

IPBeja
INSTITUTO POLITÉCNICO
DE BEJA

Escola Superior de Tecnologia e Gestão
Mestrado em Engenharia Informática e Internet das Coisas

Modelação de Motor 2 Tempos

Vasco Miguel Guerreiro Gomes

Beja, 29 de Junho de 2025

INSTITUTO POLITÉCNICO DE BEJA
Escola Superior de Tecnologia e Gestão
Mestrado em Engenharia Informática e Internet das Coisas

Modelação de Motor 2 Tempos

Vasco Miguel Guerreiro Gomes

Orientado por :
Doutor João Paulo Barros, IPBeja

Relatório de Projeto Final - Desenvolvimento Baseado em Modelos

Índice

Índice	i
Índice de Figuras	iii
1 Introdução	1
2 Motivação para o Tema	3
3 Redes de Petri	5
4 Apresentação de Modelo	7
4.1 Rede Motor 2 Tempos	7
4.1.1 Combustível	8
4.1.2 Carburador	9
4.1.3 Motor	10
4.1.4 EGR	11
4.1.5 Centralina Motor	13
4.2 Rede Centralina	14
4.3 Código Externo	15
4.3.1 AIModel	16
4.3.2 ECUController	17
4.3.3 ValicacoesMotor	18
5 Resultados do Modelo	19
6 Conclusão	23
Bibliografia	25

Índice de Figuras

2.1	Motor 2 Tempos	3
3.1	Exemplo de uma Rede de Petri	6
4.1	Rede Motor 2 Tempos	8
4.2	Combustível	9
4.3	Carburador	9
4.4	Motor 2 Tempos	11
4.5	EGR	12
4.6	Centralina	14
4.7	Rede Centralina Motor	15
4.8	Print Decisão ECU no Renew	15
4.9	Classe Java AIModel	16
4.10	Classe Java ECUController	17
4.11	Classe Java ValidacoesMotor	18
5.1	Execução da Centralina Motor	19
5.2	Execução Centralina - Motor 2 Tempos	20
5.3	Execução Carburador - Motor 2 Tempos	21
5.4	Execução Mistura no Carburador - Motor 2 Tempos	21
5.5	Execução Motor - Rede Motor 2 Tempos	22

Capítulo 1

Introdução

Este projeto tem como base realizar a modelação de um motor a 2 tempos, com recurso a redes de *Petri*, também conhecidas como redes coloridas.

Para isso vai ser utilizado o *software Renew* apresentado e trabalhado nas aulas da unidade curricular para que seja possível construir os modelos

Este relatório está dividido nos seguintes capítulos:

- **Motivação para o Tema** - Este capítulo é onde vai ser explicada a motivação pelo qual eu escolhi esta temática para o desenvolvimento do projeto;
- **Redes de *Petri*** - Explicação acerca das Redes de *Petri*;
- **Apresentação do Modelo** - Vai servir para apresentar o modelo desenvolvido;
- **Resultados do Modelo** - Pretende demonstrar as funcionalidades do modelo;
- **Conclusão** - Tirar conclusões acerca do projeto e melhorias futuras;

Capítulo 2

Motivação para o Tema

A motivação para esta temática surgiu do meu interesse em mecânica automóvel algo que sempre gostei e apreciei.

O facto de estar a aplicar conhecimentos abstratos a uma temática que percebo torna bastante mais simples o processo de modelação de uma rede de *Petri* que tem como objetivo desenhar, modelar, assim como executar redes de *Petri* de alto nível[Lak70], uma vez que se conhece todos os pormenores do negócio do projeto.

Adiciono ainda que a minha dissertação de mestrado é acerca desta temática ajuda bastante a perceber o funcionamento de um motor de combustão interna a 2 tempos como podemos ver na figura 2.1, assim como a aplicar os conhecimentos transmitidos em aula.

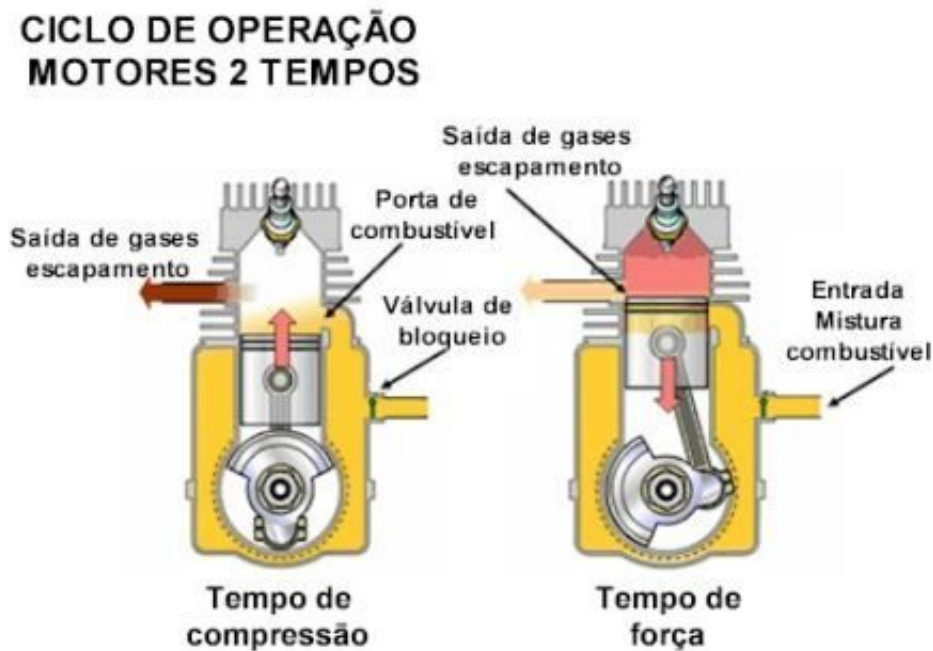


Figura 2.1: Motor 2 Tempos

Capítulo 3

Redes de Petri

Antes de falar sobre o projeto em si é necessário realizar uma introdução às redes de *Petri* faladas em aula.

Estas redes tem como objetivo representar graficamente no caso do *Renew*, ou modelar sistemas, qualquer que seja o sistema, o *Renew* utiliza *PNML* ou *Petri Net Markup Language* para representar redes *XML* num formato gráfico, sendo que existem outros[Bil+03].

Podemos observar na figura 3.1, criando assim um *Workflow* dando uma perspectiva geral do funcionamento do sistema até ao mais ínfimo detalhe[Aal98].

Para isso estas redes à semelhança das linguagens de programação onde se escreve código e temos de respeitar a sintaxe para que tudo funcione, nas redes de *Petri* possuímos um formalismo gráfico composto por:

- **Lugares** - Representados por círculos que possuem as marcas;
- **Transições** - Representados por retângulos, que descrevem os eventos a ocorrer na rede;
- **Marcas** - Representam o estado do negócio que queremos modelar através da rede;
- **Arcos** - Representados por setas, estes têm como objetivo conectar as transições às marcas e vice-versa e ainda indicar o fluxo da rede transportando as marcas;

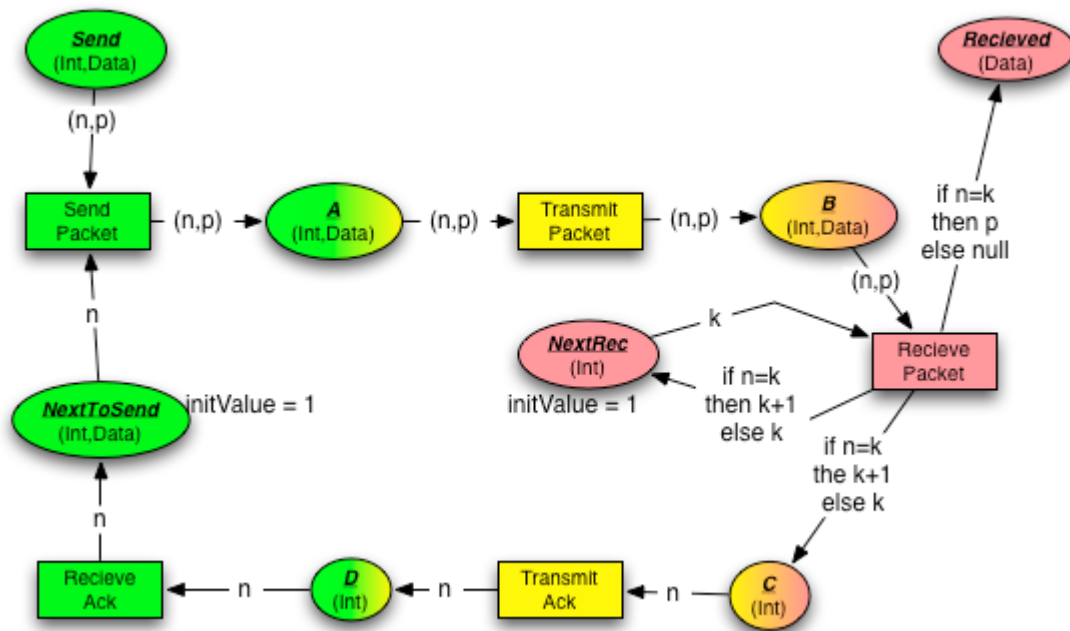


Figura 3.1: Exemplo de uma Rede de Petri

Estas redes permitem ainda modelar estados, eventos, condições, sincronizações, paralelismo, escolha e iterações, porém as redes podem ser algo longas e extensas de tentarmos modelar um sistema à letra.

Existem algumas regras para que a rede possa funcionar bem, a tal denominada sintaxe que fiz analogia mais acima.

Essas regras incluem por exemplo [Aal98]:

- Os lugares podem estar vazios, neste caso sem marcas;
- Se a transição tiver vários arcos, esta só dispara se todos os lugares tiverem marcas para disparar;
- Podem ser efetuadas chamadas a outras transições da rede através da sintaxe adequada;
- Podem ser efetuadas guardas nas transições para impedir que a transição dispare sem controle;
- As marcas podem ser vários tipos de variáveis desde Strings a tupulos de dados;

Capítulo 4

Apresentação de Modelo

O modelo presente neste trabalho é constituído por 2 redes, sendo estas a **Motor 2 Tempos** e ainda a **Centralina Motor**, sendo ainda necessário correr classes *Java* de código externas para que funcione na sua totalidade.

Esta primeira rede contém todos os componentes, sejam estes lugares, marcas, transições e arcos necessários para que seja simulado o funcionamento de um motor a combustão, desde receber o vídeo e enviar para a centralina para ser processado e esta devolver as instruções para os diferentes módulos executarem, a operar a válvula *EGR* com os dados recebidos da centralina ou a bombear combustível.

Por sua vez a segunda rede demonstra a centralina do motor em execução que faz recurso de código externo para simular que o vídeo enviado foi processado por um modelo de Inteligência Artificial e de seguida consoante o resultado faz a decisão de que métodos deve executar, por último envia os resultados de volta para a rede do motor.

4.1 Rede Motor 2 Tempos

A primeira rede a ser apresentada é a rede principal a do motor que vai chamar a centralina e utilizar os dados provenientes da mesma, como é possível verificar na figura 4.1

4. APRESENTAÇÃO DE MODELO

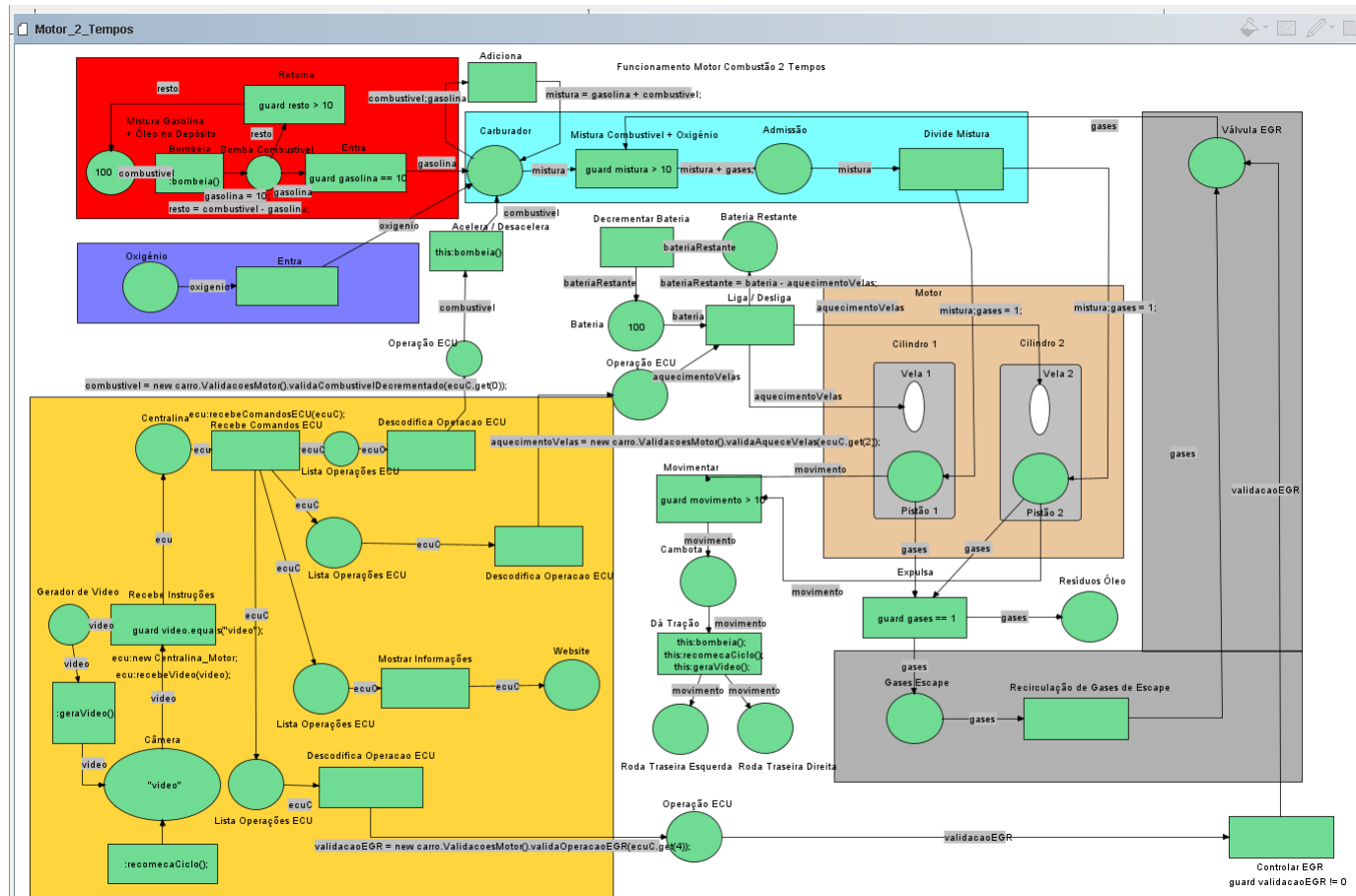


Figura 4.1: Rede Motor 2 Tempos

4.1.1 Combustível

Na figura 4.2, é possível verificar o sistema de bombeamento de combustível até ao carburador, onde existe um lugar que contém os mililitros de combustível totais no depósito, que é passada através de uma variável chamada de combustível.

Para que esse combustível chegue até ao carburador, este é necessário ser bombeado, ação essa que acontece após a centralina enviar os dados de quanto combustível deve ser injetado, uma vez que se acelerar deve ser injetado mais e caso esteja a abrandar deve ser menos, e que apenas é disparada quando a transição de acelerar/desacelerar dá orem para tal através da sintaxe do *Renew* para chamar funções dentro da mesma rede.

Por sua vez, seguem 10 mililitros de combustível bombeado para o carburador, através da guarda colocada na ação de entrada e os restantes 90 vão retornar ao carburador com recurso a uma guarda que verifica que o resto é maior que 10 para garantir que funciona tudo como deve, resto esse que é declarado numa variável num arco.

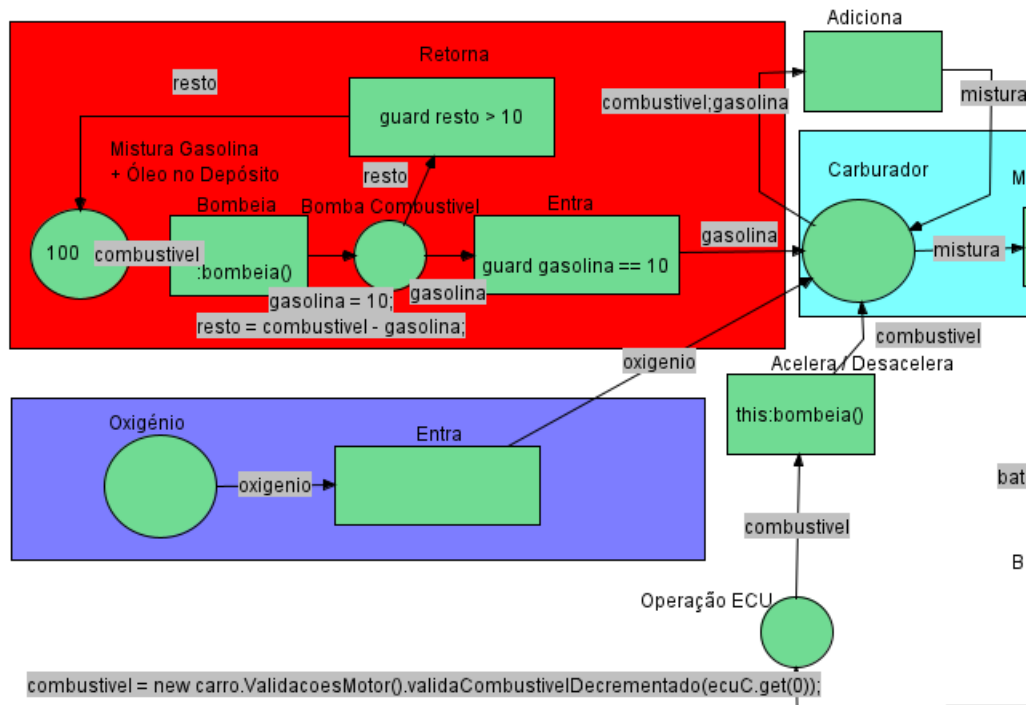


Figura 4.2: Combustível

4.1.2 Carburador

Na figura 4.3, podemos visualizar o funcionamento do carburador, que realiza a mistura entre o oxigênio com a gasolina e ainda os gases recirculados pela válvula *EGR*.

Esta mistura foi feita com recurso a uma transição que pega nos 10 mililitros de combustível que vieram bombeados do depósito e os soma ao que a centralina ordenou adicionar mais ou menos consoante aceleração ou desaceleração, onde para ter a certeza que essa transição é executada antes de seguir para o lugar de admissão, existe uma guarda na transição da mistura que apenas permite que esta passe caso seja superior a 10 mililitros uma vez que esse valor é sempre fixo é garantia que a transição de adição é sempre executada primeiro fazendo com que o valor da mistura seja sempre maior a 10.

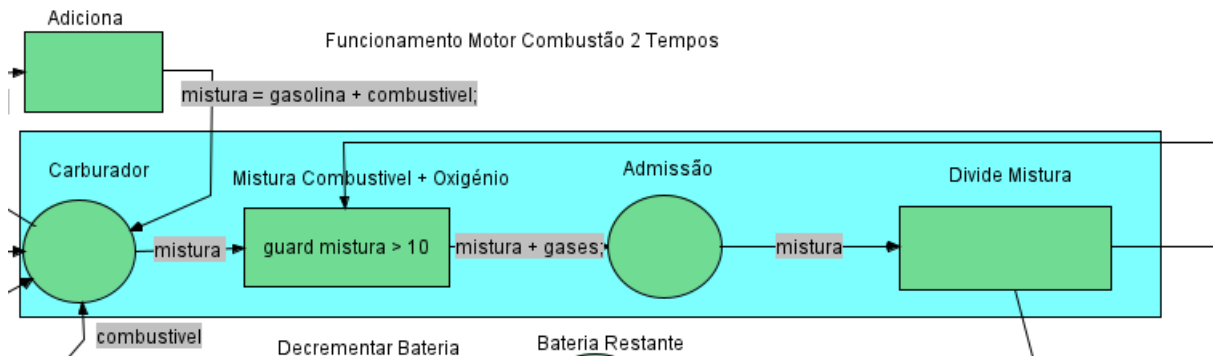


Figura 4.3: Carburador

4.1.3 Motor

Seguidamente na figura 4.4, é possível visualizar o funcionamento do motor a 2 tempos implementado, onde se começamos a análise da esquerda para a direita de cima para baixo podemos observar que existe um lugar denominado de bateria que possui uma marca de 100, sendo esta referente a uma bateria totalmente carregada que é ligada ou desligada consoante a ação que vêm da centralina, e uma vez mais o restante volta ao lugar inicial com recurso a um decrementador de bateria.

Estes lugares e transições adicionais foram utilizados para retirar um parte do valor inicial da marca original sendo que quando é disparado normalmente e não exista um sistema de recuperação se valores, este retira o valor todo, deixando o lugar original vazio, o que seria estranho devido a não ser possível de gastar a bateria toda num único disparo, assim como acontece no caso do bombeamento de combustível demonstrado anteriormente.

Seguindo os arcos, podemos observar que as velas passam a aquecer ou não consoante o que a centralina devolve e que uma vez que os pistões recebam a mistura de combustível com o oxigénio, é disparado o movimento da cambota do motor fazendo de seguida movimentar as rodas, e quando dá tração às rodas, esta transição chama as funções de bombear combustível, recomeço de ciclo de captura de vídeo e a geração de vídeo para que o modelo continue a correr enquanto tiver condições para o fazer.

Por fim desta parte do modelo paralelamente ao movimento da cambota temos ainda a captação de resíduos de óleo gerados pelo motor, uma vez que este necessita de óleo para realizar a sua lubrificação e devido ao funcionamento do mesmo este mistura-se com o combustível, e ainda a captação dos gases de escape para serem utilizados pelo sistema *EGR* de modo a conseguir mitigar a poluição do meio envolvente.

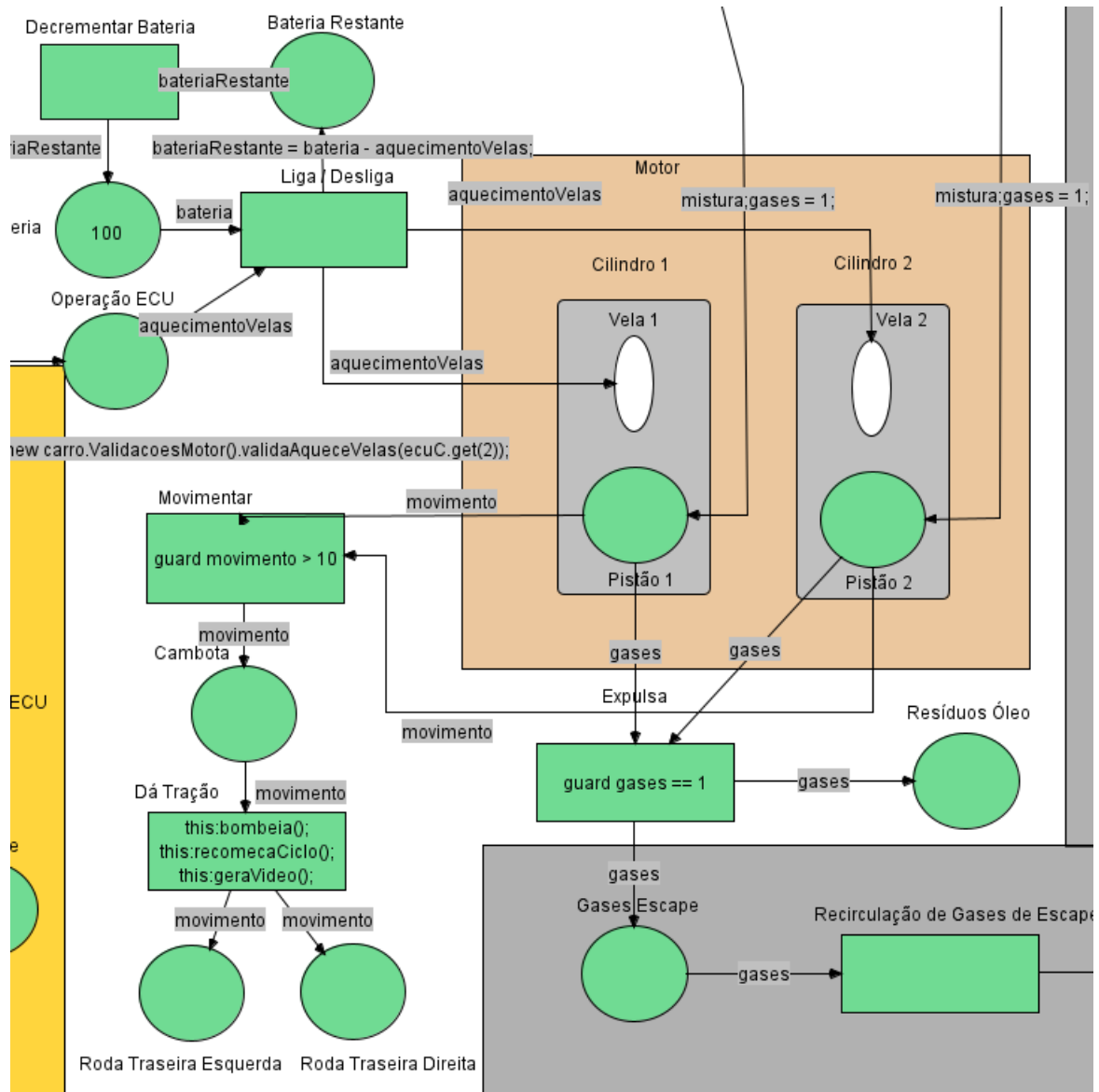


Figura 4.4: Motor 2 Tempos

4.1.4 EGR

Após visualizarmos a combustão do motor a 2 tempos, faz sentido seguirmos o sentido dos gases de escape e verificar como está montada a parte da rede dedicada a tratamento dos gases de escape pelo sistema de *EGR* acupulado ao motor, como pode ser visualizado na figura 4.5.

Este sistema permite realizar a recirculação de gases de escape para a admissão de modo a realizar a redução de *NOX*, ou também conhecidos como Óxidos Nitrosos[LL22] que é um conjunto de gases prejudiciais à saúde humana[GSL21; Xio+19].

Para que isso seja possível é necessário a introdução de *CO2* obtidos da combustão da

4. APRESENTAÇÃO DE MODELO

gasolina serem recirculados para a admissão diminuindo os gases de efeito estufa.

No caso deste modelo temos 2 opções ora ela está aberta e passam gases para a admissão, ora está fechada como por exemplo no momento de aceleração do veículo permitindo-o obter um melhor rendimento uma vez que não está a ser sufocado, no modelo isso é representado adicionando 1 marca à mistura de combustível após os pistões dispararem pela primeira vez, significando que existe uma combustão limpa na primeira vez que o motor trabalha.

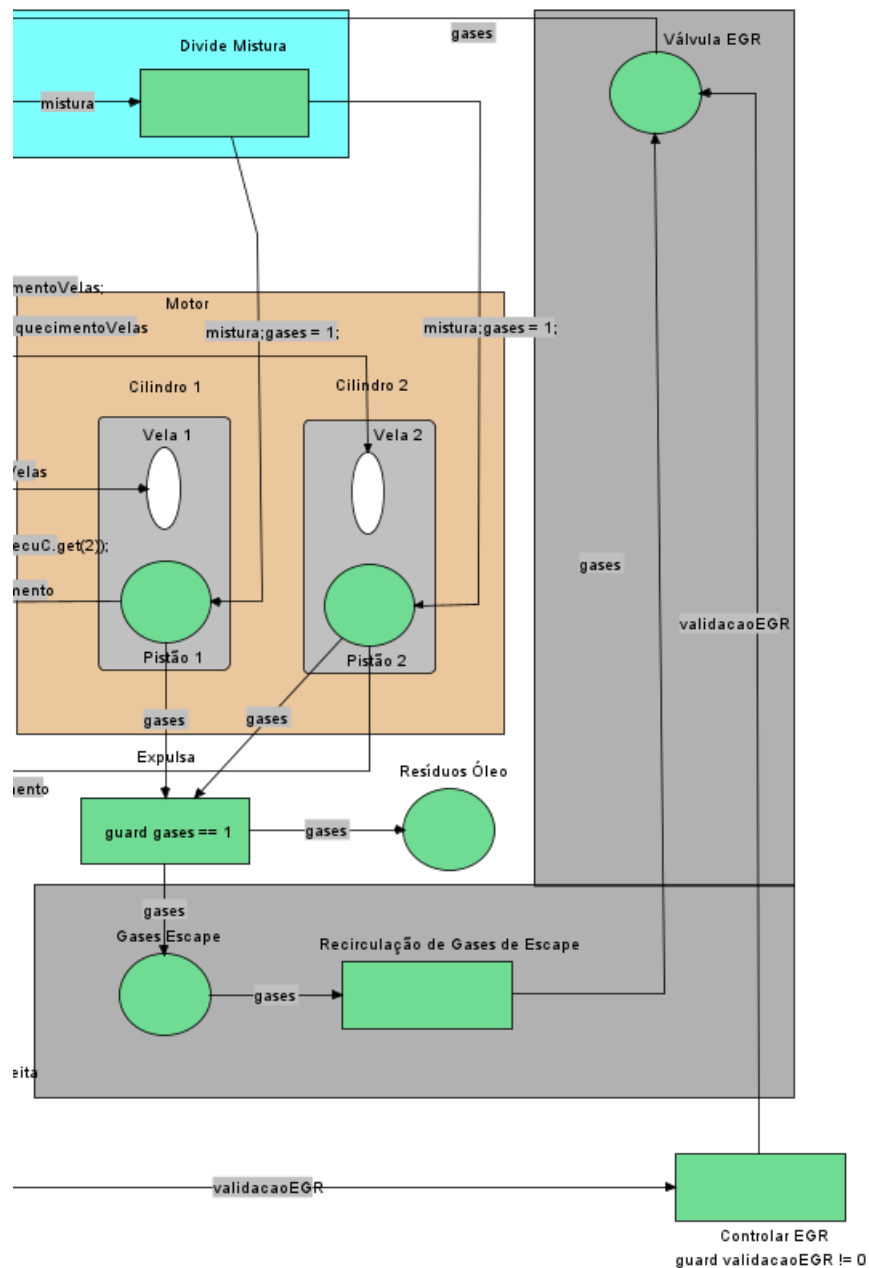


Figura 4.5: EGR

4.1.5 Centralina Motor

Por fim da rede **Motor 2 Tempos** falta apenas visualizar a centralina que pode ser visualizada na figura 4.6, é possível observar que é chamada uma outra rede apelidada de **Centralina Motor**, pela qual é enviado o vídeo capturado pela câmara na transição de **Receber Comandos ECU**, esta tanto envia como também recebe os dados vindos da outra rede, sendo estes últimos representados pela variável **ecuC** que por sua vez é repartida pelas diferentes operações possíveis no motor como **Acelerar ou Desacelerar o Carburador**, **Aquecer as Velas**, **Ativar a Válvula EGR**, ou mesmo ainda **Mostrar as Informações no Website**.

Mas para que as operações corretas possam ser efetuadas é necessário antes existir uma decodificação dos dados da *ECU*, uma vez que este vêm num tupulo de dados, onde pode ser visto a ser chamada uma classe de código externo denominada de **ValidacoesMotor** que retira a opção correta para cada uma das operações, esta necessidade de chamar código externo deveu-se à necessidade de realizar um **If Ternário** que no *Renew* não é aceite, tendo a situação que ser contornada de outra maneira.

Após isso as informações são colocadas em variáveis de modo a serem passadas às suas transições, a única exceção é o website que faz sentido que mostre todas as informações num *Dashboard* por exemplo.

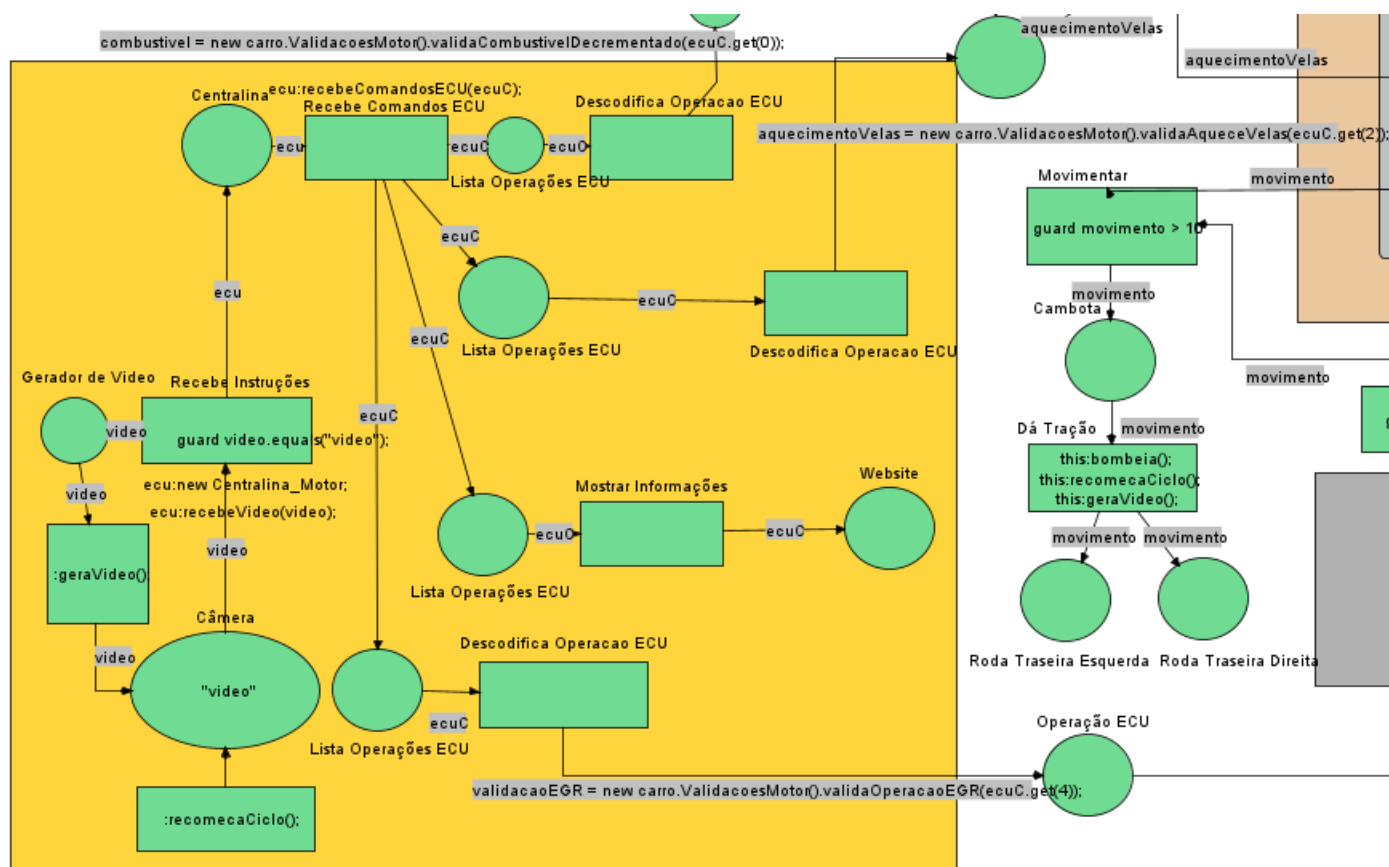


Figura 4.6: Centralina

4.2 Rede Centralina

De seguida, temos a segunda rede que pode ser visualizada na figura 4.7, rede esta que pretende representar uma centralina caseira feita através de um *Raspberry Pi* que contém um *Broker MQTT* para apanhar as comunicações que lhe são enviadas.

Após a centralina receber energia e receber o vídeo é feita uma chamada a código externo *Java* através da transição **Decidir Operações a Executar**, que cria uma instância da classe **ECUController** e chama a função **decideOperacaoAExecutar** passando o vídeo capturado como parâmetro e retornando o seu valor para a variável *result* que pode ser visualizado na figura 4.8 que é passado para a rede do **Motor de 2 Tempos** através da transição de **Enviar Comandos para o Motor**.

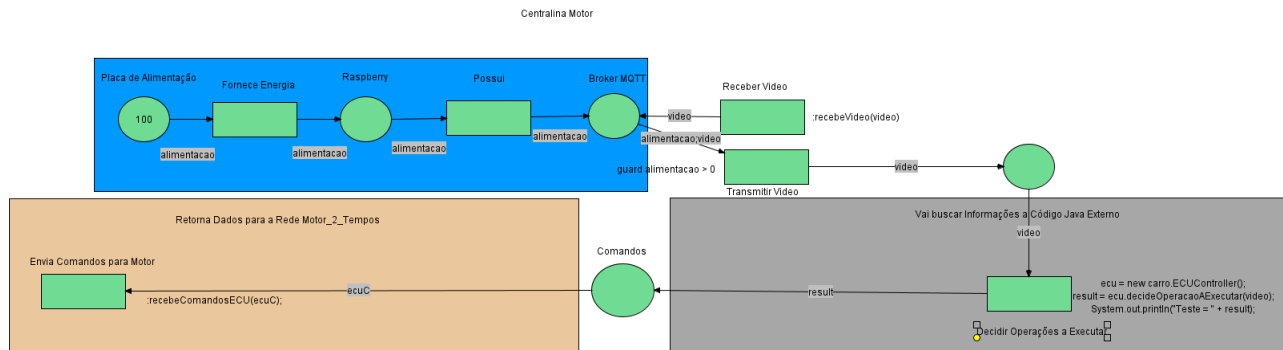


Figura 4.7: Rede Centralina Motor

```

MINGW64:/e/Universidade_Stuff/Mestrado_IoT/1_Ano/2_Semestre/Desenvolvimento_Baseado_Modelos/Renew...
INFO: no additional plugin locations set.
INFO: loading plugins...
Renew > INFO: loaded plugin: Renew Util
INFO: loaded plugin: Renew Simulator
INFO: loaded plugin: Renew Formalism
INFO: loaded plugin: Renew Misc
INFO: loaded plugin: Renew PTChannel
INFO: loaded plugin: Renew Remote
INFO: loaded plugin: Renew Window Management
INFO: loaded plugin: Renew JHotDraw
INFO: loaded plugin: Renew Gui
INFO: loaded plugin: Renew Formalism Gui
INFO: loaded plugin: Renew Logging
INFO: loaded plugin: Renew NetComponents
INFO: loaded plugin: Renew Console
INFO: loaded plugin: Renew FreeHep Export
INFO: loaded plugin: Renew Navigator
INFO: loaded plugin: Renew SplashScreen
Opening gui...
Passing args to gui...
INFO: Using default concurrent simulator ...
Resultado ECU = [Acelerar_Carburador, Andar, Liga_Bateria, Mostrar_Informacoes, Aciona_EGR]
Resultado ECU = [Desacelerar_Carburador, Parar, Desliga_Bateria, Mostrar_Informacoes, Aciona_EGR]

```

Figura 4.8: Print Decisão ECU no Renew

4.3 Código Externo

De seguida segue-se o código externo que é chamado pela **Centralina** e pelo **Motor 2 Tempos**, este último para contornar dificuldades encontradas no *Renew*, resolvendo-as assim com a execução de código externo, como é possível verificar em várias secções do modelo explicadas anteriormente.

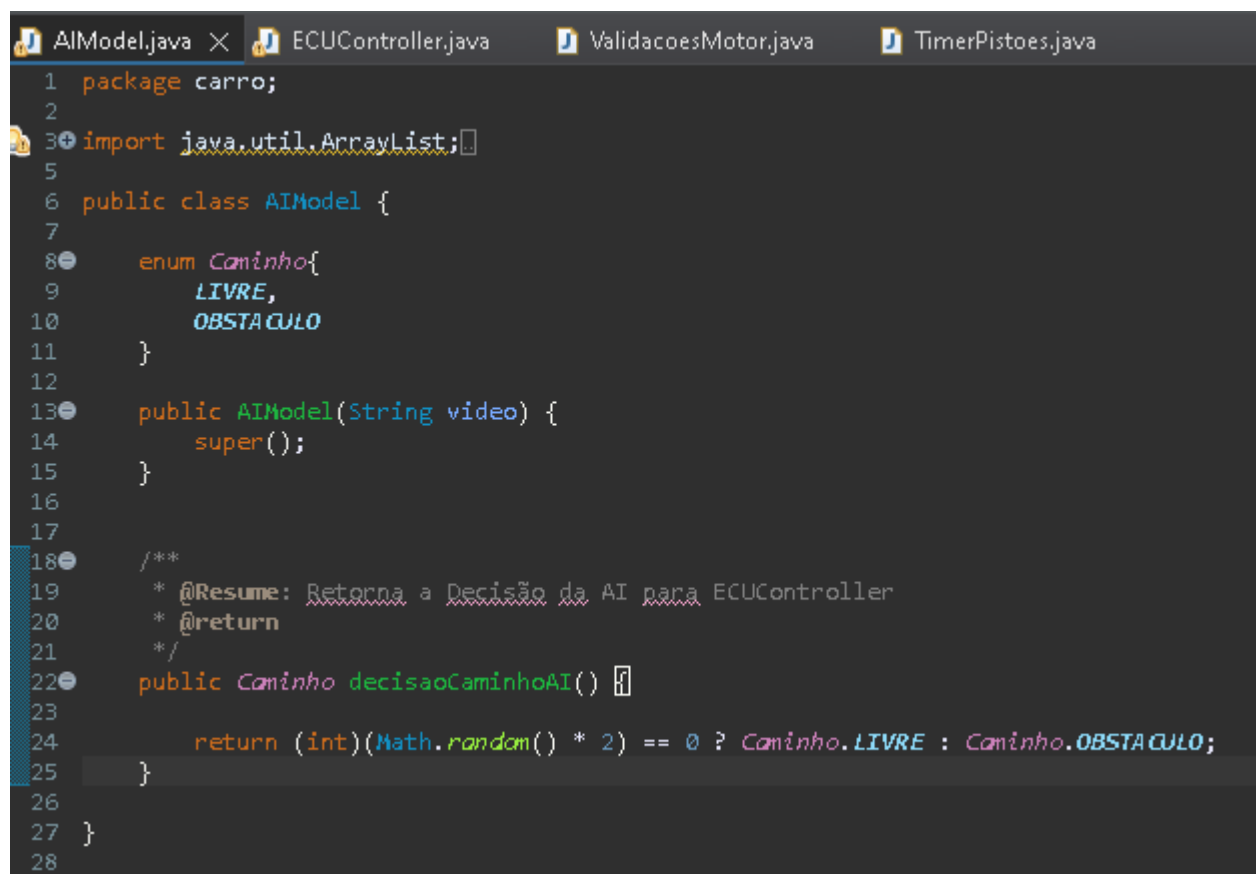
Vale a pena ainda salientar que a maneira de importar o código externo para o *Renew* é diferente da maneira mostrada em aula, no sentido que em contexto de aula foi-nos mostrado que poderíamos utilizar *stubs* para gerar código *Java*, porém estes *stubs* são também limitados uma vez que estes métodos apenas são do tipo *void* não devolvendo nenhum tipo de resultados.

Uma vez essa limitação baseie-me num código dado em aula pelo docente da unidade curricular que era apenas uma classe externa de código *Java* que era chamada pela rede, com a exceção de que esta era compilada antes de que o *Renew* fosse executado.

Uma vez que eu sabia que o *Renew* carregava os seus próprios *Jar's* para permitir a aplicação funcionar corretamente eu pensei se carrega os do *Renew* também pode carregar um *Jar* customizado, de código que eu queira executar, para isso bastou-me alterar o ficheiro *renew-s* dentro da pasta *scripts* para ir buscar o *Jar* denominado de *ECUController New* e uma vez importado o *Jar* foi apenas necessário chamar as classes *Java* que este trazia para puder tirar partido do código desenvolvido mostrado em seguida.

4.3.1 AIModel

Podemos começar por visualizar a classe **AIModel**, como mostra a figura 4.9, classe esta que permite simular uma Inteligência Artificial a tomar decisões, algo sugerido pelo docente da unidade curricular num formato de rede no *Renew*, acabei por achar que fazia mais sentido incorporar numa classe que retorna um número random entre 0 e 1 que através do Enum mostra se o caminho está Livre ou se existe um Obstáculo no caminho.

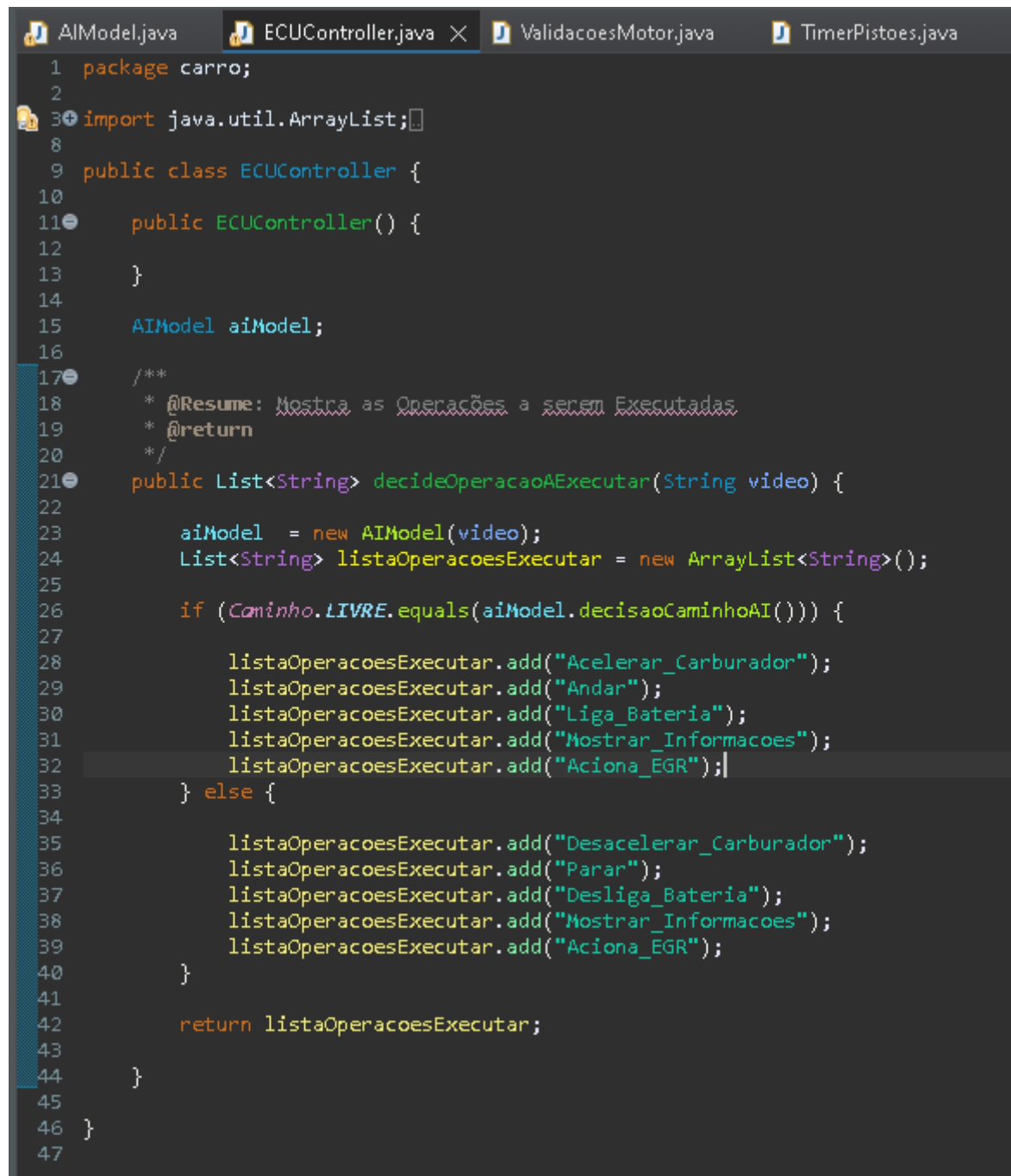


```
1 package carro;
2
3 import java.util.ArrayList;
4
5
6 public class AIModel {
7
8     enum Caminho{
9         LIVRE,
10        OBSTACULO
11    }
12
13    public AIModel(String video) {
14        super();
15    }
16
17
18    /**
19     * @Resume: Retorna a Decisão da AI para ECUController
20     * @return
21     */
22    public Caminho decisaoCaminhoAI() {
23
24        return (int)(Math.random() * 2) == 0 ? Caminho.LIVRE : Caminho.OBSTACULO;
25    }
26
27 }
28
```

Figura 4.9: Classe Java AIModel

4.3.2 ECUController

Segue-se a classe **ECUController** na figura 4.10 que dependendo do resultado gerado pela classe **AIModel** mostrada anteriormente gera uma lista de operações a serem executadas pelo motor, como por exemplo se o caminho estiver livre, o motor pode ser acelerado, permitindo-o andar, o inverso acontece caso o caminho tenha um obstáculo.



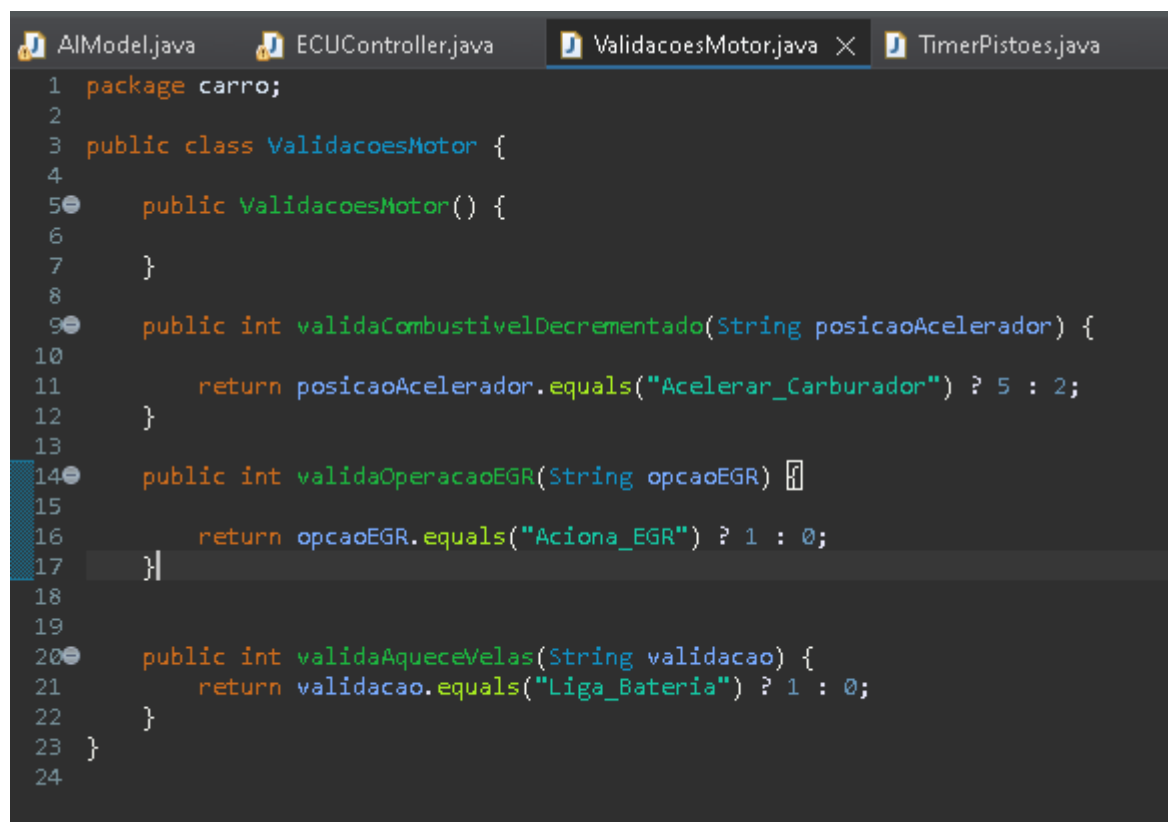
```
1 package carro;
2
3 import java.util.ArrayList;
4
5
6
7
8
9 public class ECUController {
10
11     public ECUController() {
12
13     }
14
15     AIModel aiModel;
16
17     /**
18      * @Resume: Mostra as Operações a serem Executadas
19      * @return
20      */
21     public List<String> decideOperacaoAExecutar(String video) {
22
23         aiModel = new AIModel(video);
24         List<String> listaOperacoesExecutar = new ArrayList<String>();
25
26         if (Caminho.LIVRE.equals(aiModel.decisaoCaminhoAI())) {
27
28             listaOperacoesExecutar.add("Acelerar_Carburador");
29             listaOperacoesExecutar.add("Andar");
30             listaOperacoesExecutar.add("Liga_Bateria");
31             listaOperacoesExecutar.add("Mostrar_Informacoes");
32             listaOperacoesExecutar.add("Aciona_EGR");
33         } else {
34
35             listaOperacoesExecutar.add("Desacelerar_Carburador");
36             listaOperacoesExecutar.add("Parar");
37             listaOperacoesExecutar.add("Desliga_Bateria");
38             listaOperacoesExecutar.add("Mostrar_Informacoes");
39             listaOperacoesExecutar.add("Aciona_EGR");
40         }
41
42         return listaOperacoesExecutar;
43     }
44 }
45
46
47
```

Figura 4.10: Classe Java ECUController

4.3.3 ValicacoesMotor

Por fim a classe **ValidacoesMotor** que pode ser visualizado na figura 4.11, classe esta que ajuda a ultrapassar o problema do *Renew* não permitir utilizar a sintaxe do *If Ternário*, problema que já tinha sido referido anteriormente.

Em cada uma das 3 funções de podemos verificar que as operações efetuadas são simples *If's Ternário* que devolvem números que serão utilizados no funcionamento do motor, como por exemplo caso a centralina tenha dado ordem para o motor ser acelerado o carburador recebe mais 5 mililitros de combustível, enquanto se a ordem tiver sido de desacelerar a o carburador apenas recebe 2 mililitros de combustível.



```
1 package carro;
2
3 public class ValidacoesMotor {
4
5     public ValidacoesMotor() {
6
7     }
8
9     public int validaCombustivelDecrementado(String posicaoAcelerador) {
10
11         return posicaoAcelerador.equals("Acelerar_Carburador") ? 5 : 2;
12     }
13
14     public int validaOperacaoEGR(String opcaoEGR) {
15
16         return opcaoEGR.equals("Aciona_EGR") ? 1 : 0;
17     }
18
19
20     public int validaAqueceVelas(String validacao) {
21         return validacao.equals("Liga_Bateria") ? 1 : 0;
22     }
23 }
24
```

Figura 4.11: Classe Java ValidacoesMotor

Capítulo 5

Resultados do Modelo

Uma vez feita a apresentação do modelo, podemos passar à fase de apresentação de resultados, explorando assim algumas fases da execução do modelo.

Primeiramente na figura 5.1, pode ser visualizado os dados que vêm do código externo *Java* mostrados na secção anterior deste relatório, em que neste caso os comandos foram da existência de um obstáculo no caminho, sendo por isso mesmo o motor obrigado a abrandar o seu funcionamento.

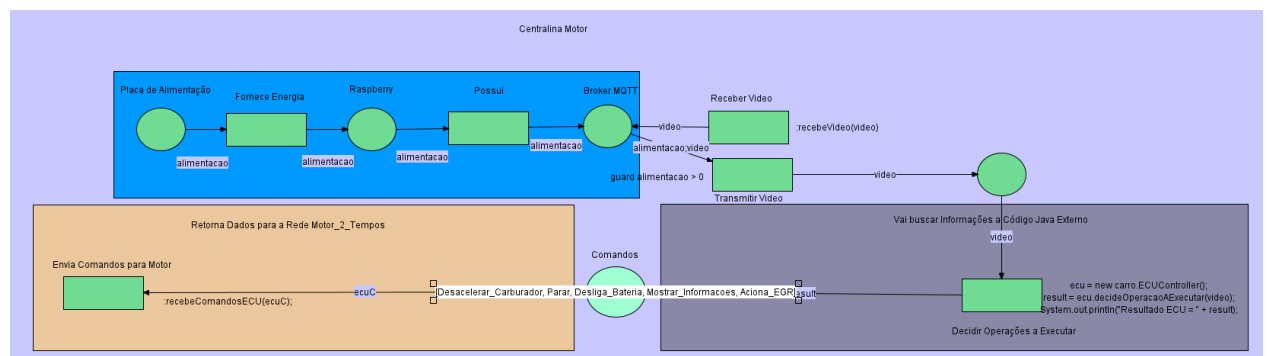


Figura 5.1: Execução da Centralina Motor

Seguidamente, faz sentido visualizar o que acontece depois no motor, e no caso da figura 5.2 é possível ver que nesta fase da execução esta já recebeu os dados, como pode ser visualizado nos lugares de **Lista de Operações de ECU**, preparando-se para fazer a decodificação das mesmas para executar os comandos enviados pela centralina.

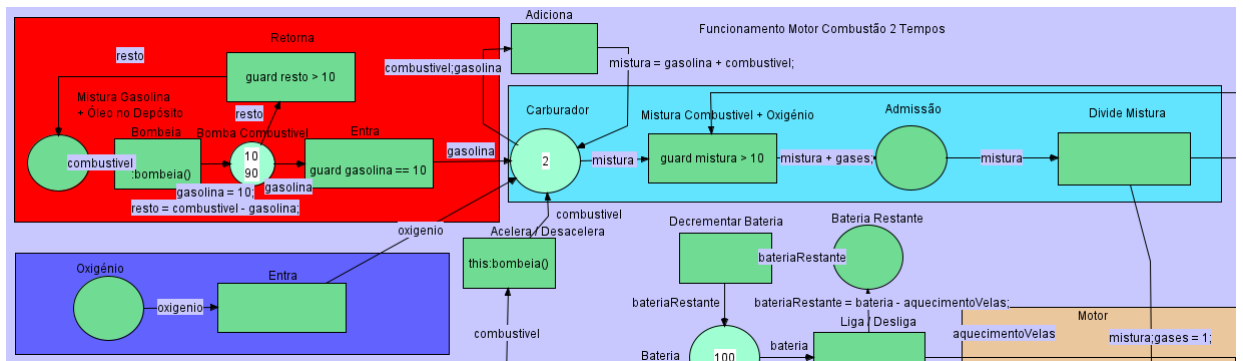


Figura 5.3: Execução Carburador - Motor 2 Tempos

Já a segunda mostra um estágio já avançado, após ser corrida a transição da adição dos 10 mililitros com os 2 requeridos pela *ECU*, podemos ainda observar que a *ECU* já contém também a marca de 1, estando pronta para juntar à mistura na admissão.

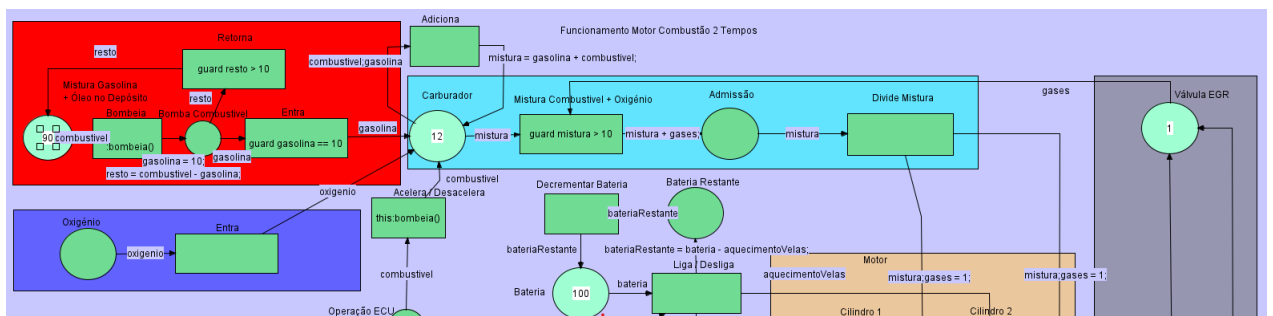


Figura 5.4: Execução Mistura no Carburador - Motor 2 Tempos

Por fim falta mostrar a execução do motor em sim em funcionamento como mostra a figura 5.5, aqui é possível verificar que o motor já trabalhou pelo menos uma vez, como pode ser visualizado nos lugares das rodas traseiras, assim como pelos resíduos de óleo.

Podemos ainda visualizar que no arco que é transmitida a mistura para os 2 pistões, os valores da mistura e dos gases aparecem repartidos, tendo os gases o valor de 1, isto acontece para que a transição de **Divisão de Mistura** dispare apenas 1 vez colocando tudo nos pistões, de modo a que quando aconteça a combustão nos mesmos seja possível fazer a transmissão do movimento da cambota às rodas e ainda os gases para os resíduos e a válvula *EGR*.

5. RESULTADOS DO MODELO

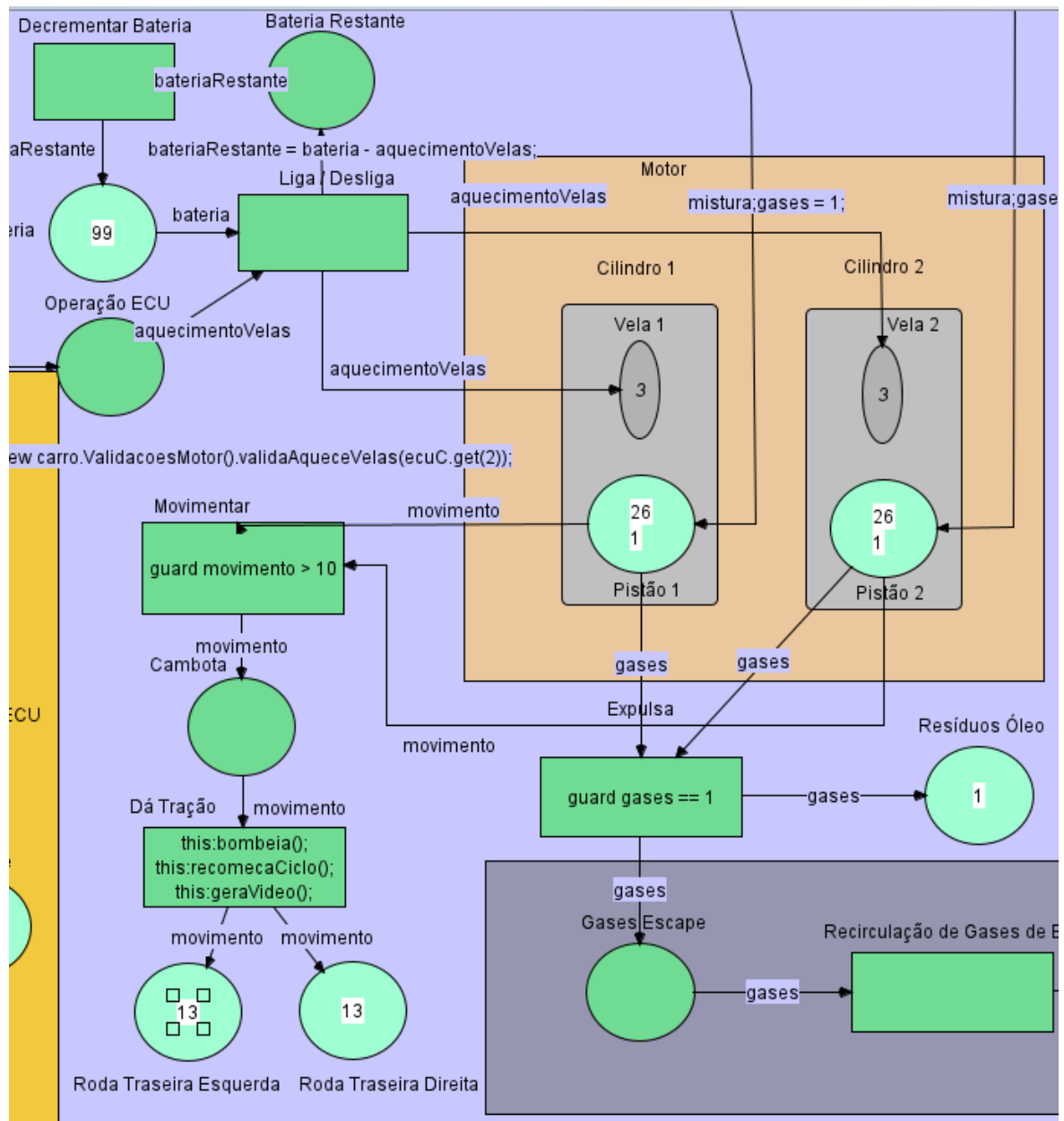


Figura 5.5: Execução Motor - Rede Motor 2 Tempos

Capítulo 6

Conclusão

Após a finalização deste projeto é possível concluir que consegui entender o propósito da utilização de redes de Petri para realizar a modelação de sistemas e testes, uma vez que é possível irmos bastantes a fundo na temática, podendo representar todo o tipo de Objetos através de Lugares, as suas propriedades através de marcas, e as suas ações através das transições, conectando até mesmo com código real e hardware.

Em termos de melhorias futuras, gostaria de estender o modelo para a temática da dissertação que é realizar um Carro Autónomo Híbrido Todo-o-Terreno modelando-o através das redes de Petri para melhor entendimento do funcionamento, utilizar mais redes para representação dos vários componentes do projeto.

Bibliografia

- [Aal98] Wil Aalst. «The Application of Petri Nets to Workflow Management». Em: *Journal of Circuits, Systems, and Computers* 8 (fev. de 1998), pp. 21–66. DOI: 10.1142/S0218126698000043 (citado nas páginas 5, 6).
- [Bil+03] Jonathan Billington et al. «The Petri Net Markup Language: Concepts, Technology and Tools». Em: jun. de 2003, pp. 483–505. ISBN: 978-3-540-40334-0. DOI: 10.1007/3-540-44919-1_31 (citado na página 5).
- [GSL21] Changming Gong, Xiankai Si e Fenghua Liu. «Combined effects of excess air ratio and EGR rate on combustion and emissions behaviors of a GDI engine with CO₂ as simulated EGR (CO₂) at low load». Em: *Fuel* 293 (jun. de 2021), p. 120442. DOI: 10.1016/j.fuel.2021.120442 (citado na página 11).
- [Lak70] Charles Lakos. «From Coloured Petri Nets to Object Petri Nets». Em: *Lecture Notes in Computer Science* 935 (fev. de 1970). DOI: 10.1007/3-540-60029-9_45 (citado na página 3).
- [LL22] Janusz Lasek e Radosław Lajnert. «On the Issues of NO_x as Greenhouse Gases: An Ongoing Discussion. . . » Em: *Applied Sciences* 12 (out. de 2022), p. 10429. DOI: 10.3390/app122010429 (citado na página 11).
- [Xio+19] Duan Xiongbo et al. «Experimental and numerical investigation of the effects of low-pressure, high-pressure and internal EGR configurations on the performance, combustion and emission characteristics in a hydrogen-enriched heavy-duty lean-burn natural gas SI engine». Em: *Energy Conversion and Management* 195 (jun. de 2019). DOI: 10.1016/j.enconman.2019.05.059 (citado na página 11).