Dados.	
Scope & Level	Simulação de Produto, Subfunção	
Preconditions	O Técnico selecionou um produto para uma simulação	
Success End Condition	As informações sobre o produto selecionado foram disponibilizadas na Área de Transferência.	
Failed End Condition	A Área de Transferência não contém informações sobre o produto selecionado.	
Primary, Secondary Actors	Técnico	
Trigger	Produto foi selecionado	
DESCRIPTION	**Step**	**Action**
	1	O sistema adquire as informações sobre o Produto selecionado pelo Técnico da base de dados.
	2	O sistema disponibiliza essas informações na Área de Transferência.
EXTENSIONS	**Step**	**Branching Action**
	1a	Não há informação suficiente associada ao produto para executar uma simulação: 1a1. O sistema solicita a seleção de um outro produto e volta ao passo 1.
VARIATIONS		**Branching Action**

USE CASE 16	Configurar as Placad DAQ	
Goal in Context	Preparar o sistema para uma simulação.	
Scope & Level	Simulação de Produto, Subfunção	
Preconditions	As informações sobre o produto selecionado estão disponíveis na Área de Transferência.	
Success End Condition	O sistema está preparado para início imediato de uma simulação	
Failed End Condition	O sistema não está preparado para início imediato de uma simulação	
Primary, Secondary Actors	Técnico	
Trigger	Produto foi selecionado	
DESCRIPTION	**Step**	**Action**
	1	O sistema configura canais de saída analógica de acordo com as informações sobre os Transdutores (em especial a Sonda Lambda) associados ao Produto presentes na Área de Transferência.
	2	O sistema configura canais de saída analógica de acordo com as informações sobre os Sensores de Frequência associados ao Produto presentes na Área de Transferência.
	3	O sistema configura canais de saída digital de acordo com as informações sobre, as Entradas On-Off e os interruptores referentes às cargas e aos eventos de parada do motor e Key-off, associadas ao Produto presentes na Área de Transferência.
	4	O sistema configura canais de entrada analógica de acordo com as informações sobre os Atuadores de Potência (em especial a Ignição) associados ao Produto presentes na Área de Transferência.
	5	O sistema configura um canal de entrada analógica de acordo com as informações sobre o Quadro Segnale associado ao Produto presentes na Área de Transferência.
	6	O sistema configura canais de entrada frequencial de acordo com as informações sobre os Atuadores em Frequência associados ao Produto presentes na Área de Transferência.
	7	O sistema configura um canal de entrada

		frequencial e um canal de entrada digital de acordo com as informações sobre os Atuadores de Marcha Lenta associados ao Produto presentes na Área de Transferência.
	8	O sistema configura canais de entrada digital de acordo com as informações sobre as Saídas On-Off associadas ao Produto presentes na Área de Transferência.
	9	O sistema disponibiliza todas as informações de configuração na Área de Transferência.
EXTENSIONS	**Step**	**Branching Action**
	1a	
VARIATIONS		**Branching Action**

USE CASE 17	Inicializar a Interface Visual	
Goal in Context	Preparar o sistema para uma simulação.	
Scope & Level	Simulação de Produto, Subfunção	
Preconditions	As informações sobre a configuração do sistema para a simulação a ser iniciada estão disponíveis na Área de Transferência.	
Success End Condition	O Técnico pode começar a interagir com a interface visual de simulação que lhe foi apresentada.	
Failed End Condition	Não foi apresentada interface visual de simulação para o Técnico.	
Primary, Secondary Actors	Técnico	
Trigger	O sistema foi configurado para iniciar uma simulação	
DESCRIPTION	**Step**	**Action**
	1	O sistema requisita ao Painel Frontal a inicialização da interface visual. Esta interface possui: * controles correspondendo a todos os sinais de entrada requeridos pelo produto; * mostradores e gráficos correspondendo às informações relevantes sobre a atuação do produto sobre suas cargas; * interruptores com a capacidade de simular a desconexão de sensores e cargas.
	2	O Painel Frontal inicializa todos os controles, interruptores, mostradores e gráficos conforme as informações sobre o Produto selecionado pelo Técnico presentes na Área de Transferência.
EXTENSIONS	**Step**	**Branching Action**
	1a	
VARIATIONS		**Branching Action**

USE CASE 18	Disponibilizar Informações na Área de Transferência	
Goal in Context	Propiciar um mecanismo de troca de informações entre métodos concorrentes que compense a falta de sincronismo inerente ao processo.	
Scope & Level	Simulação de Produto, Subfunção	
Preconditions		
Success End Condition	O Container de Informações recebido está disponível para consulta.	
Failed End Condition	O Container de Informações recebido não está disponível para consulta.	
Primary, Secondary Actors	Técnico	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	A Área de Transferência recebe um Container de Informações.
	2	A Área de Transferência armazena o Container no buffer correspondente ao tipo de informação transportada.
	3	A Área de Transferência atualiza os registros sobre o histórico da simulação e de escrita do buffer.
EXTENSIONS	**Step**	**Branching Action**
	1a	
VARIATIONS		**Branching Action**

USE CASE 19	Checar Informações na Área de Transferência	
Goal in Context	Propiciar um mecanismo de troca de informações entre métodos concorrentes que compense a falta de sincronismo inerente ao processo.	
Scope & Level	Simulação de Produto, Subfunção	
Preconditions		
Success End Condition	O Container de Informações solicitado foi entregue.	
Failed End Condition	O Container de Informações solicitado não pode ser entregue.	
Primary, Secondary Actors	Técnico	
Trigger		
DESCRIPTION	**Step**	**Action**
	1	A Área de Transferência recebe uma requisição de informações.
	2	A Área de Transferência fornece um Container de Informações com as informações do tipo requisitado que foram armazenadas no buffer correspondente após a última requisição.
	3	A Área de Transferência atualiza os registros sobre o histórico de leitura do buffer.
EXTENSIONS	**Step**	**Branching Action**
	1a	
VARIATIONS		**Branching Action**

Apêndice B – Lista de VIs

Na biblioteca Corefiles.llb

About.vi

VI para mostrar a mensagem de About do sistema simulador

Front Panel Elements:

Control #1	(bool)	Ok	Close about screen
Control #2	(bool)	About?(F)	Selects between "about" or "loading" screens
Control #3	(string)	Version	Version number retrieved from the ini file
Control #4	(string)	Date	Version date retrieved from the ini file
Indicator #1	(string)	Version Info	Information regarding this version number and date
Indicator #2	(string)	Message	Loading message

AutoSEFI.vi

VI de inicialização, que lê os parâmetros do arquivo autosefi.ini, verifica se há erros e inicializa o Engine.VI

Front Panel Elements:

Engine.vi

VI que solicita a identificação do usuário e apresenta a tela inicial do sistema simulador, de onde se acessam os diferentes recursos de cadastro, simulacão, etc.

Front Panel Elements:

Control #1	(bool)	products	Edit the parts Database, modifying, adding or removing products
Control #2	(bool)	test	Run a test in a product registered in the Database
Control #3	(bool)	components	Edit the parts Database, modifying, adding or removing components
Control #4	(bool)	exit	Exit AutoSEFI
Control #5	(bool)	reports	Analyze reports retrieved from the test histories Database
Control #6	(path)	Base Path	Specifies the base path where AutoSEFI has been installed
Control #7	(bool)	about	About AutoSEFI

Inifiles.vi

VI desenvolvido por Eric R. Nelson, Ph.D, da empresa Valiant Technology, para edição de arquivos '.ini'

Front Panel Elements:

LogIn.vi

VI para verificação da identidade do usuário. Utiliza as informações cadastradas no arquivo encriptado Users.id

Front Panel Elements:

Control #1	(string)	Login	The login provided by the user in this session
Control #2	(string)	Password	The password for the user in this session
Control #3	(cluster)	User Model	A cluster containing the register information model for an user. It has the following fields:
Control #3.1	(string)	Name	Full name of the user
Control #3.2	(string)	Login	Nickname chosen by the user
Control #3.3	(string)	Password	Password associated with the login
Control #3.4	(array)	Privileges	An array of Actors representing wich packages may be used by the user. In the current version, AutoSEFI takes in count only the privileges of the first Actor in the array
Control #3.4.1	(enum)	Actor	The possible roles the user can play with the system. A role is specified through the access permission to a package of the system. Possible values for this version are {Technician, Engineer, Manager}
Control #4	(bool)	Ok	Validates user identity
Control #5	(bool)	Cancel	Cancel idendity validation
Control #6	(path)	Base Path	Specifies the base path where AutoSEFI has been installed
Control #7	(bool)	Administrator?(F)	Is this an administrator login for security issues?
Indicator #1	(array)	Privileges	An array of Actors representing wich packages may be used by the logged user. In the current version, AutoSEFI takes in count only the privileges of the first Actor in the array. It will contain a null elements array if the login is unsuccessful
Indicator #1.1	(enum)	Actor	The possible roles the user can play with the system. A role is specified through the access permission to a package of the system. Possible values for this version are {Technician, Engineer, Manager}
Indicator #2	(path)	Database Base Path	Specifies the base path where the Parts Database is stored

My General Error Handler.vi e My Simple Error Handler.vi

VIs derivados dos originais do LabVIEW 4.0 com pequenas alterações

Front Panel Elements:

Na biblioteca DataBaseManagement.llb

CADASTRO.VI

This VI selects a component type, interactively showing all the parameters associated with the type. Then it calls 'Compsel.VI' and if it returns a non-empty path, lets the user edits the component parameters and saves it

Front Panel Elements:

Control #1	(cluster)	Sensor I		A cluster containing the registration attributes of a Transducer
Control #1.1	(ring)	Sensor Type		Indicates the kind of transducer for this component. It is used by the system to organize the Database
Control #1.2	(string)	Model		The name to serve as a reference for the component in the Database
Control #1.3	(dbl)	Maximum Value		The maximum value (in Volts) this signal can assume
Control #1.4	(string)	Transfer Function		The transfer function that fits better the points in the Values Table.This information is not being used in the current version
Control #1.5	(array)	Values Table		A look-up table with the conversion of physical to voltage unities for this component
Control #2	(bool)	Ok		
Control #3	(cluster)	Sensor II		A cluster containing the registration attributes of a Frequency Sensor
Control #3.1	(ring)	Sensor Type		Indicates the kind of frequency sensor for this component. It is used by the system to organize the Database
Control #3.2	(string)	Model		The name to serve as a reference for the component in the Database
Control #3.3	(dbl)	Maximum Value		The maximum value (in Volts) this signal can assume
Control #3.4	(string)	Amplitude Function		The transfer function relating the amplitude with the frequency of the signal.This information is not being used in the current version
Control #4	(cluster)	Actuator II		A cluster containing the registration attributes of a Frequency Actuator
Control #4.1	(ring)	Actuator Type		Indicates the kind of frequency actuator for this component. It is used by the system to organize the Database
Control #4.2	(string)	Model		The name to serve as a reference for the component in the Database
Control #4.3	(dbl)	Operating Frequency		The operating frequency for the duty cycle of the component
Control #4.4	(array)	Values Table		A look-up table with the conversion of physical to duty cycle unities for this component
Control #5	(cluster)	Actuator I		A cluster containing the registration attributes of a Power Actuator
Control #5.1	(ring)	Actuator Type		Indicates the kind of power actuator for this component. It is used by the system to organize the Database
Control #5.2	(string)	Model		The name to serve as a reference for the component in the Database
Control #5.3	(dbl)	Opening/Rise		Time spent to open the injector or time to charge the

				Time (usec)	ignition coil
	Control #5.4	(dbl)		Closing/Spark Time (usec)	Time spent to close the injector or duration time of the spark produced by the ignition coil
Control #6	(cluster)	Idle Speed			A cluster containing the registration attributes of a Idle Speed
	Control #6.1	(ring)		Control Type	Indicates the kind of idle speed control for this component. It is used by the system to organize the Database
	Control #6.2	(string)		Model	The name to serve as a reference for the component in the Database
	Control #6.3	(dbl)		Number of Steps/Operating Frequency	The maximum number of steps for Stepper Motors or the operating frequency of the duty cycle for DC Motors
	Control #6.4	(array)		Forward Step Phase Sequence	A table containing the valid ordered binary numbers representing the phase signals of a stepper motor
Control #7	(bool)	cancel			
Control #8	(bool)	Extend			
Control #9	(ring)	Type			The type of component according to the subdivision in 9 signal groups: Transducers, Frequency Sensors, Power Actuators, Frequency Actuators, Idle Speed Actuators, On/Off Inputs, On/Off Outputs, Serial and Power Supply/Reference
Control #10	(path)	Database Base Path			Specifies the base path where the Parts Database is stored

CADPROD.VI

Front Panel Elements:

Control #1	(ring)	Product		The type of product among the three supported by AutoSEFI: Electronic Control Units for Fuel Injection Systems (E.C.U.s), Dashboards and Alarm Devices
Control #2	(bool)	Ok		
Control #3	(cluster)	Panel		
	Control #3.1	(string)	Model	
Control #4	(cluster)	Alarm		
	Control #4.1	(string)	Model	
Control #5	(bool)	cancel		
Control #6	(cluster)	ECU		A cluster containing the registration attributes of an ECU
	Control #6.1	(string)	Model	The name to serve as a reference for the component in the Database
	Control #6.2	(array)	Transducers	Array of strings containing the relative paths of all transducers associated with this product
	Control #6.3	(array)	Frequency Sensors	Array of strings containing the relative paths of all frequency sensors associated with this product
	Control #6.4	(array)	Power Actuators	Array of strings containing the relative paths of all power actuators associated with this product
	Control #6.5	(array)	Frequency Actuators	Array of strings containing the relative paths of all frequency actuators associated with this product
	Control #6.6	(string)	Idle Speed Actuator	Contains the relative path of the idle speed actuator associated with this product
	Control #6.7	(string)	Associated Panel	Contains the relative path of the user interface panel associated with this product
Control #7	(path)	Database Base Path		Specifies the base path where the Parts Database is stored

ConfigProd.VI

Front Panel Elements:				
Control #1	(ring)	Product		The type of product among the three supported by AutoSEFI: Electronic Control Units for Fuel Injection Systems (E.C.U.s), Dashboards and Alarm Devices
Control #2	(bool)	Ok		
Control #3	(cluster)	Panel		
Control #3.1	(string)	Model		
Control #4	(cluster)	Alarm		
Control #4.1	(string)	Model		
Control #5	(bool)	cancel		
Control #6	(cluster)	ECU		A cluster containing the registration attributes of an ECU
Control #6.1	(string)	Model		The name to serve as a reference for the component in the Database
Control #6.2	(array)	Pin-Out		An array of Pins (clusters containing the attributes of a pin)
Control #6.2.1	(ring)	Type		The type of signal connected in the pin among the groups: Transducers, Frequency Sensors, Power Actuators, Frequency Actuators, Idle Speed Actuators, On-Off Inputs, On-Off Outputs, Serial and Power & Reference
Control #6.2.2	(string)	Name		The name or relative path of the component associated with the pin
Control #6.2.3	(u8)	System Pin		Number of the correspondent pin in the system connector
Control #6.3	(u8)	Max Pins		Number of pins at the ECU connector
Control #6.4	(string)	Associated Panel		Contains the relative path of the user interface panel associated with this product
Control #7	(path)	Database Base Path		Specifies the base path where the Parts Database is stored

EditProd.vi

Front Panel Elements:				
Control #1	(array)	ECU Pin-Out		An array of Pins (clusters containing the attributes of a pin)
Control #1.1	(string)	Name		The name or relative path of the component associated with the pin
Control #1.2	(u16)	Type		The type of signal connected in the pin
Control #1.3	(bool)	Apertou		
Control #2	(cluster)	Pin		Cluster containing the attributes of a pin
Control #2.1	(string)	Name		The name or relative path of the component associated with the pin
Control #2.2	(u16)	Type		The type of signal connected in the pin
Control #2.3	(bool)	Apertou		
Control #3	(path)	Database Base		Specifies the base path where the Parts Database is stored

		Path	
Control #4	(bool)	change?	Choose a new user interface panel
Control #5	(bool)	Ok	
Control #6	(bool)	cancel	
Control #7	(bool)	Up	
Control #8	(bool)	Down	
Control #9	(cluster)	ECU In	A cluster containing the registration attributes of an ECU
Control #9.1	(string)	Model	The name to serve as a reference for the component in the Database
Control #9.2	(array)	Pin-Out	An array of Pins (clusters containing the attributes of a pin)
Control #9.2.1	(ring)	Type	The type of signal connected in the pin among the groups: Transducers, Frequency Sensors, Power Actuators, Frequency Actuators, Idle Speed Actuators, On-Off Inputs, On-Off Outputs, Serial and Power & Reference
Control #9.2.2	(string)	Name	The name or relative path of the component associated with the pin
Control #9.2.3	(u8)	System Pin	Number of the correspondent pin in the system connector
Control #9.3	(u8)	Max Pins	Number of pins at the ECU connector
Control #9.4	(string)	Associated Panel	Contains the relative path of the user interface panel associated with this product
Indicator #1	(cluster)	ECU Out	A cluster containing the registration attributes of an ECU
Indicator #1.1	(string)	Model	The name to serve as a reference for the component in the Database
Indicator #1.2	(array)	Pin-Out	An array of Pins (clusters containing the attributes of a pin)
Indicator #1.2.1	(ring)	Type	The type of signal connected in the pin among the groups: Transducers, Frequency Sensors, Power Actuators, Frequency Actuators, Idle Speed Actuators, On-Off Inputs, On-Off Outputs, Serial and Power & Reference
Indicator #1.2.2	(string)	Name	The name or relative path of the component associated with the pin
Indicator #1.2.3	(u8)	System Pin	Number of the correspondent pin in the system connector
Indicator #1.3	(u8)	Max Pins	Number of pins at the ECU connector
Indicator #1.4	(string)	Associated Panel	Contains the relative path of the user interface panel associated with this product
Indicator #2	(string)	Model	The name to serve as a reference for the component in the Database
Indicator #3	(string)	Associated Panel	Contains the relative path of the user interface panel associated with this product
Indicator #4	(array)	Pin No.	
Indicator #5	(bool)	Remove Product? (F)	

EditProd1.vi

Front Panel Elements:				
Control #1	(array)	ECU Pin-Out		An array of Pins (clusters containing the attributes of a pin)
	Control #1.1	(string)	Name	The name or relative path of the component associated with the pin
	Control #1.2	(u16)	Type	The type of signal connected in the pin
	Control #1.3	(bool)	Apertou	
Control #2	(cluster)	Pin		Cluster containing the attributes of a pin
	Control #2.1	(string)	Name	The name or relative path of the component associated with the pin
	Control #2.2	(u16)	Type	The type of signal connected in the pin
	Control #2.3	(bool)	Apertou	
Control #3	(path)	Database Base Path		Specifies the base path where the Parts Database is stored
Control #4	(bool)	change?		Choose a new user interface panel
Control #5	(bool)	Ok		
Control #6	(bool)	cancel		
Control #7	(bool)	Up		
Control #8	(bool)	Down		
Control #9	(cluster)	ECU In		A cluster containing the registration attributes of an ECU
	Control #9.1	(string)	Model	The name to serve as a reference for the component in the Database
	Control #9.2	(array)	Pin-Out	An array of Pins (clusters containing the attributes of a pin)
	Control #9.2.1	(ring)	Type	The type of signal connected in the pin among the groups: Transducers, Frequency Sensors, Power Actuators, Frequency Actuators, Idle Speed Actuators, On-Off Inputs, On-Off Outputs, Serial and Power & Reference
	Control #9.2.2	(string)	Name	The name or relative path of the component associated with the pin
	Control #9.2.3	(u8)	System Pin	Number of the correspondent pin in the system connector
	Control #9.3	(u8)	Max Pins	Number of pins at the ECU connector
	Control #9.4	(string)	Associated Panel	Contains the relative path of the user interface panel associated with this product
Indicator #1	(cluster)	ECU Out		A cluster containing the registration attributes of an ECU
	Indicator #1.1	(string)	Model	The name to serve as a reference for the component in the Database
	Indicator #1.2	(array)	Pin-Out	An array of Pins (clusters containing the attributes of a pin)
	Indicator #1.2.1	(ring)	Type	The type of signal connected in the pin among the groups: Transducers, Frequency Sensors, Power Actuators, Frequency Actuators, Idle Speed Actuators, On-Off Inputs, On-Off Outputs, Serial and Power & Reference
	Indicator #1.2.2	(string)	Name	The name or relative path of the component associated with the pin

	Indicator #1.2.3	(u8)	System Pin	Number of the correspondent pin in the system connector
	Indicator #1.3	(u8)	Max Pins	Number of pins at the ECU connector
	Indicator #1.4	(string)	Associated Panel	Contains the relative path of the user interface panel associated with this product
Indicator #2	(string)	Model		The name to serve as a reference for the component in the Database
Indicator #3	(string)	Associated Panel		Contains the relative path of the user interface panel associated with this product
Indicator #4	(array)	Pin No.		
Indicator #5	(bool)	Remove Product? (F)		
Indicator #6	(u32)	Pressed Pin		Product pin to be edited

MESSAGES.VI

This Vi displays the simulator's error and warning messages in a Dialog Box form

Front Panel Elements:

Control #1	(i32)	Message Number	The number of the desired message
Control #2	(bool)	With Cancel?	Display or not the 'Cancel' button Default value is FALSE
Control #3	(string)	ExtraInfo	
Indicator #1	(string)	Message	The displayed message according to Message Number
Indicator #2	(u32)	Ok/Cancel	Reflects the user's action. 0 means that OK was the button pressed

NAMESEL.VI

This VI selects a name to be a reference for a component or group of components from Compsel.VI and checks its validity

Front Panel Elements:

Control #1	(string)	Name	
Control #2	(bool)	Group	Will the name to be chosen probably be a Group of components?
Control #3	(bool)	Ok	
Control #4	(bool)	Cancel	
Control #5	(enum)	Name Type	The type of the name to be selected: Component, Product or a General Use one
Indicator #1	(string)	Name Out	Name chosen
Indicator #2	(bool)	Group? Out	Is the name chosed a Group of components?
Indicator #3	(string)	group	
Indicator #4	(string)	comp	

PARTSEL.VI

This VI works with the Products & Components (Parts) Database, through the following actions: creating empty parts of the desired type, deleting and modifying them, grouping them for easier manipulation or only selecting a part to get its path.
OBS: This VI was built with the intention of making every file operations transparent in respect to the final user of the system. In this way, all the information presented on the front panel shows parts and groups of parts referencing files and directories respectively

Front Panel Elements:

Control #1	(u16)	Component/Product/Panel Type	Selects the part subtype whithin the following possible options: - Component 0- Transducer(SensorI) 1- Frequency Sensor(Sensor II) 2- Power Actuator(ActuatI) 3- Frequency Actuator(ActuatII) 4- Idle Speed Actuator(IdlSpeed) - Product & Interface Panel 0- ECU 1- Dashboard 2- Alarm
Control #2	(listbox)	Group List	Navigation box listing the names of the groups in this level of the Database. Double-clicking in the '..' symbol takes you one level up, and in a group name takes you one level down inside that group
Control #3	(bool)	del	
Control #4	(bool)	new	
Control #5	(bool)	mod	
Control #6	(bool)	can	
Control #7	(listbox)	Part List	Selection box listing the parts inside the group highlited in 'Group List'. Double-clicking a component name selects it
Control #8	(bool)	With Operations? (T)	Selects if the VI will be used to modify the Database or only to get the part path
Control #9	(enum)	Part Type (0)	Selects the part type within the following possible options: 0- Component (default) 1- Product 2- Interface Panel
Control #10	(path)	Database Base Path	Specifies the base path where the Parts Database is stored
Indicator #1	(path)	Actual Path	Path of the edited/selected part or <empty path> if not aplicable

PARTSEL1.VI

This VI works with the Products & Components (Parts) Database, through the following actions: creating empty parts of the desired type, deleting and modifying them, grouping them for easier manipulation or only selecting a part to get its path.
OBS: This VI was built with the intention of making every file operations transparent in respect to the final user of the system. In this way, all the information presented on the front panel shows parts and groups of parts referencing files and directories respectively

Front Panel Elements:			
Control #1	(u16)	Component/Product/Panel Type	Selects the part subtype whithin the following possible options: - Component 0- Transducer(SensorI) 1- Frequency Sensor(Sensor II) 2- Power Actuator(ActuatI) 3- Frequency Actuator(ActuatII) 4- Idle Speed Actuator(IdlSpeed) - Product & Interface Panel 0- ECU 1- Dashboard 2- Alarm
Control #2	(listbox)	Group List	Navigation box listing the names of the groups in this level of the Database. Double-clicking in the '..' symbol takes you one level up, and in a group name takes you one level down inside that group
Control #3	(bool)	del	
Control #4	(bool)	new	
Control #5	(bool)	mod	
Control #6	(bool)	can	
Control #7	(listbox)	Part List	Selection box listing the parts inside the group highlited in 'Group List'. Double-clicking a component name selects it
Control #8	(bool)	With Operations? (T)	Selects if the VI will be used to modify the Database or only to get the part path
Control #9	(enum)	Part Type (0)	Selects the part type within the following possible options: 0- Component (default) 1- Product 2- Interface Panel
Control #10	(path)	Database Base Path	Specifies the base path where the Parts Database is stored
Indicator #1	(path)	Actual Path	Path of the edited/selected part or <empty path> if not aplicable

PRODEDIT.VI

Front Panel Elements:				
Control #1	(listbox)	Component List		List of the components, of the type selected in the ring beside, that are currently associated with this product. Empty spaces may appear between the names during a session with add and change operations and should be disregarded
Control #2	(bool)	add		Adds a new component of the type selected in the ring beside
Control #3	(bool)	mod		
Control #4	(bool)	del		
Control #5	(bool)	Ok		
Control #6	(bool)	cancel		
Control #7	(bool)	change		Choose a new user interface panel
Control #8	(ring)	Type		The type of component according to the subdivision in 9 signal groups: Transducers, Frequency Sensors, Power Actuators, Frequency Actuators, Idle Speed Actuators, On/Off Inputs, On/Off Outputs, Serial and Power Supply/Reference
Control #9	(cluster)	ECU In		
Control #9.1	(string)	Model		A string corresponding to the product's model name, and therefore its reference name in the Database
Control #9.2	(array)	Transducers		Array of strings, each one containing the name of a component and its type, in the format: "type/name.cmp"
Control #9.3	(array)	Frequency Sensors		Array of strings, each one containing the name of a component and its type, in the format: "type/name.cmp"
Control #9.4	(array)	Actuators Type I		Array of strings, each one containing the name of a component and its type, in the format: "type/name.cmp"
Control #9.5	(array)	Actuators Type II		Array of strings, each one containing the name of a component and its type, in the format: "type/name.cmp"
Control #9.6	(string)	Idle Speed Actuator		A string containing the name of the component and its type, in the format: "type/name.cmp"
Control #9.7	(string)	Associated Panel		A string containing the name of the associated panel: "name.vi"
Control #10	(cluster)	ECU Out Std		
Control #10.1	(string)	Model		A string corresponding to the product's model name, and therefore its reference name in the Database
Control #10.2	(array)	Transducers		Array of strings, each one containing the name of a component and its type, in the format: "type/name.cmp"
Control #10.3	(array)	Frequency Sensors		Array of strings, each one containing the name of a component and its type, in the format: "type/name.cmp"
Control #10.4	(array)	Actuators Type I		Array of strings, each one containing the name of a component and its type, in the format: "type/name.cmp"
Control #10.5	(array)	Actuators Type II		Array of strings, each one containing the name of a component and its type, in the format: "type/name.cmp"
Control #10.6	(string)	Idle Speed Actuator		A string containing the name of the component and its type, in the format: "type/name.cmp"
Control #10.7	(string)	Associated Panel		A string containing the name of the associated panel: "name.vi"

Control #11	(path)	Database Base Path	
Indicator #1	(string)	Model	The name to serve as a reference for the component in the Database
Indicator #2	(string)	Associated Panel	Contains the relative path of the user interface panel associated with this product
Indicator #3	(cluster)	ECU Out	
Indicator #3.1	(string)	Model	A string corresponding to the product's model name, and therefore its reference name in the Database
Indicator #3.2	(array)	Transducers	Array of strings, each one containing the name of a component and its type, in the format: "type/name.cmp"
Indicator #3.3	(array)	Frequency Sensors	Array of strings, each one containing the name of a component and its type, in the format: "type/name.cmp"
Indicator #3.4	(array)	Actuators Type I	Array of strings, each one containing the name of a component and its type, in the format: "type/name.cmp"
Indicator #3.5	(array)	Actuators Type II	Array of strings, each one containing the name of a component and its type, in the format: "type/name.cmp"
Indicator #3.6	(string)	Idle Speed Actuator	A string containing the name of the component and its type, in the format: "type/name.cmp"
Indicator #3.7	(string)	Associated Panel	A string containing the name of the associated panel: "name.vi"

PRODEDIT1.VI

Front Panel Elements:			
Control #1	(listbox)	Component List	List of the components, of the type selected in the ring beside, that are currently associated with this product. Empty spaces may appear between the names during a session with add and change operations and should be disregarded
Control #2	(bool)	add	Adds a new component of the type selected in the ring beside
Control #3	(bool)	mod	
Control #4	(bool)	del	
Control #5	(bool)	Ok	
Control #6	(bool)	cancel	
Control #7	(bool)	change	Choose a new user interface panel
Control #8	(ring)	Type	The type of component according to the subdivision in 9 signal groups: Transducers, Frequency Sensors, Power Actuators, Frequency Actuators, Idle Speed Actuators, On/Off Inputs, On/Off Outputs, Serial and Power Supply/Reference
Control #9	(cluster)	ECU In	
Control #9.1	(string)	Model	The name to serve as a reference for the component in the Database
Control #9.2	(array)	Pin-Out	An array of Pins (clusters containing the attributes of a pin)
Control #9.2.1	(ring)	Type	The type of signal connected in the pin among the groups: Transducers, Frequency Sensors, Power Actuators, Frequency Actuators, Idle Speed Actuators, On-Off Inputs, On-Off Outputs, Serial and Power &

					Reference
	Control #9.2.2	(string)	Name		The name or relative path of the component associated with the pin
	Control #9.2.3	(u8)	System Pin		Number of the correspondent pin in the system connector
	Control #9.3	(u8)	Max Pins		Number of pins at the ECU connector
	Control #9.4	(string)	Associated Panel		Contains the relative path of the user interface panel associated with this product
Control #10	(cluster)	ECU Out Std			
	Control #10.1	(string)	Model		The name to serve as a reference for the component in the Database
	Control #10.2	(array)	Pin-Out		An array of Pins (clusters containing the attributes of a pin)
		Control #10.2.1	(ring)	Type	The type of signal connected in the pin among the groups: Transducers, Frequency Sensors, Power Actuators, Frequency Actuators, Idle Speed Actuators, On-Off Inputs, On-Off Outputs, Serial and Power & Reference
		Control #10.2.2	(string)	Name	The name or relative path of the component associated with the pin
		Control #10.2.3	(u8)	System Pin	Number of the correspondent pin in the system connector
	Control #10.3	(u8)	Max Pins		Number of pins at the ECU connector
	Control #10.4	(string)	Associated Panel		Contains the relative path of the user interface panel associated with this product
Control #11	(path)	Database Base Path			
Control #12	(array)	Pin-Out		An array of Pins (clusters containing the attributes of a pin)	
		Control #12.1	(string)	Name	The name or relative path of the component associated with the pin
		Control #12.2	(u16)	Type	The type of signal connected in the pin
		Control #12.3	(bool)	Apertou	
Control #13	(cluster)	Pin		A cluster containing the attributes of a pin	
		Control #13.1	(string)	Name	The name or relative path of the component associated with the pin
		Control #13.2	(u16)	Type	The type of signal connected in the pin
		Control #13.3	(bool)	Apertou	
Indicator #1	(string)	Model		The name to serve as a reference for the component in the Database	
Indicator #2	(string)	Associated Panel		Contains the relative path of the user interface panel associated with this product	
Indicator #3	(cluster)	ECU Out Std			
	Indicator	(string)	Model		The name to serve as a reference for the component in the

#3.1				Database
Indicator #3.2	(array)	Pin-Out		An array of Pins (clusters containing the attributes of a pin)
Indicator #3.2.1	(ring)	Type		The type of signal connected in the pin among the groups: Transducers, Frequency Sensors, Power Actuators, Frequency Actuators, Idle Speed Actuators, On-Off Inputs, On-Off Outputs, Serial and Power & Reference
Indicator #3.2.2	(string)	Name		The name or relative path of the component associated with the pin
Indicator #3.2.3	(u8)	System Pin		Number of the correspondent pin in the system connector
Indicator #3.3	(u8)	Max Pins		Number of pins at the ECU connector
Indicator #3.4	(string)	Associated Panel		Contains the relative path of the user interface panel associated with this product
Indicator #4	(bool)	Remove Product? (F)		

PRODUCTS1.VI

Reference for other VIs of the data structure of a Product

Front Panel Elements:

Indicator #1	(cluster)	ECU		A cluster containing the registration atributes of an ECU
Indicator #1.1	(string)	Model		The name to serve as a reference for the component in the Database
Indicator #3.2	(array)	Pin-Out		An array of Pins (clusters containing the attributes of a pin)
Indicator #1.2.1	(ring)	Type		The type of signal connected in the pin among the groups: Transducers, Frequency Sensors, Power Actuators, Frequency Actuators, Idle Speed Actuators, On-Off Inputs, On-Off Outputs, Serial and Power & Reference
Indicator #1.2.2	(string)	Name		The name or relative path of the component associated with the pin
Indicator #1.2.3	(u8)	System Pin		Number of the correspondent pin in the system connector
Indicator #1.3	(u8)	Max Pins		Number of pins at the ECU connector
Indicator #1.4	(string)	Associated Panel		Contains the relative path of the user interface panel associated with this product

VTFITTIN.VI			
Front Panel Elements:			
Control #1	(array)	Input VT	The Input Values Table to be processed. It receives the current values in the attribute box of 'Cadastro.vi' but can be edited before pressing the 'Go' button.
Control #2	(bool)	Go	to process the data in the Input VT.
Control #3	(dbl)	Interval between Samples	The interval between samples, in physical units (1st column), in the Interpolated VT.
Control #4	(bool)	Use Resistive Divider (F)	Turn on to convert resistance values to voltage values in the Interpolated Values Table, according to the Rup and Vcc values.
Control #5	(dbl)	Rup Value (kW)	
Control #6	(dbl)	Vcc Value (V)	
Control #7	(bool)	Use Exponential Transfer Function (F)	Turn on to replace the original transfer function with that evaluated by the Best Exponential Fit algorithm.
Control #8	(string)	Current Input Transfer Function	Trasfer Function as it comes from 'Cadastro.VI'.
Indicator #1	(array)	Best Exp Fit VT	
Indicator #2	(dbl)	mean squared error	The mean squared error of the best exponential fit, in comparison with the original set of data.
Indicator #3	(array)	Interpolated VT	The Values Table after being processed by the spline interpolation on the Input VT and the resistance/voltage convertion, if applicable.
Indicator #4	(string)	New Transfer Function	Original or exponential transfer function evaluated as decided by the user.
Indicator #5	(string)	Exponential Transfer Function	Transfer function evaluated by the Best Exponential Fit algorithm.

Nesta biblioteca também estão os seguintes VIs, que não serão apresentados aqui em detalhe: EncodeECU.vi, CheckPin_Out.vi, CheckPin_Out1.vi, CLUSTCFG.VI, COMPNNTS.VI, COMPOPS.VI, ConfigCluster.VI, PRODOPS.VI, PRODOPS1.VI e PRODUCTS.VI.

Na biblioteca Security.llb

Retype.VI

Front Panel Elements:		
Control #1	(string)	Password
Control #2	(bool)	Ok
Control #3	(bool)	Ccl
Indicator #1	(string)	Retyped Password

UserID.VI

Front Panel Elements:			
Control #1	(cluster)	User Properties	A cluster containing the register information about the selected user. It has the following fields:
Control #1.1	(string)	Login	Nickname chosen by the user
Control #1.2	(string)	Password	Password associated with the login
Control #1.3	(array)	Privileges	An array of Actors representing wich packages may be used by the user. In the current version, AutoSEFI takes in count only the privileges of the first Actor in the array.
Control #1.3.1	(enum)	Actor	The possible roles the user can play with the system. A role is specified through the access permission to a package of the system.
Control #1.4	(string)	Name	Full name of the user.
Control #2	(listbox)	Users	These are the currently registered users.
Control #3	(bool)	exit	Exits the User Manager
Control #4	(bool)	remove	Removes selected user.
Control #5	(bool)	modify	Modifies selected user
Control #6	(bool)	add	Adds new user
Control #7	(cluster)	User Model	A cluster containing the register information model for an user. It has the following fields:
Control #7.1	(string)	Login	Nickname chosen by the user
Control #7.2	(string)	Password	Password associated with the login
Control #7.3	(array)	Privileges	An array of Actors representing wich packages may be used by the user. In the current version, AutoSEFI takes in count only the privileges of the first Actor in the array.
Control #7.3.1	(enum)	Actor	The possible roles the user can play with the system. A role is specified through the access permission to a package of the system.
Control #7.4	(string)	Name	Full name of the user.
Control #8	(bool)	Just Added?	Flag to indicate that the current user is a new one and thus, if no complete information is provided, he should be expelled from the Database in the Modify User State

Na biblioteca Simulation.llb

CheckStepPosit2.VI			
Version for aquiring stepper through digitals			
Front Panel Elements:			
Control #1	(array)	Data In	
Control #1.1	(u32)	pattern	
Control #2	(i32)	Iteration	
Control #3	(cluster)	Config Info	
Control #3.1	(u32)	SensorITaskID	
Control #3.2	(u32)	DigitalInputTaskID	
Control #3.3	(u32)	SensorIITaskID	
Control #3.4	(u32)	IdleSpeedTaskID	
Control #3.5	(u32)	ActuatorITaskID	
Control #3.6	(u32)	DigitalOutputTaskID	
Control #3.7	(u32)	ActuatorIITaskID	
Control #3.8	(u32)	TimingRefTaskID	
Control #3.9	(cluster)	Config error	
Control #3.9.1	(bool)	status	
Control #3.9.2	(i32)	code	
Control #3.9.3	(string)	source	
Control #3.10	(dbl)	ActuatorIIPar	
Control #3.11	(u32)	VbattTaskID	
Control #3.12	(cluster)	IdleSpeedPars	
Control #3.12.1	(dbl)	Number of Steps	
Control #3.12.2	(array)	Phase Sequence	
Control #4	(i32)	Reference In	
Control #5	(bool)	Read?	
Indicator #1	(cluster)	Stepper	
Indicator #1.1	(array)	Data	
Indicator #1.2	(bool)	Read?	
Indicator #1.3	(i32)	Last	
Indicator #2	(array)	Data Out	
Indicator #3	(cluster)	error out	
Indicator #3.1	(bool)	status	
Indicator #3.2	(i32)	code	
Indicator #3.3	(string)	source	
Indicator #4	(i32)	Reference Out	

CTR Control2.VI
This VI controls and reads groups of counters. Control operations include starting, stopping, and setting the output state
Front Panel Elements:

CTR Group Config2.VI

This VI collects one or more counters into a group. Counter groups containing more than one counter are useful for starting, stopping, or reading multiple counters simultaneously. Multiple counter groups are not currently supported by MIO-E Series boards.

Front Panel Elements:

CTR Mode Config2.VI

This VI configures one or more counters for the type of counter operation you want to perform and selects the source signal, gating mode, and output behavior on terminal count. This VI does not start the counters. If you are using a counter for pulse generation, you do not have to call this VI unless you want to change the gating mode or output behavior.

Front Panel Elements:

General Error Handler for Sim.VI

This error handler is used primarily to inform the user if an input error exists, to describe the error, and to identify where it occurred. The information for this is derived from the inputs error in, error code, and error source, and from an internal error description table. The table describes all errors that can be created by LabVIEW or its associated I/O operations. The handler has provisions to take alternative actions, such as to cancel or set an error status, and to test for and describe user-defined errors. See instruction on the front panel.

Front Panel Elements:

GET_PARS.VI

This Vi retrieves the information about the components of an ECU from the Database, that are required for the configuration of the DAQ boards

Front Panel Elements:

Control #1	(path)	ECU Path	The path of the ECU from which the information shall be retrieved.	
Control #2	(cluster)	limit settings		
Control #2.1	(sgl)	high limit (10 V)	specifies the maximum voltage the board measures at a particular channel.	
Control #2.2	(sgl)	low limit (-10 V)	specifies the minimum voltage the board measures at a particular channel.	
Control #2.3	(u16)	reference source (no change:0)		
Control #3	(cluster)	input limits	an array of clusters, of which each array element specifies the range limits for the channel(s) in the corresponding element of the channels array. If there are fewer elements in this array than the number of channels, the VI uses the last element for the rest of the channels. The default input is an empty array, which means the input limits do not change from their default settings. Each cluster contains the following parameters:	
Control #3.1	(sgl)	high limit (10 V)	specifies the maximum voltage the board measures at a particular channel.	
Control #3.2	(sgl)	low limit (-10 V)	specifies the minimum voltage the board measures at a particular channel.	
Control #4	(path)	Database Base Path		
Indicator #1	(cluster)	SensorI parameters	a cluster containing information for the Transducers simulation.	
Indicator #1.1	(array)	SensorI Port List	Array of strings with the analog output channels to be configured.	
Indicator #1.2	(array)	limit settings (Output)	The signal limits and reference for each channel	
Indicator #1.2.1	(cluster)			
Indicator #1.2.1.1	(sgl)	high limit (10 V)		
Indicator #1.2.1.2	(sgl)	low limit (-10 V)		
Indicator #1.2.1.3	(u16)	reference source (no change:0)		
Indicator #2	(cluster)	SensorII parameters	a cluster containing information for the Frequency Sensors simulation.	
Indicator #2.1	(array)	SensorII Port List	Array of strings with the analog output channels to be configured.	
Indicator #2.2	(array)	limit settings (Output)	The signal limits and reference for each channel	
Indicator #2.2.1	(cluster)			

		Indicator #2.2.1.1	(sgl)	high limit (10 V)	
		Indicator #2.2.1.2	(sgl)	low limit (-10 V)	
		Indicator #2.2.1.3	(u16)	reference source (no change:0)	
Indicator #3	(cluster)	ActuatorI parameters		a cluster containing information for the Power Actuators acquisition.	
	Indicator #3.1	(array)	ActuatorI Port List		Array of strings with the analog input channels to be configured.
	Indicator #3.2	(array)	input limits (Input)		an array of clusters, of which each array element specifies the range limits for the channel(s) in the corresponding element of the channels array. If there are fewer elements in this array than the number of channels, the VI uses the last element for the rest of the channels. The default input is an empty array, which means the input limits do not change from their default settings. Each cluster contains the following parameters:
		Indicator #3.2.1	(cluster)		
		Indicator #3.2.1.1	(sgl)	high limit (10 V)	specifies the maximum voltage the board measures at a particular channel.
		Indicator #3.2.1.2	(sgl)	low limit (-10 V)	specifies the minimum voltage the board measures at a particular channel.
Indicator #4	(path)	Associated Panel		Path of the interface panel associated with this ECU.	
Indicator #5	(cluster)	ActuatorII parameters		a cluster containing information for the Frequency Actuators acquisition.	
	Indicator #5.1	(array)	Operating Frequencies		The operating frequencies for each actuator, used to configure the counter/timers and calculate duty cycles
		Indicator #5.1.1	(dbl)	Operating Frequency	
Indicator #6	(cluster)	IdleSpeed parameters			
	Indicator #6.1	(dbl)	Number of Steps/Op. Freq.		
	Indicator #6.2	(array)	Forward Step Phase Sequence		Each pattern is a (u32)

My Error Handler.VI

This error handler is used primarily to inform the user if an input error exists, to describe the error, and to identify where it occurred. The information needed to do this is derived from the inputs error cluster in, error code, and error source, and from an internal error description table. The table lists all errors that can be created by LabVIEW or its associated I/O operations.

Front Panel Elements:

POWERCFG2.VI

This VI configures the hardware and allocates a buffer for a buffered analog input operation for a specified group of channels. Before performing any configuration, AI Config checks to see if the input cluster error in indicates that an error has already occurred. If so, then this VI does no configuration, but passes the error information unmodified through error out. In an error condition, taskID is 0. Otherwise, this VI calls several low-level analog input VIs to set up a task for a buffered analog input acquisition. AI Config calls Analog Input Group Config for the specified device, group, and channels to create a taskID. AI Config then calls Analog Input Hardware Config to record information about the hardware configuration. The input limits input passes to Analog Input Hardware Config, which sets up the lower and upper voltage limits for each channel; if unspecified, the input limits do not change. This VI configures the number of AMUX boards, the coupling used for each channel (that is, AC, DC, ground, or internal reference), and the input configuration (differential, referenced single-ended, or non-referenced single ended). You can use the low-level Analog Input Hardware Config VI instead if you prefer to specify the input signal range in terms of signal range, polarity, and gain. AI Config configures the buffer for the acquisition by calling Analog Input Buffer Config. Analog Input Buffer Config allocates memory for the specified number of buffers (the default value is one), where each buffer contains buffer size number of scans.

Front Panel Elements:

PulseMeas.VI

Measures the pulse width (length of time a signal is high or low) or period (length time between adjacent rising or falling edges) of a TTL signal connected to the counter's GATE pin. This VI is useful in measuring the frequency of relatively low frequency signals; use the Measure Frequency VI for relatively high frequency signals. The VI iterates until a valid measurement, timeout, or counter overflow occurs. A valid measurements exists when count is >= 4 without a counter overflow. If counter overflow occurs, lower the timebase. If you start a pulse width measurement during the phase you want to measure, you will get a short count. Therefore, make sure the pulse does not occur until after the counter is started. This restriction does not apply to period measurements.
The VI calls the Pulse Width or Period Meas Config, Counter Start, Counter Read, and Counter Stop VIs.

Front Panel Elements:

PulseMeas2.VI

Measures the pulse width (length of time a signal is high or low) or period (length time between adjacent rising or falling edges) of a TTL signal connected to the counter's GATE pin. This VI is useful in measuring the frequency of relatively low frequency signals; use the Measure Frequency VI for relatively high frequency signals. The VI iterates until a valid measurement, timeout, or counter overflow occurs. A valid measurements exists when count is >= 4 without a counter overflow. If counter overflow occurs, lower the timebase. If you start a pulse width measurement during the phase you want to measure, you will get a short count. Therefore, make sure the pulse does not occur until after the counter is started. This restriction does not apply to period measurements.
The VI calls the Pulse Width or Period Meas Config, Counter Start, Counter Read, and Counter Stop VIs.

Front Panel Elements:

PulseMeas2Config.VI

Configures the specified counter to measure the pulse width or period of a TTL signal connected to its GATE pin. The measurement is done by counting the number of cycles of the specified timebase between the appropriate starting and ending events. To accurately measure pulse width, the pulse must occur after the counter is started. Call Counter Start to start the operation. You can also use this VI to measure the frequency of low frequency signals.

Front Panel Elements:

PulseMeasConfig.VI

Configures the specified counter to measure the pulse width or period of a TTL signal connected to its GATE pin. The measurement is done by counting the number of cycles of the specified timebase between the appropriate starting and ending events. To accurately measure pulse width, the pulse must occur after the counter is started. Call Counter Start to start the operation. You can also use this VI to measure the frequency of low frequency signals.

Front Panel Elements:

RPMARRAY_3.VI

This VI generates the bidimensional array that contains the values to be sent to the Output Buffer, in order to simulate the desired RPM and Knock signals, according to the 8333,33 points/sec chosen update rate and the RPM values that can be simulated with the current version of the system.

Front Panel Elements:

Control #1	(array)	RPM Patterns	A tridimensional array of (sgl) containing the bidimensional arrays for each of the possible RPM values, that could be sent to the Output Buffer, in order to simulate the desired RPM signal.
Control #2	(array)	Knock Patterns	A bidimensional array of (dbl) containing the singledimensional arrays for each of the possible RPM values, that could be sent to the Output Buffer, in order to simulate the desired Knock signal.
Control #3	(bool)	Setup?(F)	If Setup is true, the VI prepares the tables with all the available RPM and Knock Patterns, which takes a long execution time.
Control #4		RPM	The RPM value set by the user.
Control #5	(bool)	Knock?(F)	When True, the table Current RPM Pattern will also contain Knock information to be simulated.
Indicator #1	(array)	Current RPM Pattern	An array of (dbl) containing the values to be sent to the Output Buffer, in order to simulate the desired RPM and Knock signals.
Indicator #2	(i16)	Actual RPM	The RPM value that can be simulated and is the closest to the user selected RPM.

TRANSCFG.VI

This VI configures an analog output operation for a specified set of channels. This VI records the hardware configuration and allocates a buffer for a buffered analog output operation. Before performing any configuration, AO Config checks the input cluster error in to determine whether an error has already occurred. If so, then this VI does no configuration, but passes the error information modified through error out. In an error condition, taskID is 0. Otherwise, this VI calls several low-level analog output VIs to set up a task for a buffered analog output. AO Config calls Analog Output Group Config for the specified device, group, and channels to create a taskID. AO Config then calls Analog Output HW Config to record information about the hardware configuration. AO Config configures the buffer for buffered waveform generation by calling Analog Output Buffer Config. Analog Output Buffer Config allocates memory for the number of updates specified by buffer size.

Front Panel Elements:

Runtest.VI

This VI performs an entire simulation on a product, from the Tecnician identification, through the stimulus generation and responses analisys and to the simulation report.

Front Panel Elements:

Control #1	(cluster)	Config Info		Contains all configuration information required to access the DAQ boards.
Control #1.1	(u32)	SensorITaskID		Reference for AO operations regarding Transducers.
Control #1.2	(u32)	DigitalInputTaskID		
Control #1.3	(u32)	SensorIITaskID		Reference for AO operations regarding Frequency Sensors.
Control #1.4	(u32)	IdleSpeedTaskID		Reference for counter operations regarding Idle Speed Actuators.
Control #1.5	(u32)	ActuatorITaskID		Reference for AI operations regarding Power Actuators.
Control #1.6	(u32)	DigitalOutputTaskID		
Control #1.7	(u32)	ActuatorIITaskID		Reference for AI operations regarding Frequency Actuators.
Control #1.8	(u32)	TimingRefTaskID		
Control #1.9	(cluster)	Config error		
Control #1.9.1	(bool)	status		
Control #1.9.2	(i32)	code		
Control #1.9.3	(string)	source		
Control #1.10	(dbl)	ActuatorIIPar		
Control #1.11		VbattTaskID		
Control #1.12	(cluster)	IdleSpeedPars		
Control #1.12.1	(dbl)	Number of Steps		
Control #1.12.2	(array)	Phase Sequence		
Control #2	(path)	Database Base Path		Specifies the base path where the Parts Database is stored.
Control #3	(bool)	Off		
Control #4	(cluster)	Panel Input Data		
Control #4.1	(u32)	Digital Inputs		
Control #4.2	(dbl)	Advance Angle		
Control #4.3	(dbl)	Dwell Angle		
Control #4.4	(dbl)	Cannister		
Control #4.5	(u32)	Stepper Motor		
Control #4.6	(array)	Injectors		Array of (sgl)
Control #4.7	(dbl)	Battery		
Control #5	(bool)	AIOk?		
Indicator #1	(bool)	AllOk?		
Indicator #2	(dbl)	Cannister		

Indicator #3	(bool)	Fatal error		
Indicator #4	(string)	Panel Error message		
Indicator #5	(cluster)	Panel Output Data		
Indicator #5.1	(bool)	Panel Power		
Indicator #5.2	(u32)	Digital Outputs		
Indicator #5.3	(array)	Transducers	Array of (sgl)	
Indicator #5.4	(i8)	Selected Power Actuator		
Indicator #5.5	(i16)	RPM		
Indicator #5.6	(bool)	KNOCK		
Indicator #5.7	(u16)	Lambda Mode		
Indicator #5.8	(i8)	Trigger Adjust		
Indicator #6	(i16)	Actual RPM		
Indicator #7	(string)	Stepper Error		
Indicator #8	(u32)	Position		
Indicator #9	(bool)	End Test?		

Runtest2.VI

This VI performs an entire simulation on a product, from the Tecnician identification, through the stimulus generation and responses analisys and to the simulation report.
This version acquires the stepper signal through digitals.

Front Panel Elements:

Control #1	(cluster)	Config Info	Contains all configuration information required to access the DAQ boards.	
Control #1.1	(u32)	SensorITaskID	Reference for AO operations regarding Transducers.	
Control #1.2	(u32)	DigitalInputTaskID		
Control #1.3	(u32)	SensorIITaskID	Reference for AO operations regarding Frequency Sensors.	
Control #1.4	(u32)	IdleSpeedTaskID	Reference for counter operations regarding Idle Speed Actuators.	
Control #1.5	(u32)	ActuatorITaskID	Reference for AI operations regarding Power Actuators.	
Control #1.6	(u32)	DigitalOutputTaskID		

	Control #1.7	(u32)	ActuatorIITaskID		Reference for AI operations regarding Frequency Actuators.
	Control #1.8	(u32)	TimingRefTaskID		
	Control #1.9	(cluster)	Config error		
		Control #1.9.1	(bool)	status	
		Control #1.9.2	(i32)	code	
		Control #1.9.3	(string)	source	
	Control #1.10	(dbl)	ActuatorIIPar		
	Control #1.11		VbattTaskID		
	Control #1.12	(cluster)	IdleSpeedPars		
		Control #1.12.1	(dbl)	Number of Steps	
		Control #1.12.2	(array)	Phase Sequence	
Control #2	(enum)	Initial State		{Initialization, Configuration, Simulation, History, Error, Stop}	
Control #3	(path)	Database Base Path		Specifies the base path where the Parts Database is stored.	
Control #4	(bool)	Acquiring			
Control #5	(bool)	Off			
Control #6	(cluster)	Panel Input Data			
	Control #6.1	(u32)	Digital Inputs		
	Control #6.2	(dbl)	Advance Angle		
	Control #6.3	(dbl)	Dwell Angle		
	Control #6.4	(dbl)	Cannister		
	Control #6.5	(array)	Stepper Motor		Array of (u32)
	Control #6.6	(array)	Injectors		Array of (sgl)
	Control #6.7	(dbl)	Battery		
Control #7	(bool)	Monitoring Only			
Control #8	(bool)	AIOk?			
Indicator #1	(bool)	AllOk?			
Indicator #2	(dbl)	Cannister			
Indicator #3	(dbl)	Mark time			
Indicator #4	(dbl)	Period			
Indicator #5	(bool)	Fatal error			
Indicator #6	(string)	Panel Error message			
Indicator #7	(cluster)	Stepper			
	Indicator #7.1	(array)	Data		Array of (u32)
	Indicator #7.2	(bool)	Read?		
	Indicator #7.3	(i32)	Last		

Indicator #8		(cluster)	Panel Output Data	
	Indicator #8.1	(bool)	Panel Power	
	Indicator #8.2	(u32)	Digital Outputs	
	Indicator #8.3	(array)	Transducers	Array of (sgl)
	Indicator #8.4	(i8)	Selected Power Actuator	
	Indicator #8.5	(i16)	RPM	
	Indicator #8.6	(bool)	KNOCK	
	Indicator #8.7	(u16)	Lambda Mode	
	Indicator #8.8	(i8)	Trigger Adjust	
Indicator #9		(i16)	Actual RPM	

Nesta biblioteca também estão os seguintes VIs, que não serão apresentados aqui em detalhe: Runtesttemp.vi, Interact.vi, CheckStepPosit.vi, Startup.vi e Transduce.vi.

Na biblioteca UserInterface.llb

Panel 1AVB76AT.VI				
This VI ist the main HMI with the Technician.				
Front Panel Elements:				
Control #1	(i16)	THROTTLE [degree]		
Control #2	(i16)	RPM		
Control #3	(i16)	Air Pressure (MAP) [mbar]		
Control #4	(i16)	Air Temp [°C]		
Control #5	(i16)	H2O Temp [°C]		
Control #6	(bool)	KNOCK		
Control #7	(bool)	Air Conditioning Switch		
Control #8	(bool)	Key on		
Control #9	(sgl)	Oxygen Control		
Control #10	(cluster)	Panel Input Data		
	Control #10.1	(u32)	Digital Inputs	
	Control #10.2	(dbl)	Advance Angle	
	Control #10.3	(dbl)	Dwell Angle	
	Control #10.4	(dbl)	Cannister	
	Control #10.5	(u32)	Stepper Motor	
	Control #10.6	(array)	Injectors	Array of (sgl)
	Control #10.7	(dbl)	Battery	
Control #11	(bool)	Setup Panel?		
Control #12	(path)	Panel Config File Path		
Control #13	(i16)	Temporary max step number		
Control #14	(bool)	Hold Output		
Control #15	(bool)	Run Scipt		
Control #16	(bool)	Inj 1		
Control #17	(bool)	Inj 2		
Control #18	(bool)	Inj 3		
Control #19	(bool)	Inj 4		

Control #20	(u8)	Injectors Choice	
Control #21	(i8)	Trig Adj	
Control #22	(bool)	Wired	Air Pressure (MAP)
Control #23	(bool)	Wired	Air Temp
Control #24	(bool)	Wired	H2O Temp
Control #25	(bool)	Wired	THROTTLE
Control #26	(bool)	Wired	KNOCK
Control #27	(bool)	Wired	RPM
Control #28	(bool)	Information	
Control #29	(bool)	Power	
Control #30	(u16)	Mode	
Control #31	(bool)	RPM Up	
Control #32	(bool)	RPM Down	
Control #33	(bool)	Show Ignition(F)	
Indicator #1	(sgl)	Advance Angle	
Indicator #2	(bool)	Air Conditioning Relay	
Indicator #3	(u32)	Stepper Motor	
Indicator #4	(array)	Fuel Injectors 1,2,3 and 4 (A x ms)	Array of (sgl)
Indicator #5	(array)	Oxygen Sensor	Array of (sgl)
Indicator #6	(cluster)	Panel Output Data	

Indicator #6.1	(bool)	Panel Power	
Indicator #6.2	(u32)	Digital Outputs	
Indicator #6.3	(array)	Transducers	Array of (sgl)
Indicator #6.4	(i8)	Selected Power Actuator	
Indicator #6.5	(i16)	RPM	

	Indicator #6.6	(bool)	KNOCK		
	Indicator #6.7	(u16)	Lambda Mode		
	Indicator #6.8	(i8)	Trigger Adjust		
Indicator #7	(bool)	Fatal error			
Indicator #8	(string)	Error message			
Indicator #9	(array)	Transducers Data			
	Indicator #9.1	(cluster)			
		Indicator #9.1.1	(u16)	Sensor Type	
		Indicator #9.1.2	(array)	Values Array 1	Array of (dbl)
		Indicator #9.1.3	(array)	Values Array 2	Array of (dbl)
		Indicator #9.1.4	(dbl)	Max Value	
		Indicator #9.1.5	(dbl)	Min Value	
Indicator #10	(array)	Actuators II Data			
	Indicator #10.1	(cluster)			
		Indicator #10.1.1	(u16)	Actuator Type	
		Indicator #10.1.2	(array)	Values Array 1	Array of (dbl)
		Indicator #10.1.3	(array)	Values Array 2	Array of (dbl)
		Indicator #10.1.4	(dbl)	Max Value	
		Indicator #10.1.5	(dbl)	Min Value	
Indicator #11	(cluster)	Idle Speed Data			
	Indicator #11.1	(u16)	Control Type		
	Indicator #11.2	(array)	Forward Step Phase Sequence	Array of (u8)	
Indicator #12	(sgl)	Battery [V]			
Indicator #13	(bool)	Pump			
Indicator #14	(bool)	Power Latch			
Indicator #15	(sgl)	Cannister (SLPM)			

Panel 1AVB76AT_3.VI

This VI ist the main HMI with the Technician.

Front Panel Elements:

Control #1	(i16)	THROTTLE [degree]		
Control #2	(i16)	RPM		
Control #3	(i16)	Air Pressure (MAP) [mbar]		
Control #4	(i16)	Air Temp [°C]		
Control #5	(i16)	H2O Temp [°C]		
Control #6	(bool)	KNOCK		
Control #7	(bool)	Air Conditioning Switch		
Control #8	(bool)	Key on		
Control #9	(sgl)	Oxygen Control		
Control #10	(cluster)	Panel Input Data		
	Control #10.1	(u32)	Digital Inputs	
	Control #10.2	(dbl)	Advance Angle	
	Control #10.3	(dbl)	Dwell Angle	
	Control #10.4	(dbl)	Cannister	
	Control #10.5	(u32)	Stepper Motor	
	Control #10.6	(array)	Injectors	Array of (sgl)
	Control #10.7	(dbl)	Battery	
Control #11	(bool)	Setup Panel?		
Control #12	(path)	Panel Config File Path		
Control #13	(i16)	Temporary max step number		
Control #14	(bool)	Hold Output		
Control #15	(bool)	Run Scipt		
Control #16	(bool)	Inj 1		
Control #17	(bool)	Inj 2		
Control #18	(bool)	Inj 3		
Control #19	(bool)	Inj 4		

Control #20	(u8)	Injectors Choice	
Control #21	(i8)	Trig Adj	
Control #22	(bool)	Wired	Air Pressure (MAP)
Control #23	(bool)	Wired	Air Temp
Control #24	(bool)	Wired	H2O Temp
Control #25	(bool)	Wired	THROTTLE
Control #26	(bool)	Wired	KNOCK
Control #27	(bool)	Wired	RPM
Control #28	(bool)	Information	
Control #29	(bool)	Power	
Control #30	(u16)	Mode	
Control #31	(bool)	RPM Up	
Control #32	(bool)	RPM Down	
Control #33	(bool)	Show Ignition(F)	
Indicator #1	(sgl)	Advance Angle	
Indicator #2	(bool)	Air Conditioning Relay	
Indicator #3	(u32)	Stepper Motor	
Indicator #4	(array)	Fuel Injectors 1,2,3 and 4 (A x ms)	Array of (sgl)
Indicator #5	(array)	Oxygen Sensor	Array of (sgl)
Indicator #6	(cluster)	Panel Output Data	

Indicator #6.1	(bool)	Panel Power	
Indicator #6.2	(u32)	Digital Outputs	
Indicator #6.3	(array)	Transducers	Array of (sgl)
Indicator #6.4	(i8)	Selected Power Actuator	
Indicator #6.5	(i16)	RPM	

Indicator #6.6	(bool)	KNOCK		
Indicator #6.7	(u16)	Lambda Mode		
Indicator #6.8	(i8)	Trigger Adjust		
Indicator #7	(bool)	Fatal error		
Indicator #8	(string)	Error message		
Indicator #9	(array)	Transducers Data		
Indicator #9.1	(cluster)			
Indicator #9.1.1	(u16)	Sensor Type		
Indicator #9.1.2	(array)	Values Array 1	Array of (dbl)	
Indicator #9.1.3	(array)	Values Array 2	Array of (dbl)	
Indicator #9.1.4	(dbl)	Max Value		
Indicator #9.1.5	(dbl)	Min Value		
Indicator #10	(array)	Actuators II Data		
Indicator #10.1	(cluster)			
Indicator #10.1.1	(u16)	Actuator Type		
Indicator #10.1.2	(array)	Values Array 1	Array of (dbl)	
Indicator #10.1.3	(array)	Values Array 2	Array of (dbl)	
Indicator #10.1.4	(dbl)	Max Value		
Indicator #10.1.5	(dbl)	Min Value		
Indicator #11	(cluster)	Idle Speed Data		
Indicator #11.1	(u16)	Control Type		
Indicator #11.2	(array)	Forward Step Phase Sequence	Array of (u8)	
Indicator #12	(sgl)	Battery [V]		
Indicator #13	(bool)	Pump		
Indicator #14	(bool)	Power Latch		
Indicator #15	(sgl)	Cannister (SLPM)		

Panel GetConfig.VI

Front Panel Elements:				
Control #1	(path)	Config File		
Indicator #1	(array)	Transducers Data		
Indicator #1.1	(cluster)			
Indicator #1.1.1	(u16)	Sensor Type		
Indicator #1.1.2	(array)	Values Array 1	Array of (dbl)	
Indicator #1.1.3	(array)	Values Array 2	Array of (dbl)	
Indicator #1.1.4	(dbl)	Max Value		
Indicator #1.1.5	(dbl)	Min Value		
Indicator #2	(array)	Actuators II Data		
Indicator #2.1	(cluster)			
Indicator #2.1.1	(u16)	Actuator Type		
Indicator #2.1.2	(array)	Values Array 1	Array of (dbl)	
Indicator #2.1.3	(array)	Values Array 2	Array of (dbl)	
Indicator #2.1.4	(dbl)	Max Value		
Indicator #2.1.5	(dbl)	Min Value		
Indicator #3	(cluster)	Error Indicator		
Indicator #3.1	(bool)	Error Status		
Indicator #3.2	(i32)	Error Code		
Indicator #3.3	(string)	Error Source		
Indicator #4	(cluster)	Idle Speed Data		
Indicator #4.1	(u16)	Control Type		
Indicator #4.2	(array)	Forward Step Phase Sequence	Array of (u8)	

Nesta biblioteca também estão os seguintes VIs, que não serão apresentados aqui em detalhe: Panel OutValueFromTable.vi, Panel StepperMotorCalculator.vi e ZrO2 Capone.vi.