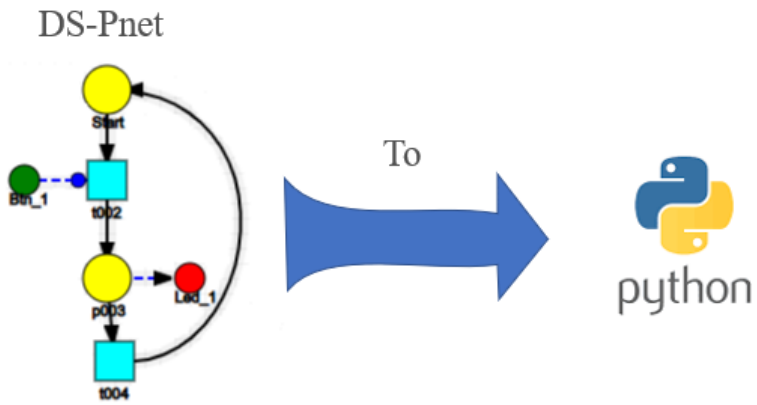
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Eletrotécnica Energia e Automação



Geração Automática de Código Python para Sistemas Embutidos/Ciber-físicos

PEDRO MIGUEL PIEGAS VALE
(Licenciado em Engenharia de Automação Controlo e Instrumentação)

Trabalho Final de Mestrado para obtenção do grau de Mestre
em Engenharia Eletrotécnica – Ramo de Automação e Eletrónica Industrial

Orientador:

Professor Doutor Fernando Joaquim Ganhão Pereira

Júri:

Presidente: Professor Doutor Hiren Canacsihn

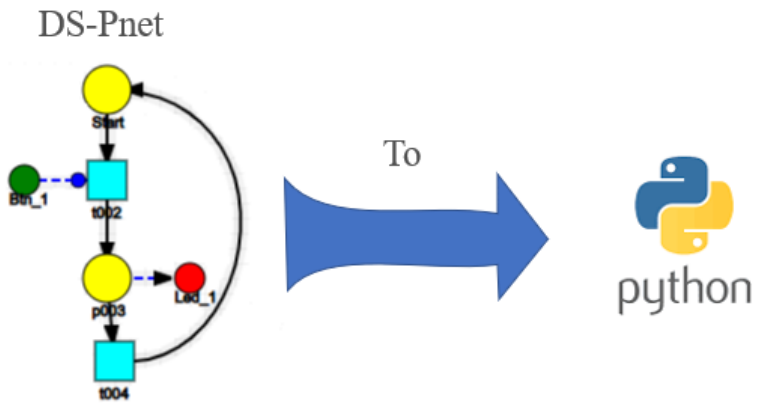
Vogais: Professor Doutor Filipe de Carvalho Moutinho

Professor Doutor Fernando Joaquim Ganhão Pereira

Fevereiro 2023

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Eletrotécnica Energia e Automação



Geração Automática de Código Python para Sistemas Embutidos /Ciber-físicos

PEDRO MIGUEL PIEGAS VALE
(Licenciado em Engenharia de Automação Controlo e Instrumentação)

Trabalho Final de Mestrado para obtenção do grau de Mestre em Engenharia Eletrotécnica – Ramo de Automação e Eletrónica Industrial

Orientador:

Professor Doutor Fernando Joaquim Ganhão Pereira

Júri:

Presidente: Professor Doutor Hiren Canacsihn

Vogais: Professor Doutor Filipe de Carvalho Moutinho

Professor Doutor Fernando Joaquim Ganhão Pereira

Fevereiro 2023

Agradecimento

A conclusão desta dissertação, representa o fim de uma das etapas mais importantes da minha vida, abrindo portas para novos desafios profissionais. Como tal, gostaria de agradecer a todas as pessoas que se envolveram direta ou indiretamente, que em muito contribuíram para a sua feitura e às quais estou eternamente grato.

Em primeiro lugar quero agradecer ao meu orientador Engenheiro Fernando Joaquim Ganhão Pereira, pela oportunidade, dedicação e constante disponibilidade que me foi prestada ao longo destes meses de trabalho. Graças à sua ajuda fui capaz de aprender novas ferramentas que me habilitaram a ultrapassar este desafio, contribuindo assim, positivamente para o meu desenvolvimento académico e profissional.

À minha família, por todo o apoio prestado ao longo de todo o meu caminho académico, demonstrando um amor incondicional fornecendo-me todas as ferramentas, para que fosse capaz de concluir desafios pessoais e académicos.

À minha namorada, que me acompanhou ao longo deste trajeto, estando sempre ao meu lado e apoiando-me.

Quero agradecer aos meus amigos pelo apoio e força que me deram durante esta difícil batalha, contribuindo sempre para a vivência de bons momentos ao longo destes anos.

Resumo

Este trabalho apresenta o desenvolvimento de uma ferramenta de geração automática de código para linguagem de alto nível Python. Esta ferramenta tem como objetivo a implementação de controladores de sistemas embutidos e sistemas ciber-físicos, utilizando como base modelos gráficos DS-Pnet (*Dataflow Signals and Petri Nets*), desenhados em ambiente Web IOPT-Flow, sendo uma das suas principais vantagens a diminuição de erros causados pela codificação manual.

A ferramenta de geração de código tem por base as linguagens XML/XSL (*eXtensible Markup Language/ eXtensible Stylesheet Language)* para transformar os modelos gerados no ambiente Web, em código fonte na linguagem de programação Python. Para verificar este método utilizou-se um protótipo de um braço robótico, que realiza a tarefa de *pick and place*, para aplicar a abordagem proposta ao desenvolvimento de uma aplicação industrial. Foi ainda desenvolvido um protótipo de alarme de Porta que permite a operação, ativar a fechadura, programar a combinação secreta e simular intrusões.

Palavras-Chave

Modelos DS-Pnet, *Dataflows*, Código Python; Sistemas ciber-físicos, Sistemas embutidos, IOPT-Flow, *Raspberry Pi*, Gerador de Código automático, Petri-Nets, XSLT/XML, *No-Code/Low-Code*

Abstract

This work introduces the development of an automatic code generation tool for high-level programming language Python. This tool aims the development of embedded systems controllers and cyber-physical systems, using as base graphical models (DS-Pnet), designed in the IOPT-Flow Web environment, contributing to the reduction of errors caused by manual coding. The code generation tool is based on XML/XSL languages to transform models designed in the Web environment into Python programming language. To verify this method, a prototype of a robotic arm was used, which performs the *pick and place* task, to simulate an industrial approach to the developed process. A prototype of a Door Alarm was also used, which allows the operator to activate the lock, program a secret combination and simulate several intrusions.

Keywords

DS-Pnet model, Dataflows, Python Code, Cyber-physical systems, Embedded systems, IOPT-Flow, Raspberry Pi, Automatic Code Generator, Petri-Nets, XSLT/XML, No-Code/Low-Code

Índice

Índice de Imagens

Índice de Tabelas

Índice de Listagens

Lista de Abreviaturas, Siglas e Símbolos

Abreviaturas:

AJAX – *Asynchronous JavaScript and XML*

ASIC – *Application- Specific Integrated Circuit*

CPN-Tools – *Coloured Petri Net Tools*

CPS – *Cyber-Physical system*

DS-Pnet – *Dataflow Signals and Petri Nets*

FPGA – *Field Programable Gate Array*

GALS – *Globally Asynchronous Locally Synchronous*

IEC – *International Electrotechnical Commission*

IOPT – *Input Output Place Transition*

PLC –Controlador Lógico Programável

PN – *Petri Nets*

PNML – *Petry Net Markup Language*

SBC – *Single board Computer*

SIM – *Signal Interpretation Models*

UML – *Unified Modeling Language*

VHDL – *VHSIC Hardware Description Language*

XML – e*Xtensible Markup Language*

XPath – *XML Path Language*

XSL – *eXtensible Stylesheet Language*

XSLT – e*Xtensible Stylesheet language for Transformation*

Capítulo 1. Introdução

Neste capítulo serão abordados os motivos que levaram à realização deste trabalho, os objetivos que se pretende atingir com a nova ferramenta e uma breve introdução à estrutura utilizada no documento.

1.1 Motivação

A linguagem Python tem tido nos últimos anos um crescimento considerável de popularidade, contudo continua a haver poucos geradores capazes de converter código automaticamente para esta linguagem de alto nível. Uma das principais razões para escolher esta linguagem foi a variedade de recursos e bibliotecas, que permitem elaborar variados projetos para diversos fins. Esta linguagem permite desenvolver projetos para dispositivos de hardware, como também permite desenvolver aplicações Web, *Data Science*, inteligência artificial, etc. Devido a estas razões oferece muitos recursos para futuros projetos que poderão ser desenvolvidos utilizando o ambiente Web IOPT-Flow.

Uma das maiores vantagens das ferramentas de geração automática de código é a capacidade de converter modelos gráficos e textuais criados por utilizadores, para uma linguagem de programação especifica, de forma rápida e sem erros causados por codificação manual. Este trabalho exibe uma ferramenta gratuita baseada na Web, que permite realizar este tipo de transformações com base em formalismos DS-Pnet (*Dataflow Signals and Petri Nets)*.

Neste trabalho irá ser abordado a utilização do ambiente Web IOPT-Flow para o desenvolvimento de controladores para plataformas embutidas e ciber-físicas. Estas plataformas utilizam interpretadores de código Python para correr o código gerado. O código convertido pode ser aplicado numa plataforma *Raspberry Pi* (como é o caso dos exemplos mostrados neste trabalho), bem como em outras placas SBC (*Single-Board Computer*), por exemplo: Odroid, Rock Pi, Banana Pi, etc. O objetivo é mostrar a flexibilidade da aplicação deste gerador de código, que pode ser útil tanto para dispositivos industriais quanto para dispositivos comerciais, bastando apenas correr os ficheiros ".py" diretamente na plataforma hardware pretendida, para que esta comece a executar as ações planeadas.

Para a realização deste projeto, foram realizadas as seguintes etapas:

– Estudar o ambiente IOPT-Flow e a correspondente linguagem gráfica;

– Desenvolver modelos de sistemas aplicados a dispositivos de *hardware* para mais tarde testar e validar o gerador de código;

– Estudar a arquitetura interna do ambiente IOPT-Flow e das ferramentas de desenvolvimento empregues para construir o gerador de código a partir de ficheiros XML (e*Xtensible Markup Language*) com a informação dos modelos;

– Desenvolver protótipos aplicando o gerador de código Python;

– Implementar modelos de validação num sistema embutido/ciber-físicos real, usando o código gerado pelo protótipo a correr numa plataforma de *hardware Raspberry Pi*;

1.2 Objetivos

O objetivo principal desta dissertação, consiste no desenvolvimento de uma ferramenta para geração automática de código em linguagem Python, para implementar em controladores de sistemas embutidos e ciber-físicos, utilizando o ambiente Web IOPT-Flow.

A dissertação apresentada foi elaborada na sequência de outros trabalhos anteriores, uma vez que o ambiente Web IOPT-Flow já possuía diversas opções para quem desejasse gerar código automático para outras linguagens de programação. Este ambiente utiliza formalismos baseados em *Dataflows* e redes de Petri, para criar modelos gráficos que posteriormente são validados e simulados.

A ferramenta de conversão apresentada neste trabalho, emprega a linguagem de transformação XSL (*eXtensible Stylesheet Language)*, para transformar arquivos XML contendo modelos DS-Pnet (criados no ambiente IOPT-Flow), em código Python.

Quando o utilizador pretende importar o código Python alusivo ao modelo que criou para o seu computador, o ambiente IOPT-Flow irá gerar um arquivo ZIP, com todos os ficheiros ".py". Estes ficheiros gerados, são posteriormente implementados diretamente nas placas usando interpretadores de Python. O sistema operativo da placa Raspberry Pi inclui um interpretador de código Python (*Raspberry Pi* OS).

1.3 Estrutura do Documento

A estrutura deste documento, é dividida por 6 Capítulos, incluindo referências bibliográficas e anexos. Cada um destes capítulos aborda um tema importante, que descreve o trabalho apresentado:

Capítulo 1 – Introdução do tema em estudo, a estrutura do documento, as razões que motivaram a elaboração deste trabalho e o seu objetivo.

Capítulo 2 – Estado da arte e informação acerca de outros trabalhos relacionados com o tema desta dissertação.

Capítulo 3 – Processo de geração de código automático.

Capítulo 4 – Modelos construídos para validar a ferramenta automática de geração de código Python.

Capítulo 5 –Notas Finais.

Capítulo 6 – Conclusão.

Capítulo 2. Estado da Arte

Este capítulo apresenta Redes de Petri, Dataflow, ferramentas que existem no ambiente IOPT-Flow e outras ferramentas relacionadas. Neste capítulo serão apresentados diversos trabalhos sobre geração automática de código para sistemas embutidos e ciber-físicos. São trabalhos com conteúdo relevante já publicados até ao momento e que foram usados como base para a criação da nova ferramenta.

2.1 Redes de Petri

2.1.1 Introdução

Inventadas por Carl Adam Petri, que documentou esta invenção na sua tese de doutoramento, em 1962, as redes de Petri [2] [3] [4] [5] são uma linguagem de modelação gráfica e matemática.

As redes de Petri são bastante populares entre a comunidade académica e científica para modelar sistemas de eventos discretos, com particular destaque para simulação de sistemas concorrentes. As redes de Petri possuem um conjunto alargado de propriedades matemáticas que permitem a validação automática de modelos, contribuindo para reduzir a probabilidade de erros e reduzir o tempo de desenvolvimento [13].

Existem classes de redes de Petri autónomas e não autónomas. Uma rede diz-se não autónoma quando a evolução do modelo não depende apenas do condicionalismo resultante da marcação da rede, mas interage com o ambiente externo. As classes de redes não autónomas podem ser utilizadas na modelação de sistemas embutidos. A combinação de formalismos matemáticos com a representação gráfica, aliados à possibilidade de simular, verificar e validar os modelos gerados no sistema têm grandes vantagens para o desenvolvimento dos sistemas embutidos e ciber-físicos.

As redes de Petri são uma linguagem intuitiva que utilizam elementos gráficos chamados transições, lugares e arcos para construir modelos, sem necessidade de recorrer a linguagens textuais [1]. Esta linguagem gráfica tem um tempo de aprendizagem reduzido e os modelos são facilmente entendidos por pessoas sem background em tecnologias de informação. As redes de Petri permitem modelar sistemas semelhantes a fluxogramas e máquinas de estados, mas também são muito usadas para modelar sistemas concorrentes com vários componentes correndo em paralelo [2].

2.1.2 Modelação

As redes de Petri [1] [3] [5] [15], são compostas por transições, lugares e Arcos, como pode ser visto na Figura 1. A marcação dos lugares (representados por círculos) define o estado do sistema, enquanto as transições (representadas por retângulos) realizam ações, removendo e adicionando marcas (*tokens*) aos lugares. As transições são conectadas a lugares através de arcos

[4] [11]. Nós do mesmo tipo não podem estar ligados diretamente. Os Arcos de redes de Petri são utilizados para transmitir *tokens* entre as transições e os lugares, definindo uma sequência de evolução do estado do sistema.

Existem dois tipos de arcos:

– Arcos de Entrada: Arcos que iniciam nos lugares e terminam nas transições;

– Arcos de Saída: Arcos que iniciam nas transições e terminam nos lugares;

As redes de Petri não autónomas de baixo nível, utilizadas no ambiente IOPT-Flow, possuem arcos cujo peso define o número de *tokens* que são retirados e adicionados aos lugares. Para habilitar o disparo de uma transição é necessário que o número de *tokens* que existe no lugar de entrada sejam igual ou superior ao valor do peso do arco.

Na figura 1, é possível observar a composição de diferentes tipos de construções básicas realizadas com redes de Petri. É a partir destas construções que se criam outras redes mais complexas para a execução de projetos mais detalhados. Apesar de cada construção básica ter uma composição diferente, a sua representação visual é composta apenas por transições e lugares. As transições são representadas sob a forma de quadrados ou retângulos e os lugares são representados por círculos [3] [11].

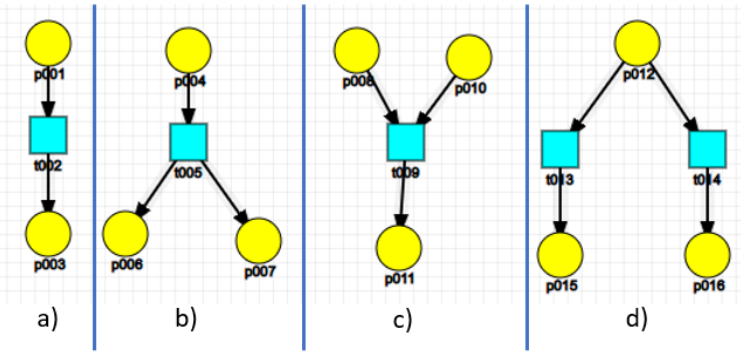


Figura 1: a) Transição simples; b) Distribuição; c) Junção; d) Escolha Não-Determinística;

A Figura 1 apresenta 5 modelos de exemplo, que são descritos por:

a) Transição simples – Se o lugar p001 estiver marcado, a transição t002 dispara.

b) Distribuição – Se o lugar p004 estiver marcado, a transição t005 dispara. Ao disparar, a transição irá recolher 1 *token* de p004 e colocar 1 *token* nos lugares p006 e p007. Esta situação denomina-se de processo "paralelo", que acontece sempre que uma transição atribui *tokens* para dois lugares.

c) Junções – Se o lugar p008 e p010 estiverem marcados, dispara a transição t009. Esta transição vai recolher ambas as marcações e devolver 1 *token* ao lugar p011.

d) Escolha não determinada – Enquanto que a «Distribuição» pode ser considerada (em termos de linguagem de programação) como um "AND", a «escolha não determinada» pode ser considerada como um "OR". Esta montagem só dispara uma transição, tornando a outra inabilitada.

Esta rede possui um fator não determinístico de escolha, levando assim a um conflito de prioridades que pode ser classificado como estrutural ou efetivo. Ambos os conflitos estão associados ao fato de ambas as transições (t1 e t2) possuírem o mesmo lugar p012 inicial. Se a rede não possuir *token*s pode-se considerar como um conflito estrutural, se existir *tokens* no lugar p012, pode-se dizer que o conflito é efetivo. Uma forma de resolver esta situação, seria adicionar dois sinais de *inputs* nas transições t013 e t014 de maneira a controlar a passagem de *tokens* pelo modelo. As ferramentas IOPT-Tools [27] e IOPT-Flow [4] permitem resolver os conflitos através da atribuição de prioridades diferentes a cada transição. Dessa forma os conflitos são resolvidos de forma determinística.

Os lugares podem conter um determinado número marcas/*tokens*. Ao conjunto de marcas existentes em todos os lugares da rede chama-se "marcação" de rede. Os *tokens* permanecem guardados temporariamente nos lugares e são uma peça chave para a realização do disparo das transições. Quando uma transição dispara, os *tokens* associados ao lugar de entrada são destruídos e a quantidade de *tokens* que são criados, passam para o lugar seguinte, que irá depender do peso do arco de saída [15]. Para disparar uma transição, os lugares de entrada no instante imediatamente anterior têm de conter *tokens* suficientes. Desta forma a marcação dos lugares define as transições que estão habilitadas a disparar em cada momento e estabelece a sequência de funcionamento do modelo. Pode-se afirmar que o peso do arco controla a quantidade de fluxo de dados que passam por cada etapa (Figura 2) [1] [3] [11] [12].

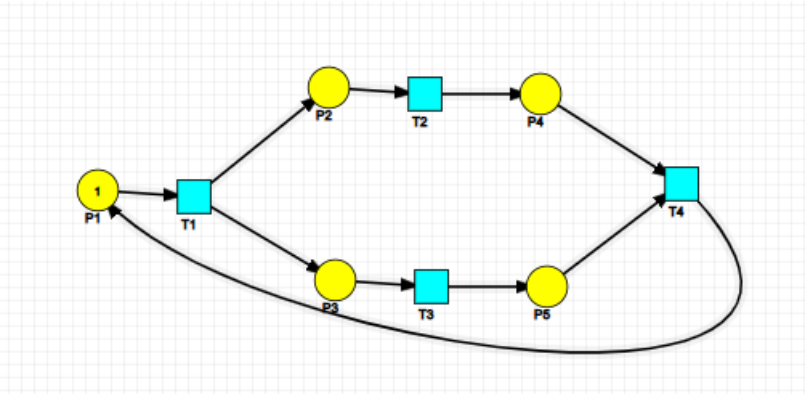


Figura 2: *Exemplo de rede Petri (Amarelo – Lugares; Azul – Transições)*

Os dois conceitos principais na modelação das redes de Petri são o conceito de ação (associado às transições) e o conceito de pré-condição (associado aos lugares) [3] [5]. Para ocorrer uma ação é necessário que certas condições sejam verdadeiras. As pré-condições determinam o disparo das transições. Quando o disparo ocorre, as pré-condições deixam de ser verdadeiras, permitindo que outras condições se tornem válidas no próximo passo de execução.

2.1.3 Definições

Os autores [2] [3] [13] apresentam duas definições de Redes de Petri fundamentadas em teorias diferentes. De acordo com estes autores existem dois tipos de fundamentações que definem as redes de Petri, a primeira corresponde à teoria dos multiconjuntos (Bags) [3] [13]. A segunda utiliza conceitos algébricos matriciais para validar as condições das ações [3] [13].

1ª: teoria dos multiconjuntos (Bags):

Segundo a teoria dos multiconjuntos, as redes de Petri podem ser definidas por 5 elementos:

PN = (P, T, I, O, K)

Em que:

– PN - *Petri Nets*

– P= $\{p1, p2, \ldots, pm\}$: Conjunto finito de lugares;

– T= $\{t1, t2, \ldots, tn\}$: Conjunto finito de Transições, em que P *U* T! = 0 e P *Ո T = 0*

– I= (P x T): *Input* (arcos de entrada), que atribui os arcos desde os lugares até transições;

– O= (T x P): *Output* (arcos de saída), que atribui os arcos desde as transições até lugares;

– K ={m1,m2, …}: Conjunto de tokens que define a marcação inicial da rede

Cada elemento desempenha um papel fundamental para o desenvolvimento de modelos novos. O elemento "I", define os lugares de entrada de cada transição. O elemento "O" define os lugares de saída de cada transição [2] [3] [12].

Para exemplificar, criou-se um modelo simples que simula as 4 estações do ano. Por cada estação, está associado o clima habitual da época. O ano começa com o Inverno, nesta época o tempo é muito frio. De seguida entra a Primavera, onde o clima já é mais ameno. Depois da Primavera entra o Verão e o clima mais quente. Por fim temos o Outono, que contém um clima muito chuvoso (Figura 3), inspirado no artigo [1].

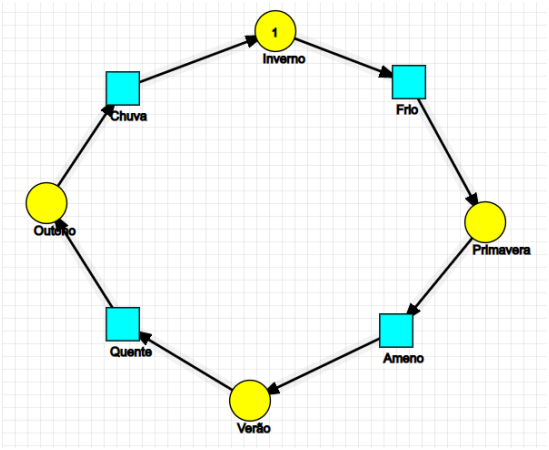


Figura 3: Estações do ano e o seu clima Habitual representados Graficamente em redes de Petri

O conjunto de Lugares (P) são descritos por:

P= $\{Inverno, Primavera, Verão, Outono\}$;

O conjunto de Transições (T) são descritos por:

T= $\{Frio, Ameno, Calor, Chuva\}$;

O conjunto de lugares de entrada (I) são descritos por:

I (Frio) = {Inverno}
I (Ameno) = {Primavera}
I (Calor) = {Verão}
I (Chuva) = {Outono}

I = {I (Frio) = [Inverno]; I (Ameno) = [Primavera]
I (Calor) = [Verão]; I (Chuva) = [Outono] }

O conjunto de lugares de saída (O) são descritos por:

I (Frio) = {Primavera}
I (Ameno) = {Verão}
I (Calor) = {Outono}
I (Chuva) = {Inverno}

O = {O (Frio) = [Primavera]; O (Ameno) = [Verão]
O (Calor) = [Outono]; O (Chuva) = [Inverno]}

O conjunto dos *tokens* (K) são descritos por:

K = {Inverno= 1; Primavera= 1; Verão= 1; Outono= 1}

2ª: álgebra matricial

A definição da rede de Petri, segundo o ponto de vista da álgebra matricial, como apresentado em [3], é composta por 5 elementos:

PN = (P, T, I, O, K) em que :

– P= $\{p1, p2, \dots, pm\}$: conjunto finito de lugares;

– T= $\{t1, t2, \dots, tn\}$: conjunto finito de Transições, P *U* T! = 0 e P *Ո T = 0;*

– I= (P x T): Matriz de pré-condição, ocorre quando a condição anterior se verifica (Lugar → Transição);

– O= (T x P): Matriz de pós-condição, ocorre quando a ação anterior se verifica (Transição → Lugar);

– K ={m1,m2, …}: Corresponde à marcação da rede, ou seja, o conjunto de *tokens* associados a cada lugar.

A matriz de Pré-condições é descrita por:

$$I = \begin{array}{c} \begin{array}{cccc} \text{Frio} & \text{Ameno} & \text{Calor} & \text{Chuva} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{array}{l} \text{Inverno} \\ \text{Primavera} \\ \text{Verão} \\ \text{Outono} \end{array}$$

A matriz de Pós-condições é descrita por:

$$O = \begin{array}{c} \begin{array}{cccc} \text{Frio} & \text{Ameno} & \text{Calor} & \text{Chuva} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{l} \text{Inverno} \\ \text{Primavera} \\ \text{Verão} \\ \text{Outono} \end{array}$$

Cada matriz representa as pré-condições e as pós-condições de cada etapa da Figura 3. Juntando ambas as matrizes, é possível desenhar uma Tabela que mostra a conexão entre as ações e as condições (Tabela 1).

Pré-condições	Ação	Pós-condições
Inverno	Frio	Primavera
Primavera	Ameno	Verão
Verão	Calor	Outono
Outono	Chuva	Inverno

Tabela 1: Estações do ano e o seu clima Habitual

Ao comparar a Tabela 1 com o modelo gerado na Figura 3 é possível notar o processo sequencial do sistema. Pegando no exemplo da primeira ação, o Frio só acontece se na condição anterior estiver o Inverno e na condição seguinte estiver a Primavera. Isto quer dizer que, para uma ação poder ser realizada, a condição anterior tem de corresponder à sequência que se encontra na Tabela 1, caso contrário a ação não se verifica.

2.1.4 Classes

Os artigos e dissertação [3] [4] [11] [13] [14] descrevem diversas classes de redes de Petri. Estas classes podem ser divididas em redes de baixo (ordinárias) e alto nível (não ordinárias). As redes de Petri de baixo nível utilizam marcação unitária (inteiro positivo). As redes de Petri de alto nível utilizam marcação particular (não unitária).

As classes de redes de Petri podem ser classificadas em duas características: Redes de Petri não-Autónomas e redes de Petri Autónomas. Redes de Petri autónomas dependem apenas de

condicionalismos resultantes das sua estrutura e marcação. Redes de Petri não-autónomas possuem dependência de eventos externos.

Redes de Petri ordinárias – Autónomas:

– **Redes de Petri Binárias ou Condição-Evento:** Classe que permite o máximo de um *token* em cada lugar e todos os pesos dos arcos possuem valores unitários [3] [11];

– **Redes de Petri Place-Transition:** Classe que permite múltiplos *tokens* no mesmo lugar e arcos com peso não unitário, que possibilitam a passagem de mais de um *token* (Utilizada no ambiente IOPT-Flow) [3];

– **Redes de Petri Não-Ponderadas:** Classe que permite acumular mais do que um *token* no mesmo lugar, mas os valores dos arcos são unitários [13];

– **Redes de Petri Ponderadas:** Classe que permite ter arcos múltiplos, associados a um único arco que possibilita a passagem de maís do que um *token* [13];

Redes de Petri ordinárias – Não-Autónomas:

– As Redes de Petri Temporizadas são sistemas cujo funcionamento é totalmente dependente do tempo;

- **Redes de Petri Temporizadas Determinísticas**: Adiciona aspetos temporais determinísticos a componentes de rede dos modelos (tempo associado a: lugares, transições e marcas) [3] [14];
- **Redes de Petri Temporizada Estocásticas**: Emprega uma análise probabilística para determinar o disparo de transições [3] [11] [14];

– **Redes de Petri Sincronizadas**: Cada transição está associada a um evento. Para disparar a transição, esta tem de estar habilitada e o evento associado ocorrer [15];

As redes de Petri não-ordinárias – Autónomas:

– **Redes de Petri coloridas**: Classes que têm o objetivo de reduzir o tamanho dos modelos, permitindo que os *tokens* sejam individualizados pelas cores que lhes são atribuídas [3] [4] [11] [15];

– **Redes de Petri Hierárquicas**: Classe que contém mecanismos para agrupar partes do modelo, através de dois componentes denominados por superpágina e subpágina. A superpágina gera um modelo mais compacto composto por transições, lugares e arcos. A subpágina detalha a informação da super-página [3] [14] [15];

2.1.5 DS-Pnet

As redes DS-Pnet (*Dataflow Systems Petri Nets*) descritas em [4] [25], foram concebidas com o intuito de suportar o desenvolvimento de sistemas ciber-físicos. Esta modelação combina redes de Petri [5] e *Dataflows* para a criação dos modelos. As redes de Petri da linguagem DS-Pnet são redes ordinárias que herdam as características das redes IOPT (*Input*/*Output Place*/*Transition*). Estas redes permitem definir máquinas de estado que evoluem de acordo com

eventos externos. O elementos de Dataflow são utilizados para realizar transformações matemáticas e lógicas dos sinais de entrada, de forma a poder determinar os resultados nos nós de saída.

A Tabela 2, apresenta os diversos tipos de elementos de modelação oferecidos pelo ambiente Web IOPT-Flow. Este ambiente possui todas as ferramentas necessárias para elaborar modelos de raiz utilizando a linguagem de modelação DS-Pnet, simular modelos em tempo real através do navegador Web e também tem a possibilidade de realizar geração automática de código (C, VHDL (*VHSIC Hardware Description Language)*, JavaScript e agora Python) para correr em plataformas computacionais que têm o objetivo de controlar entidades físicas.

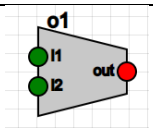
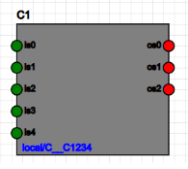
Identificação	Elementos das redes de Petri
Lugar (*Place*)	
Transição (*Transition*)	
Arco de rede de Petri (*Normal Arc*)	
Arco de *Dataflow* (*Read Arc*)	
Sinal de Entrada (*Input Signal*)	
Sinal de Saída (*Output Signal*)	
Evento de Entrada (*Input Event*)	
Evento de Saída (*Output Event*)	
Sinal Interno (*Internal Signal*)	
Operação de *Dataflow* (*New Operation*)	
Componente (*Component*)	

Tabela 2: Elementos do modelo DS-Pnet

Com base em [4] e [21], que incluem uma explicação detalhada de cada elemento das redes de Petri, pode-se notar que:

− Os elementos compostos por lugares, transições e arcos têm o comportamento igual aos nós utilizados nas redes de Petri de baixo nível (ordinárias). Estes elementos geram modelos que em conjunto com os arcos, compõem as máquinas de estado. É de notar que não se pode ligar elementos do mesmo tipo, pois isso provoca erros de sintaxe.

− Os sinais e os eventos de entrada e saída definem a interface externa de um modelo. Estes sinais e eventos, são utilizados para ler sensores, controlar atuadores elétricos (qualquer dispositivo capaz de converter energia elétrica em energia mecânica) e controlar outros sistemas. Os sinais são compostos por valores booleanos ou valores com intervalos inteiros.

− Um sinal interno guarda informação para depois ser utilizada noutras operações. Por exemplo, no caso de um contador, o sinal interno guarda a informação que recebe da operação "Timer" (Figura 40). A operação compara o valor anterior do sinal interno com o valor mínimo estipulado. Quando o valor mínimo é atingido, a contagem termina e a operação seguinte inicia.

− Os arcos de leitura com início em elementos de *Dataflows* são utilizados para transmitir valores de sinais e eventos de entrada, transmitir resultados de outras operações e transmitir o estado dos nós dos modelos de rede Petri. Os arcos de leitura com início em elementos das redes de Petri permitem ler as marcações de lugares e eventos acionados e as transições quando estas ficam ativas. Os valores da marcação de lugares podem ser usados para realizar cálculos de fluxos de dados ou serem atribuídos aos sinais e eventos de saída. Uma forma de melhorar a estética dos arcos de leitura, consiste em desenhar os arcos na forma simbólica: Ao usar a opção "*collapse*", os arcos são escondidos, aparecendo apenas uma referência textual junto do seu destino. Em alternativa, os arco de rede de Petri são utilizados para transmitir informações (marcações/ *tokens*) entre as transições e os lugares, definindo a ordem de funcionamento do modelo.

A parte de fluxo de dados de um modelo é composta por operações e arcos de leitura. As saídas das operações, são definidas por expressões matemáticas, que realizam transformações de dados consoante valores que recebem na entrada. Tal como os arcos, as operações podem ser colapsadas, mostrando apenas as expressões matemáticas.

De acordo com os artigos [24] [25], os *Dataflows* são considerados fluxos de dados síncronos, pois em teoria são capazes de executar cálculos de forma instantânea. Por essa razão, todas as operações de fluxo de dados podem ser realizadas na mesma etapa de execução. As expressões que definem as operações de fluxo de dados possuem as seguintes características:

– Interação com elementos do formalismo DS-Pnet, nomeadamente arcos de leitura *(Figura 5)*;

– Valores literais/constantes *(Figura 6)*;

– Operadores Aritméticos *(Tabela 3)*;

– Operadores de Comparação *(Tabela 4)*;

– Operadores Lógicos *(Tabela 5)*;

– Operadores de Lógica Binária *(Tabela 6)*;

– Operadores de construção *(Tabela 7)*;

– Implementação de valores Tabelados (*table [idx1,idx2]*)

Com base em [4], os *Dataflows* no ambiente IOPT-Flow, são representados por um ícone com a designação de "operações" (*operations*). Estas "operações" permitem receber sinais contendo valores booleanos ou inteiros como parâmetros de entrada e por fim devolver o valor

calculado/condicionado, na saída. O utilizador pode estabelecer o número de entradas que necessita para gerar a operação, mas por omissão, o programa sugere sempre 2 *inputs* (*Figura 4*).

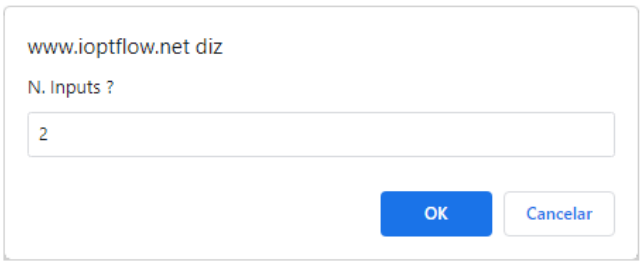


Figura 4: *Número de Inputs*

Por omissão, as operações apresentam apenas uma saída, contudo existe a possibilidade de editar o modelo e inserir mais do que uma saída. Para isso a saída terá de estar associada a um tipo de dados e a uma expressão matemática, porque caso contrário, esse *output* inserido terá sempre o valor pré-definido de 1. Como vantagem, o modelo fica muito mais organizado e caso os *inputs* sejam os mesmos, evita a repetição de código.

No exemplo da *Figura 5* a saída da operação _1 fica ativo se o sensor _11 estiver acionado e _2 fica ativo se o sensor _22 estiver acionado. Isto é uma forma de criar uma condição, sem usar uma expressão condicional "WHEN/OTHERWISE". Esta operação transmite o valor de cada uma das entradas para cada uma das saídas.

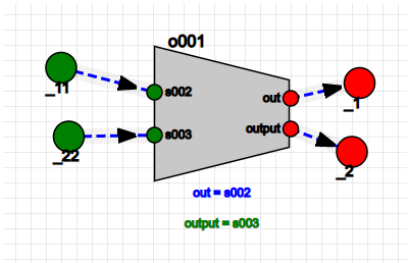


Figura 5: *Editar Janela de Operações*

A janela de edição das operações pode ser composta por dois tipos de expressões:

– **Expressão Matemática**: Usada quando o utilizador apenas pretende definir o valor de saída da operação. O exemplo mostrado na Figura 6 é composto por uma equação que multiplica as duas primeiras constantes (5 * 3 = 15) e de seguida, o seu resultado é subtraído pelo somatório das últimas três constantes (15 – 13 = 2);

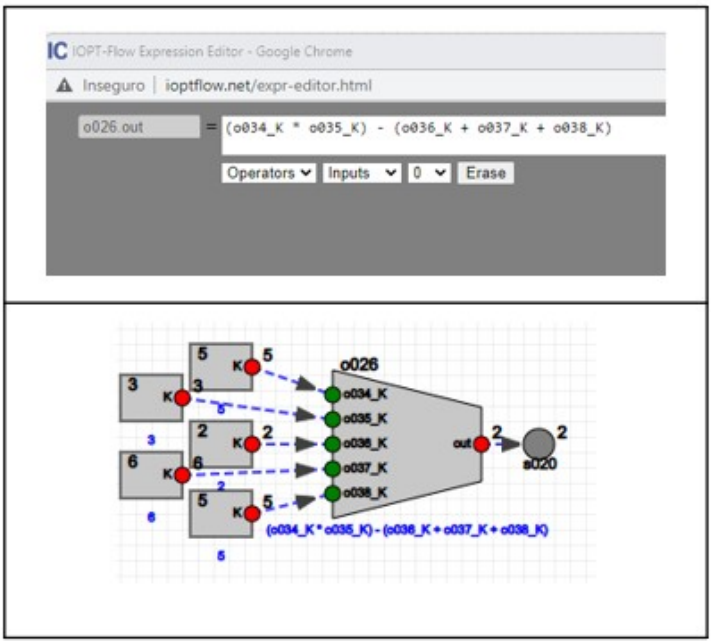


Figura 6: Expressão Matemática

– **Expressão Condicional**: Usada pelo utilizador sempre que este pretenda obter um resultado diferente de acordo com o conjunto de condições. O modelo mostrado na Figura 7 contém quatro condições compostas por "WHEN/OTHERWISE", uma condição só é verdadeira, se todos os parâmetros depois do "WHEN" se confirmarem. Se nenhuma condição for verdadeira, a saída da operação será o valor definido na clausula "OTHERWISE";

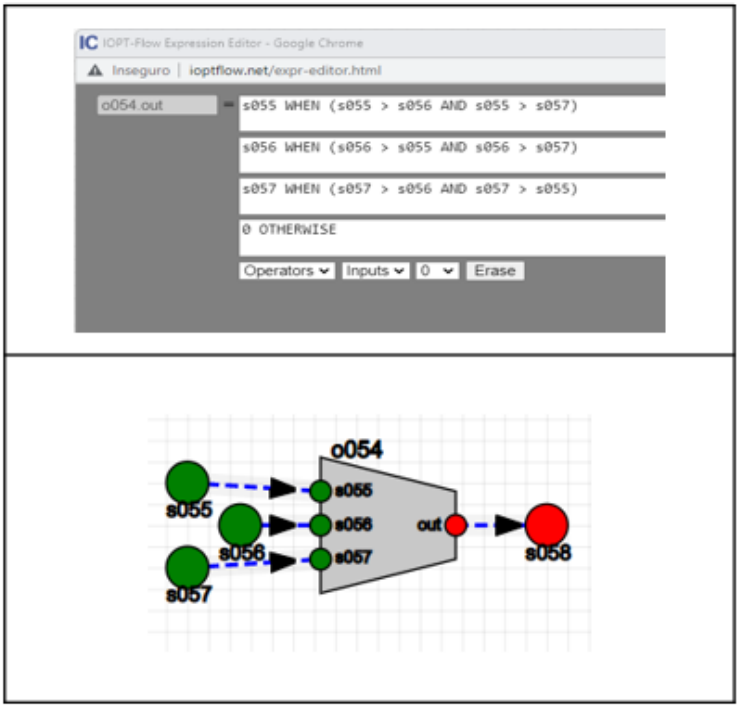


Figura 7: Expressão Condicional

Por definição, quando o utilizador pretende impor condições "WHEN" a todos os argumentos, obrigatoriamente tem de colocar "OTHERWISE" na condição final, caso contrário obtém-se um erro de sintaxe.

De acordo com a dissertação [4], as expressões matemáticas são compostas por operandos e operadores. Os operandos podem corresponder a valores decimais, valores hexadecimais ou nomes das entradas da operação. Os operadores disponíveis são:

– Operações aritméticas (Tabela 3) : Formado por operações básicas da matemática ;

Operações Aritméticas	
Adição	+
Subtração	-
Multiplicação	*
Divisão	/

Tabela 3: Operações Aritméticas

– Operações de Comparação (Tabela 4): Formado por operadores que têm o objetivo de analisar as condições de comparação de uma expressão e retornar um valor booleano;

Operações de Comparação	
Inferior	<
Superior	>
Inferior ou Igual	<=
Maior ou Igual	>=
Igual	=
Diferente	<>

Tabela 4: *Operações Comparação*

– Operações de Lógica (Tabela 5): Formado por operadores que têm o objetivo de analisar as condições lógicas de uma expressão e retornar um valor booleano;

Operações Lógicas	
AND Lógico	AND
OR Lógico	OR
XOR Lógico	XOR
NOT Lógivo	NOT

Tabela 5: *Operações Lógicas*

– Operação de Lógica Binária (Tabela 6): Formado por operadores lógicos binários, que têm o objetivo de analisar operações bit a bit, realizando comparações entre expressões hexadecimais;

Lógica Binária	
Lógica Binária AND	ANDB
Lógica Binária OR	ORB
Lógica Binária XOR	XORB

Tabela 6: *Operações Binárias*

– Outras Operações (Tabela 7): Formado por outros operadores;

Operadores de construção	
Operador Condicional	WHEN
Condição Pré-definida	OTHERWISE
Sub-expressão	( )

Tabela 7: *Outras Operações*

Todas as entradas e saídas de uma operação possuem um nome e um tipo de dados associado, que é definido automaticamente quando a operação é criada. No entanto, existe sempre possibilidade de mudar as predefinições, escolhendo o nome e o tipo de passagem de dados preferencial.

Os Dataflows possuem quatro tipos de dados [4]:

– Booleanos: Expressão que apresenta apenas dois estados, verdadeiro (1) e Falso (0);

– *Integer Range*: Expressão que permite ao operador apresentar uma gama de valores inteiros (positivos e negativos), que varia entre uma gama máxima e o mínima;

– *Fixed Point Range*: Expressão que permite apresentar valores fracionários;

– *Event*: Expressão que apresenta um acontecimento instantâneo que fica ativo em apenas uma etapa de execução;

Sempre que se ligam arcos com sinais de entrada a uma operação, a respetiva âncora de entrada assume o nome das variáveis onde o arco originou.

2.1.6 Sistemas ciber-físicos e embutidos

Os sistemas ciber-físicos (CPS- *Cyber-Physical system*) são sistemas que utilizam elementos computacionais (*software*) para desenvolver e controlar sistemas físicos (componentes mecânicos ou eletrónicos), isto é, uma plataforma que deteta, processa e comunica informações em tempo real.

O principal objetivo dos sistemas ciber-físicos é controlar o comportamento dos processos físicos através de um sistema cibernético que muda a sua conduta através de ações. A conexão entre os meios cibernéticos com os processos físicos, pode ser feita através de sensores, motores e atuadores, de forma que a monitorização e o controlo do ambiente físico seja efetuado através de sistemas virtuais [21].

Os sistemas ciber-físicos utilizam a parte cibernética para gerar elementos computacionais (programas de comando e controlo) e a parte física para controlar entidades físicas (modelos físicos de teste). O processo físico é responsável pela realização das operações e o processo cibernético é responsável pelas aplicações múltiplas de programação.

Os CPS têm vindo a ser estudados e desenvolvidos ao longo dos anos, tornando-se bastante importantes em muitas aplicações no ramo da manufatura [16], ramo da automação [17], ramo da segurança [18], ramo da saúde [19] e ramo da instrumentação [20], etc. Estes sistemas são considerados habitualmente como a evolução dos sistemas embutidos.

Os sistemas embutidos [30] são equipamentos utilizados em vários aparelhos que existem no dia-a-dia, tanto podem ser empregues em simples brinquedos, como em máquinas industriais. Um equipamento é classificado como embutido quando tem uma funções exclusivas para uma tarefa única. Este sistema depende de elementos computacionais limitados, para controlar entidades físicas. O sistema cibernético é composto pelo gerador de código Python, pela semântica do modelo do IOPT-Flow e pelo código de *inputs* e *outputs* desenvolvido em Python

2.1.7 Normas

Existem duas normas que definem linguagens de programação para PLCs (Controladores Lógicos Programáveis). Estas normas são habitualmente usadas para desenvolver controladores para sistemas embutidos e oferecem funcionalidades semelhantes às ferramentas IOPT-Flow e ao gerador de código proposto nesta dissertação:

IEC61131 é uma norma técnica internacional para sistemas de Controladores Lógicos Programáveis (PLC). Esta possui requisitos referentes ao *hardware* e ao sistema de programação do sistema. A norma IEC61131-3 representa a parte programável do ambiente de *software* do aparelho de PLC. Esta norma padrão suporta duas linguagens de programação textuais (*Structure Text* (ST) e *Instruction List* (IL)), duas linguagens gráficas (*Ladder Diagram* (LD) e *Function Block Diagram* (FBD)) e ainda uma linguagem identificada como gráfica, mas pode também permitir programações textuais (*Sequencial Function Chart* (SFC)) [23].

IEC61499 é uma norma baseada nas funções IEC61131, que tem como objetivo definir um modelo genérico para sistemas de controlo baseado em blocos de funções, para aplicações industriais. Face às semelhanças de arquitetura com as redes de Petri, esta norma apresenta funcionalidades semelhantes às exibidas pelo ambiente IOPT-Flow. Neste caso pode-se considerar que estas nomas oferecem funcionalidade semelhante à linguagem de modelação apresentada nas DS-Pnet, pois a sua geração permite aplicar os scripts gerados em elementos de hardware industriais.

2.2 IOPT-FLOW

2.2.1 Ambiente IOPT-Flow

O ambiente IOPT-Flow é um ambiente Web (disponível em: http://www.IOPT-Flow.net/ e http://gres.uninocva.pt) [4], que emprega uma linguagem de redes de Petri para controlar/definir o estado do sistema e uma linguagem de *Dataflows* para manipulação de dados. A interação entre ambas as linguagens permite processar os dados em sinais de entrada e gerir o estado do sistema em resposta a eventos externos [25]. Enquanto os fluxos de dados processam

sinais de entrada e calculam os valores de saída, as redes de Petri são utilizadas para controlar a evolução do estado do sistema. Uma das vantagens deste ambiente é que, para além de servir para a construção de modelos com base nas linguagens mencionadas, este também tem a possibilidade de construir modelos que emulam outras linguagens gráficas usadas em controladores lógicos programáveis. As linguagens gráficas dos controladores lógicos programáveis são *Ladder* e *Grafcet* [4]. Existem vários artigos que exploram as utilidades deste ambiente, as referências [22] e [25] são o exemplo disso.

O ambiente Web IOPT-Flow suporta todos os processos de desenvolvimento dos modelos, desde a conceção e edição, simulação e validação, geração automática de código (C, JavaScript, VHDL, Python), extração de resultados para ficheiros CSV, registo de histórico das simulações e geração de espaço de estados. O novo gerador de código Python foi adicionado aos geradores já existentes que se encontravam já disponíveis nesta plataforma.

À semelhança do *software* LabView [38] que apresenta janelas de edição de diagramas de blocos e um painel frontal, este ambiente também tem uma interface composta por duas janelas (Figura 8) a primeira é uma janela de trabalho/edição (Figura 8: a)) e a segunda é uma janela de simulação (Figura 8: b)). A janela de edição permite desenhar o modelo, a janela de simulação apresenta uma interface gráfica que corre os modelos criados. A imagem da Figura 8 exibe as ferramentas disponíveis na janela de trabalho e na janela de simulação:

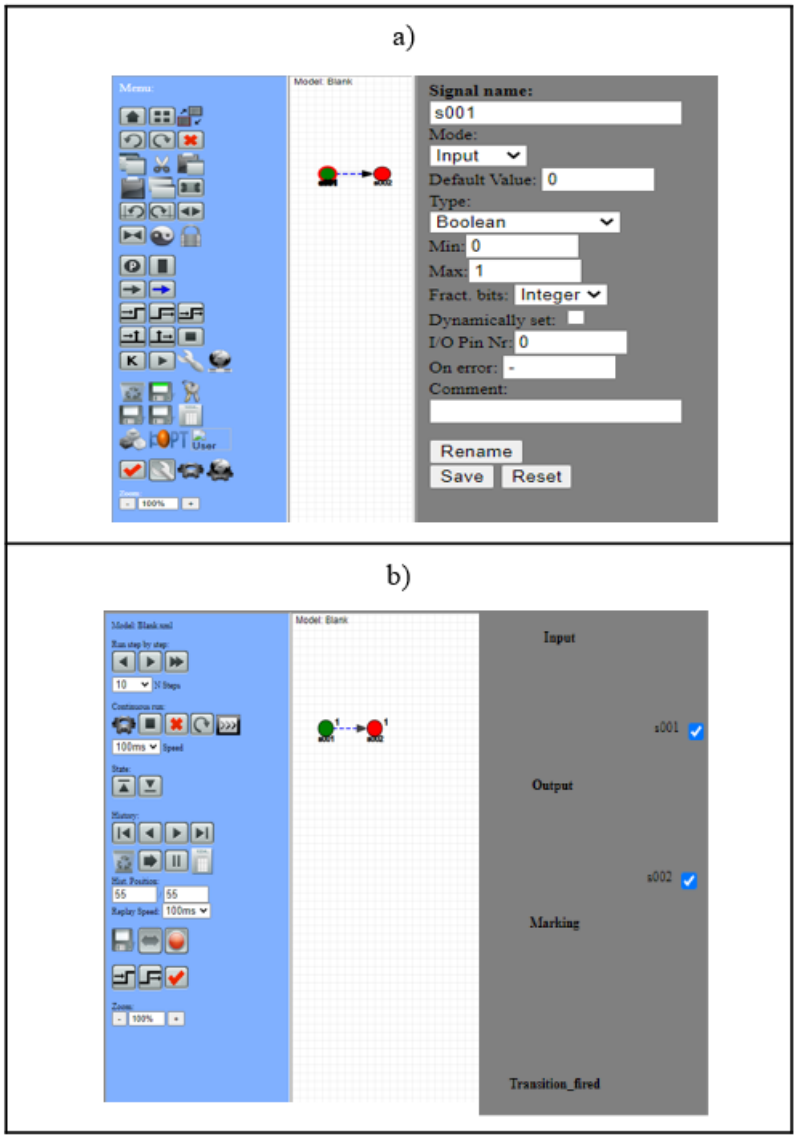


Figura 8: a) Janela de Trabalho; b) Janela de Simulação

A estrutura do ambiente Web de ambas as janelas, é composta por três secções:

– A Secção do lado esquerdo, contém a caixa de ferramentas disponibilizadas ao utilizador. É com base nestas ferramentas que se constrói e controla as características dos modelos desenvolvidos;

– A Secção central contém a área de trabalho. Onde se efetua a construção do modelo ou, no caso da simulação, é onde se visualiza e controla o estado dos modelos;

– Secção do lado direito mostra as propriedades dos elementos selecionados na janela de Trabalho. Para a janela de simulação, esta secção mostra a atividade do modelo em tempo real.

A caixa de ferramentas da Janela de Trabalho (Figura 8: a), oferece diversas opções de edição de modelos, para além das ferramentas de geração de código compostas pelas redes de Petri e *Dataflows*, também é possível utilizar as opções de copiar/colar/cortar, desfazer e refazer, carregar e guardar, colapsar as ligações, criar um semáforo para as zonas selecionadas, inversão de seleção, criação e reutilização de componentes, simular o modelo desenvolvido, controlar o sistema

remotamente e gerar código automaticamente (C, JavaScript, VHDL e a partir de agora Python) [4].

As ferramentas geração automática de código (*Code Generators*) disponíveis no ambiente Web IOPT-Flow, utilizam transformações XSLT (e*Xtensible Stylesheet language for Transformation*) para gerar ficheiros de código fonte que implementam a semântica de execução do modelo. Esta semântica XML, traduz o comportamento para as linguagens de programação disponíveis (C, JavaScript) ou para a geração das descrições de *hardware* (VHDL), capazes de serem implementadas em dispositivos FPGA (*Field Programable Gate Array*) ou ASIC (*Application- Specific Integrated Circuit*).

A caixa de ferramentas da janela de trabalho, permite armazenar e exportar para o computador, os modelos desenvolvidos pelo utilizador:

-Por omissão, se o utilizador carregar no ícone «*Save Model*» antes de definir o nome do ficheiro no servidor do IOPT-Flow («*Save as*»), o ambiente Web gera um arquivo XML com todos os dados que compõem o modelo;

-Caso o utilizador deseje exportar a semântica com as regras principais do modelo para o seu computador, a ferramenta de geração exporta arquivos XML através do ícone "*Code Generatores*".

-Para importar documentos XML para o ambiente IOPT-Flow, basta aceder ao ícone "*Open Model*" e indicar a localização do ficheiro no próprio computador. Quando este ficheiro é importado, ele não fica guardado no servidor IOPT-Flow, a menos se o utilizador execute a opção "*Save as* ".

-Se o utilizador preferir manter os ficheiros dos modelos, no servidor IOPT-Flow este pode optar por guardar o modelo criado numa pasta pública no servidor, de forma a poder partilhar o seu conteúdo com outros utilizadores, ou pode optar por criar uma conta ("*Login on Private Folder* ") e guardar o modelo gerado numa pasta privada.

A ferramenta de simulação (*Run Simulator*) disponível na janela de trabalho (Figura 8: b)), tem o objetivo de simular a execução do modelo construído em ambiente Web, sem a necessidade de recorrer a um sistema físico de testes. O gerador de código automático de JavaScript utiliza o ambiente Web como ferramenta de compilação de modelos DS-Pnet. O simulador permite realizar a execução continua do modelo ou pode optar pela execução passo-a-passo.

A caixa de ferramentas da Janela de Simulação (Figura 8: b)), é composta por 5 elementos. Cada um desses elementos tem influência para a execução dos modelos:

– Execução passo a passo ("*Run step by step*"): Corre o modelo de forma gradual;

– Execução continua ("*Continuos Run*"): Corre o modelo de forma ininterrupta;

– Estado ("*State*"): Força as marcações no modelo. Utilizando o exemplo da Figura 2, é possível forçar o deslocamento da marca para o seu estado inicial (P1) ou colocar marcações em qualquer lugar (P1, P2, P3, P4, P5);

– Histórico (“*History*”): Registo de informação acerca do desempenho do modelo. Este registo pode ser exportado para um ficheiro em CSV ou o desempenho pode ser visto através de um gráfico de formas de ondas;

– Zoom: Amplia ou minimiza a área de trabalho;

2.2.2 Arquivos de dados e o seu formato

Os modelos criados em ambiente Web de IOPT-Flow, de acordo com a dissertação [4], são armazenados em formato XML (*eXtensible Markup Language*). O anexo VII apresenta um exemplo de XML com todos os dados que existem do modelo da Figura 3. Esta linguagem de marcação tem o objetivo de criar documentos com base na estruturação de dados, relacionados numa hierarquia, onde cada elemento da estrutura ocupa um nível específico. A extensibilidade deve-se ao facto de a linguagem permitir definir os elementos de marcação de um documento, organizando o conteúdo de forma a ser possível a integração com outras linguagens. A composição de um modelo XML é constituída pela declaração, por um elemento/nó raiz, pelos atributos dos elementos/nós, pelos próprios elementos/nós em si e por fim pela possibilidade da existência de elementos/nós vazios (Figura 9).

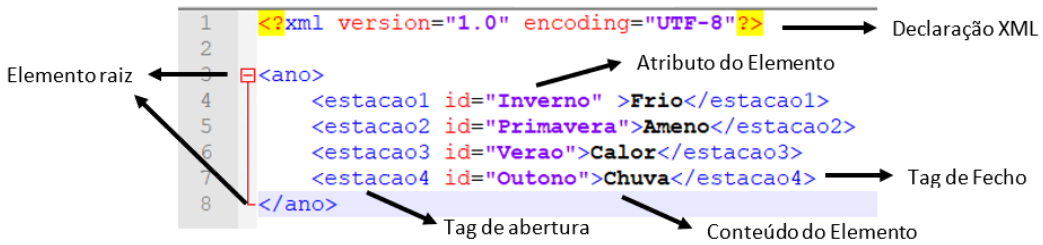


Figura 9: Estrutura XML (linguagem de marcação)

Os elementos/nós da linguagem XML (Figura 9) são compostos por tudo o que se encontra entre a tag de abertura do elemento até a tag de encerramentos do mesmo elemento.

A linguagem XSL permite formatar documentos XML para um determinado tipo. XSL *(eXtensible Stylesheet Language*) é uma linguagem que se expressa em folhas de estilo, cujo objetivo é converter a semântica de execução XML para outro tipo de linguagem, que no caso deste trabalho é a linguagem Python.

A linguagem XSL é composta por três características:

– **Transformações XSL (XSLT)**: Transforma um documento XML num documento de outro tipo. Durante o processo de transformação, o XSLT poderá copiar, alterar, editar o novo documento, removendo elementos ou atributos, restruturar elementos e realizar testes;

– **Linguagem de caminho XML (XPath- *XML Path Language*)**: Linguagem utilizada para aceder ou fazer referência a elementos/nós de um documento XML. Esta linguagem é baseada na representação hierárquica do documento XML, possibilitando a sua consulta e realiza busca/pesquisas dentro do documento;

– **Objetos de Formatação (XSL-FO)**: Linguagem composta por vocabulário que especifica a semântica de formatação. Estes documentos são documentos XML onde o elemento raiz é “fo:root” em vez de “XSL:root”;

As Figuras 10 e 11, exibe um exemplo simples de transformação (XSLT) de um documento de linguagem XML e XSL. Neste exemplo pretende-se mostrar a associação entre a linguagem de marcação com a linguagem de formatação, para construir um documento de texto.

```
<?xml version="1.0"?>
<estacoes>
        <estacao nome="Inverno" clima="Frio" periodo="Dezembro - Março"/>
        <estacao nome="Primavera" clima="Ameno" periodo="Março - Junho"/>
        <estacao nome="Verao" clima="Calor" periodo="Junho - Setembro"/>
        <estacao nome="Outono" clima="Chuva" periodo="Setembro - Dezembro"/>
</estacoes>
```

Figura 10: Exemplo de XML

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="estacoes">
        <xsl:text> Estações do Ano: &#10;</xsl:text>
        <xsl:for-each select="estacao">
                <xsl:value-of select="@nome"/>
                <xsl:text>- Clima: </xsl:text>
                <xsl:value-of select="@clima" />
                <xsl:text>- Periodo: </xsl:text>
                <xsl:value-of select="@periodo" />
                <xsl:text>&#10;</xsl:text>
        </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Figura 11: Exemplo de XSL

A Figura 10 mostra a composição hierárquica de um documento XML, enquanto a Figura 11 apresenta o estilo de formatação que processa o ficheiro a transformar. O resultado da transformação é apresentado na Figura 12, onde exibe as quatro estações do ano, o seu clima e o período das mesmas. Utilizou-se um ciclo for (*for-each*) na Figura 11, para adicionar linhas de informação no documento de texto, em função das estações do ano que constam no ficheiro XML. A seleção de cada atributo, é feita através da definição do valor (*value-of*). A passagem de linha é realizada através de uma referência de caracter numérico em código Decimal pela Tabela Unicode (
), cuja identificação é *line feed* (*LF*).

```
 Estações do Ano:
Inverno- Clima: Frio- Periodo: Dezembro - Março
Primavera- Clima: Ameno- Periodo: Março - Junho
Verao- Clima: Calor- Periodo: Junho - Setembro
Outono- Clima: Chuva- Periodo: Setembro - Dezembro
```

Figura 12: Exemplo de transformação

2.2.3 Gerador de espaço de estados

Uma das vantagens dos modelos baseados em redes de Petri consiste na capacidade de gerar espaço de estados. Esta aplicação dos ambientes IOPT-Flow e IOPT-Tools [27], permite

detetar todas as possibilidades de funcionamento do modelo, desde o estado inicial até ao estado final, mostrando todas as combinações possíveis de transições que podem disparar.

O gerador de espaço de estados [4] é capaz de detetar automaticamente, os estados que "prendem" o sistema. Devido a isto, consegue identificar e mostrar ao utilizador a localização dos problemas que o sistema poderá ter. Esta ferramenta é bastante útil para qualquer modelo, principalmente nos modelos onde a modelação é mais complexa e mais difícil de detetar os *deadlocks* (anomalias).

A Figura 3 exibe um modelo em redes de Petri, que será usado como base de exemplo para o desenvolvimento de um gerador de espaço de estados. Neste modelo existem quatro transições que ao disparar, o modelo deslocar-se-á para quatro estados diferentes. O gerador analisa cada um dos quatro estados e verifica, quais as condições de cada um deles (existência de *deadlocks*). O modelo encontra-se correto se o gerador de espaço de estados não detetar nenhuma anomalia (Figura 13).

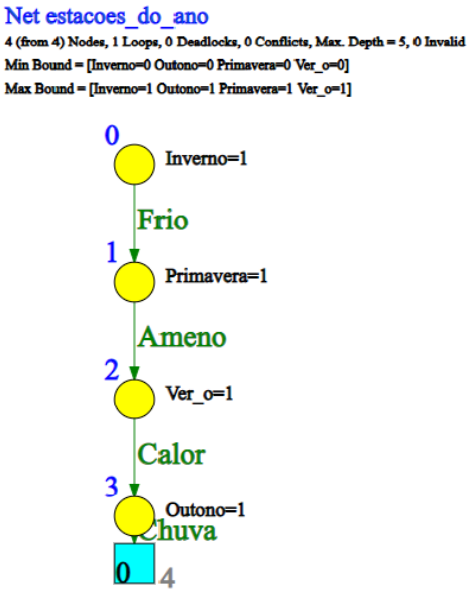


Figura 13: Espaço de Estado

2.2.4 Geração automática de código

Existem diversos geradores de código automático disponíveis no ambiente Web IOPT-Flow e IOPT-Tools. Estes geradores têm o objetivo de produzir ficheiros de uma dada linguagem fonte para um determinado sistema de *hardware*. A geração automática tem a vantagem de reduzir o tempo de desenvolvimento de código, bem como reduz os erros de codificação. O código gerado deve implementar a semântica de execução do modelo original na linguagem a que se destina. Como se pode ver em [4], já era possível gerar código em linguagem C, JavaScript e VHDL, no ambiente Web IOPT-Flow. Como resultado do trabalho apresentado neste documento, passou a ser possível também gerar automaticamente código Python (Figura 14).

A Figura 14 mostra um conjunto de opções que o utilizador poderá escolher, para gerar o seu código. Nesta Figura é possível ver os geradores de código C, VHDL (código monolítico, código modelar), JavaScript, Python (desenvolvido neste trabalho) e semântica do modelo (XML).

Figura 14: Opções de geração de código automática

O gerador de código automático no ambiente Web IOPT-Flow, é implementado utilizando as transformações XSLT. Estas transformações, permitem gerar *software* (C, JavaScript, Python) ou *hardware* (VHDL), capazes de correr em sistemas ciber-físicos/embutidos. De acordo com o artigo [25], a geração automática de código é realizada em três passos:

– **1º Passo**: O editor de modelos IOPT-Flow analisa os arcos que definem as dependências entre os elementos de Dataflow e determina a sequência de calculo correta;

– **2º Passo**: Gera a semântica de execução, que descreve o funcionamento do modelo (XML);

– **3º Passo**: Transformar a semântica gerada (XML) para a sintaxe de linguagem C, VHDL, JavaScript e agora Python;

2.2.5 PNML

O PNML (*Petri Net Mark-up Language*), é um formato de ficheiro XML padrão, que define a sintaxe de representação das redes de Petri. Apesar de não ser utilizado em todas as plataformas de redes de Petri, é o formato usado para transferir dados entre diversas ferramentas diferentes [6]. Nos ambientes IOPT-Flow e IOPT-Tools, existe a possibilidade de converter o formato XML escolhido para PNML usando transformações XSL simples [4].

De acordo com [4], a IOPT-Flow possui uma transformação que permite extrair partes das redes de Petri de um modelo em DS-Pnet criado pelo utilizador e, converte essas partes para um modelo IOPT armazenado num ficheiro PNML. Se o editor IOPT-Flow detetar um documento XML formatado usando a sintaxe em PNML, invoca automaticamente uma transformação XSL que converte o modelo IOPT num modelo DS-Pnet.

2.3 Trabalhos relacionados

2.3.1 Geradores de código a partir de outras linguagens gráficas

Nesta secção serão mencionados algumas plataformas que utilizam formalismos baseados em redes de Petri e Fluxo de dados, para gerar automaticamente código. Estas aplicações são capazes de gerar automaticamente código para linguagens de descrição *hardware* ou para linguagem de programação de alto nível, utilizadas no desenvolvimento de sistemas embutidos ou ciber-físicos.

Pode-se notar que os programas baseados em ambientes Web (IOPT-Flow, IOPT-Tools, *Signal Interpretation Model*), permitem desenvolver sistemas embutidos/ciber-físicos, através da simulação, verificação e geração automática de código, oferecendo todas as ferramentas necessárias sem custos de utilização. As ferramentas com interface Web, têm a vantagem de poder ser facilmente acedida em qualquer lugar, desde que haja ligação à Internet.

Aplicações como Matlab/Simulink, Labview utilizam um formalismo de modelação/programação apelativos, sendo bastante populares entre a comunidade académica. Estes programas possuem um vasto e variado conjunto de ferramentas, contudo é necessário obter uma licença de uso perpétuo ou anual, que requer custos substanciais.

2.3.1.1 Matlab/Simulink

Matlab (*matrix laboratory*) [34] [35] é uma ferramenta de *software* e linguagem de programação de alto nível, utilizada para cálculos numéricos/científicos e análise de dados, criado pela empresa *MathWorks*. Este ambiente de programação interativo de linguagem matemática dispõe de diversos recursos tais como, análise de dados, gráficos, programação, criação de aplicações de desktop e Web, interface com linguagens externas, paralelização de simulações, partilha de programas e conexão de dados com *hardware*.

Simulink é uma ferramenta de simulação, acedida através da janela de comandos do Matlab, baseada em diagramas de blocos e fluxogramas, para realizar simulações em sistemas não lineares, modelizar e analisar modelos dinâmicos e matemáticos.

Existem várias ferramentas no Matlab/Simulink, que possibilitam a geração automática de código para várias linguagens de *software* e *hardware*. Estas ferramentas podem ser acedidas através da barra de ferramentas designada por "*APPS*", ou pelo ambiente Simulink. É possível enumerar as ferramentas por: MATLAB *Coder* [32], *HDL Coder* [33], *Petri Net Toolbox* [34] [35], *Embedded Coder* [36], Simulink *Coder* [37].

Embedded Coder – Este gerador transforma diagramas de blocos em linguagem C e C++ (Tabela 8), para uso em sistemas embutidos. O *Embedded Coder* é uma extensão das capacidades oferecidas pelo MATLAB *Coder* [32] e Simulink *Coder* [37], sendo possível realizar otimizações para controlo rigoroso das funções, arquivos e dados. O código gerado por esta ferramenta pode ser compilado e corrido em qualquer tipo de processador e dispositivos de *hardware* [36].

MATLAB *Coder* – Este gerador transforma programas de Matlab em linguagens C e C++ [31] (Tabela 8), para correr em microcontroladores e plataformas como Arduino e *Raspberry Pi* [32].

Simulink *Coder* – Gerador de código C e C++ [31] (Tabela 8), a partir de diagramas Simulink, gráficos de fluxo, diagramas de estado e funções de Matlab para execuções em tempo real. O código gerado também permite controlar dispositivos tais como microcontroladores e plataformas como Arduino e *Raspberry Pi* [37].

HDL Coder – Este gerador utiliza *scripts* de Matlab, diagramas de blocos e gráficos de fluxo de estados para gerar descrições de *hardware* na linguagem de programação VHDL (Tabela 8). A transformação de código realizada pela ferramenta *HDL Coder*, gera algoritmos a partir da linguagem Matlab e Simulink, para criar protótipos em plataformas de *hardware* como FPGA/ASIC/SOC [33].

Petri Net Toolbox – Ferramenta de *software* composta por uma interface gráfica de utilizador (GUI), que permite desenhar, simular, armazenar e projetar sistemas e eventos discretos baseados em redes de Petri (Tabela 8). Esta ferramenta contém cinco tipos de redes de Petri não autónomas, descritas por: redes não temporizadas, redes temporizadas por transição, temporizadas por lugar, estocásticas e estocásticas generalizadas. Esta ferramenta é acedida através da janela de comandos do Matlab ao digitar o comando "pntool". Existem dois modos de análise que definem a caixa de ferramentas de redes de Petri: modo de Desenho, onde o utilizador pode construir modelos novos de redes de Petri ou reutilizar propriedades dos modelos existentes; modo de exploração, que permite o acesso do utilizador às ferramentas de simulação, análise e desenho de ferramentas [34] [35] (Tabela 8).

Ferramentas Matlab	Linguagem
MATLAB *Coder*	C/C++
Embedded Coder	C/C++
Simulink *Coder*	C/C++
HDL Coder	Código VHDL
Petri Net Toolbox	Redes de Petri

Tabela 8: Ferramentas geradoras de código

2.3.1.2 LabVIEW

O LabView [37] é uma linguagem de programação (denominada por "G"), gráfica baseada em fluxo de dados, onde os fluxos determinam a execução das aplicações. Esta linguagem (originária da *National Instruments*) baseia-se no formalismo de fluxo de dados para aplicar a geração automática de código, de forma a construir protótipos de controlo e aquisição de dados, para sistemas embutidos. Esta aplicação fornece um ambiente de simulação e geração de código aplicável a *software* e *hardware* (FPGA). Para implementações em *hardware* utiliza-se uma linguagem de descrição HDL – VHDL [39] e para implementações em *software* utiliza-se ferramentas para linguagem C [40].

Os programas criados através da ferramenta de LabView são denominados por Instrumentos Virtuais (VI). Estes programas são compostos por três componentes principais: painel frontal, diagrama de blocos e painel de ícones e conectores.

Painel Frontal – Corresponde ao painel de interface entre o utilizador e o programa. Este painel é constituído por simuladores de dispositivos de entrada de instrumentos, que recebem e distribuem os dados para o diagrama de blocos do instrumento virtual (controlo). Contém ainda

simuladores de dispositivos de saída de instrumentos, onde estes exibem os dados recebidos do diagrama de blocos (indicadores) [38].

Diagrama de blocos – Corresponde ao painel que incorpora o código gráfico. Este painel é constituído por ícones terminais e conectores. Os ícones terminais correspondem à interface entre dados trocados pelo painel frontal e o diagrama de blocos. Estes ícones podem ser caracterizados por ícones de controlo e ícones indicadores. A colocação de ícones de controlo ou ícones indicadores no painel frontal, faz com que sejam adicionados ícones terminais no diagrama de blocos [38].

Ícones e conectores – É possível utilizar um instrumento virtual dentro de outro instrumento virtual, através da utilização do painel frontal e diagrama de blocos para construir um modelo, composto pelos seus respetivos ícones e conectores, gerando assim submodelos de instrumentos virtuais (subVI). Um subVI corresponde a um ícone terminal, composto por uma sub-rotina em linguagem de programação baseada em texto [38].

Esta aplicação possui uma caixa de ferramentas, que permite ao utilizador realizar a geração de código LabView de forma automática. Esta caixa de ferramentas fornece aplicações para o desenvolvimento de código, para fácil utilização e interpretação do utilizador, pois o código criado é semelhante ao código Labview padrão [39].

2.3.1.3 *Signal Interpretation Models* (SIM)

Esta aplicação consiste num editor gratuito baseado na Web, para modelos de interpretação de sinal (SIM) [41]. O SIM (disponível em: http://gres.uninova.pt/SIM-Tools/), é um formalismo criado para controlar ocorrências de eventos com base na variação de sinais. O objetivo deste formalismo passa por validar, especificar e interpretar sinais de entrada do sistema ou sinais internos do sistema. Este ambiente segue a mesma abordagem utilizada em IOPT-Flow e IOPT-Tools, baseado na Web utilizando princípios AJAX (*Asynchronous JavaScript and XML*).

Para destacar as diferenças entre as ferramentas de modelação gráfica baseadas na Web já mencionadas, pode-se referir que as IOPT-Tools utilizam um formalismo baseado em redes de Petri [28], as IOPT-Flow utilizam um ambiente baseado nas DS-Pnets [5] e as SIM-Tools usam um formalismo baseado na modelação SIM [41].

A SIM-Tools tem a capacidade de gerar código automático em linguagem de programação C e além disso também permite gerar componentes DS-Pnet, através de ficheiros em formato XML, capazes de serem transferidos para o ambiente IOPT-Flow.

2.3.2 Geradores existentes

A programação em código C e em código Python é usada para correr aplicações em microcontroladores, computadores industriais e plataformas como Arduino ou *Raspberry Pi*. O ambiente de desenvolvimento integrado do Arduíno (IDE) [7] não suporta Python, como alternativa o utilizador pode usar:

– O OpenMV, é uma plataforma que suporta programação de placas de Arduino com *MicroPython*. O *MicroPython* é uma implementação da linguagem de programação *Python3* composta por um pequeno subconjunto de bibliotecas padrão Python, projetada para correr em microcontroladores [8] [9] [12].

O JavaScript é uma linguagem de alto nível capaz de manipular o aspeto e o comportamentos das plataformas Web. No simulador DS-Pnet é usado para correr modelos diretamente no browser Web. O JavaScript é uma linguagem de programação usada para desenvolver "frontend", que correspondem à parte visível das aplicações Web, isto é, a geração de ambientes gráficos que permitam interagir com o utilizador.

O VHDL (*VHSIC Hardware Description Language*) é uma linguagem de descrição de *Hardware*, utilizada para conceber circuitos ASIC ou FPGA. Um microchip do tipo ASIC é projetado para conceber uma aplicação especifica, não tendo possibilidade de ser reprogramado ou modificado depois de produzido. Um microchip do tipo FPGA é projetado para conceber várias tarefas, tendo a possibilidade de ser programado e reprogramado.

2.3.2.1 Geradores de Código C

A linguagem C é uma linguagem de alto nível que pode lidar diretamente com dispositivos de *hardware*, incluindo plataformas programáveis de eletrónica de baixo custo como o Arduino [7]. Esta linguagem, pertence a uma seleção limitada de opções de geração de código automático disponível no ambiente Web IOPT-Flow (Figura 14) [4] e IOPT-Tools (Figura 15) [29]. Ambos os ambientes Web, quando executam a geração automática de código para a linguagem C, geram um arquivo compactado em formato ZIP.

As funcionalidades da ferramenta de geração de código C [6] no ambiente IOPT-Flow são:

- Funções que inicializam os pinos (pinMode) para controlar o plataformas de eletrónica (interface de entrada/saída);
- Produção de ambientes gráficos de interface com o utilizador;
- Comunicação entre componentes distribuídos remotamente;
- Geração a semântica de execução do modelo;

2.3.2.2 Gerador de Código JavaScript

A linguagem JavaScript é uma linguagem de programação orientada a objetos que pode ser gerada pelo ambiente IOPT-Flow [22] [25]. Quando o modelo é transformado, o programa cria um ficheiro único com todos os dados do modelo desenvolvido. A função principal do gerador automático de código JavaScript no ambiente de desenvolvimento Web IOPT-Flow, consiste na simulação de modelos DS-Pnet diretamente do browser Web [4]. A janela de simulação da Figura 8: b) permite correr os modelos criados, de forma continua ou passo-a-passo, podendo mesmo associar interrupções nas transições ou operações de fluxo de dados. O código JavaScript gerado automaticamente é usado pelo simulador IOPT-Flow para efetuar cada passo de execução.

Apesar de a sintaxe ser semelhante, o objetivo do código Python é controlar sistemas embutidos/ciber-físicos, enquanto que o objetivo do código JavaScript é elaborar plataformas Web.

2.3.2.3 Gerador de Código VHDL

A linguagem VHDL [30] é uma linguagem bastante útil para descrever circuitos de *hardware*. A geração de código para este tipo de dispositivos permite implementar modelos IOPT-Flow em *hardware* (ASIC's, FPGA's). Esta linguagem apresenta-se no ambiente IOPT-Flow [22] e IOPT-Tools [26], como a única linguagem direcionada para implementações em *hardware*.

O código VHDL produzido pelo gerador automático, necessita de três sinais de entrada (*reset*, *enable* e *clock*) para além dos sinais e eventos existentes nos modelos DS-Pnet.

O ambiente IOPT-Flow oferece duas formas de gerar código VHDL. De acordo com a Figura 14, é possível ver a opção para gerar código monolítico e para códigos modelar. O código monolítico é constituído por um único modulo que contem a implementação de todo o modelo. O código modelar é constituído por um ficheiro ZIP (gerado automaticamente pelo gerador de código) composto por um módulo principal (*main*) e diversos módulos adicionais associados aos componentes DS-Pnet.

O formalismo UML (*Unified Modeling Language*) usa fluxogramas (*statecharts*) para controlar sistemas embutidos, realizar simulações e gerar código automaticamente para a linguagem escolhida de acordo com as opções disponíveis [30].

2.4 IOPT-Tools

2.4.1 Ambiente IOPT-Tools

A construção do ambiente IOPT-Flow, foi inspirado em trabalhos realizados em IOPT-Tools, herdando muitos aspetos e soluções de design para o desenvolvimento dos modelos [4]. O ambiente IOPT-Tools [27] é uma plataforma Web de livre acesso(disponível em: http://gres.uninova.pt/IOPT-Tools-V1.2/login.php), que permite ao utilizador criar modelos gráficos para aplicações em sistemas embutidos. Contudo as ferramentas têm conceitos diferentes, enquanto as IOPT-Flow utilizam modelos DS-Pnet (redes de Petri e *Dataflows*), as IOPT-Tools utilizam as redes de Petri IOPT para construir os seus modelos. Os sinais de entrada e saída de ambas as plataformas, permitem interagir com elementos externos de hardware.

Este ambiente utiliza o princípio AJAX [26] para implementar as ferramentas interativas diretamente no navegador Web, como por exemplo a janela de interface (Figura 15) ou a janela de trabalho para a construção dos modelos (Figura 16). As operações de armazenamento de arquivos e processamento são arquivadas no servidor [26]. É importante notar que para o bom funcionamento do ambiente Web, é recomendável correr os modelos num *browser* moderno.

2.4.2 Interface IOPT-Tools

A Figura 15, mostra o ambiente gráfico de interface com o utilizador. Esta página exibe o modelo IOPT construído e na secção inferior mostra uma barra ferramentas horizontal com várias

opções. Dentro dessas opções o utilizador pode abrir o ambiente Web de compilação ou edição do modelo, pode descarregar o modelo para o computador ou pode optar por gerar o código automático do modelo para linguagem C [6], VHDL [27] ou código *Simulink*, pode utilizar o gerador e visualizador do espaço de estados [28] ou pode escolher um sistema de consulta do modelo.

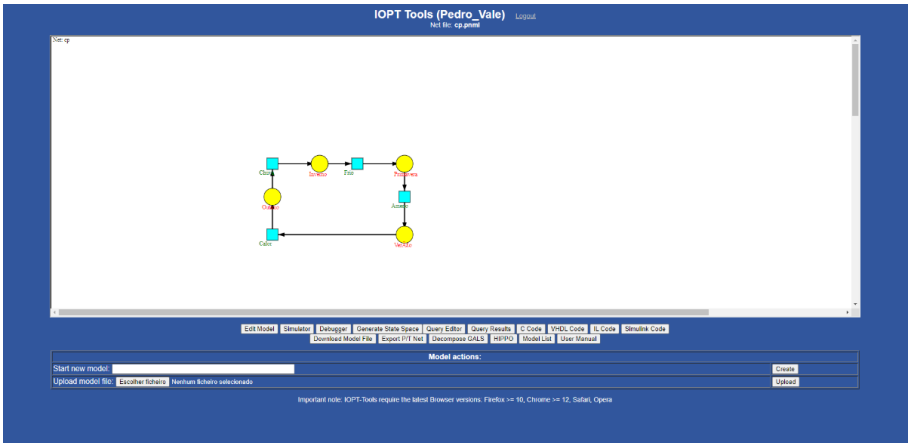


Figura 15: Janela de interface de IOPT-Tools

A semelhança entre ambos os ambientes (IOPT-Flow e IOPT-Tools) é notável, a começar pela estrutura do ambiente de trabalho (Figura 16), no painel do lado esquerdo localiza-se a barra de ferramentas formada pelos ícones de edição de modelos, no centro encontra-se o local de construção dos modelos e no lado direito encontra-se um painel de edição das propriedades dos objetos [26]. Tal como no IOPT-Flow, a barra de ferramentas das IOPT-Tools é composta por transições, lugares, arcos, sinais de entrada e saída e eventos de entrada e saída. O painel do lado direito permite personalizar as propriedades dos objetos das redes de Petri. Os lugares, transições e arcos podem ter os seus parâmetros de propriedades alterados de acordo com a descrição do modelo construído. É nesta janela que o utilizador consegue definir o peso do arco, o número de *tokens* dos lugares, a prioridade das transições e o nome dos objetos (Figura 16).

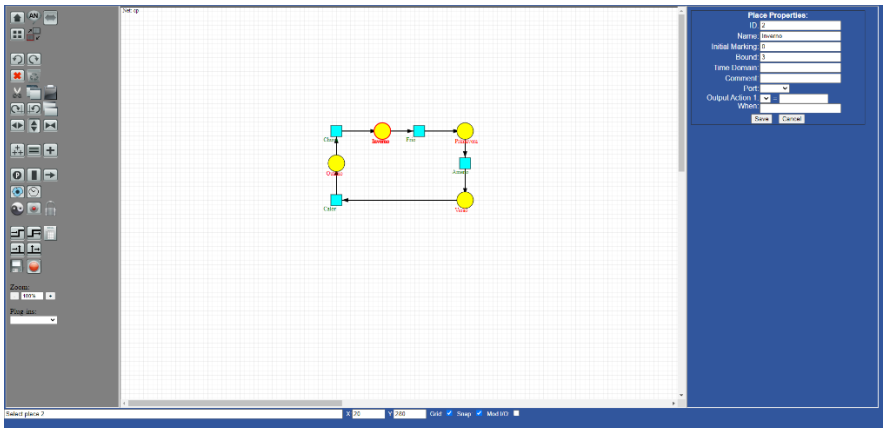


Figura 16: Janela de Trabalho IOPT-Tools

A janela de simulação dos modelos (Figura 17), comanda a simulação em tempo real, permitindo execuções continuas ou execuções passo-a-passo, tal como no ambiente IOPT-Flow.

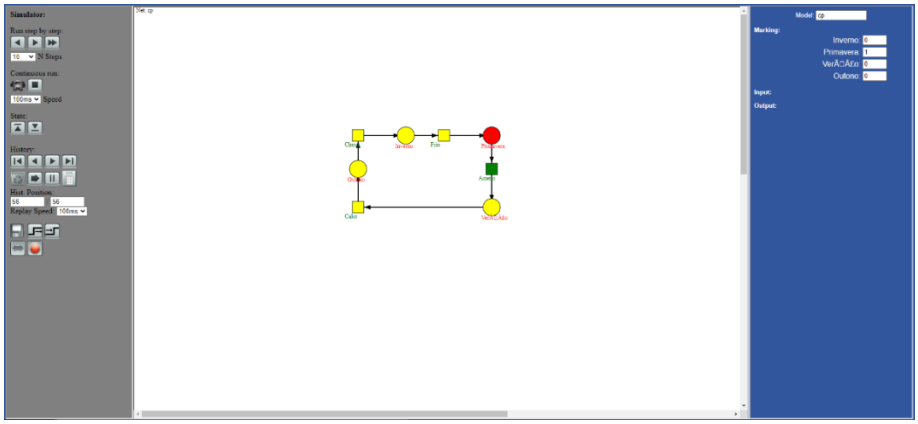


Figura 17: Janela de simulação do ambiente IOPT-Tools

O ambiente Web IOPT-Tools e o ambiente IOPT-Flow são compostos por *frameworks* [29], que contêm ferramentas de verificação de modelos para desenvolver controladores de sistemas embutidos. O sistema de geração de código automático do ambiente IOPT-Tools utiliza transformações XSL para abrir arquivos XML contendo modelos IOPT em formato PNML, para criar código-fonte de *software*. O sistema de geração de código automático do ambiente IOPT-Flow, para além de permitir converter modelos IOPT em formato PNML (pois o formato PNML apenas transforma modelos inteiramente compostos por redes de Petri), também permite aplicar transformações XSL para abrir arquivos XML contendo modelos DS-Pnet em formato XSLT, para criar código fonte de *software* e *hardware*. A utilização das ferramentas de geração, permitem implementar a semântica dos modelos, para as linguagens de programação C [6] ou linguagem de descrição de *hardware* VHDL [28], que depois são inseridas em plataformas físicas (sistemas embutidos e ciber-físicos).

2.4.3 Globally Asynchronous Logically Synchronous

Um sistema GALS (*Globally Asynchronous Locally Synchronous*) é composto por um conjunto de componentes síncronos, que comunicam entre si, com o objetivo de reunir os benefícios entre sistemas síncronos e assíncronos [29]. Esta arquitetura é utilizada no ambiente IOPT-Tools com a função de dividir um modelo de redes de Petri em diversas partes/submodelos (IOPT). Cada submodelo corre num local diferente (*Time Domain* diferente), mas os canais síncronos/assíncronos garantem que os submodelos continuam a ter um comportamento idêntico.

2.5 Linguagem Python

2.5.1 Introdução à linguagem Python

O Python [43] é uma linguagem de programação *Open-Source* de alto nível, inventada no final da década de 1980, pelo cientista holandês Guido van Rossum. O nome da linguagem Python, foi inspirada no grupo de comédia inglês *Monty Python*, embora haja quem associe também o nome ao réptil. De acordo com a avaliação realizada pelo IEEE, a linguagem Python mantém-se como uma das mais populares utilizadas pelos desenvolvedores ocupando a primeira posição no ranking de 2022 [42]. Uma das mais valias desta linguagem é a facilidade de aprendizagem,

simplicidade na programação, suporte para multiplataformas (vários sistemas operativos têm capacidade de correr programas em Python), forte comunidade, capacidade de criação de *scripts* para controlar sistemas de *hardware*, possibilidade de criação de modelos de inteligência artificial, vasta gama de bibliotecas já criada pelos utilizadores e diversas opções de ambientes de desenvolvimento para a criação de vários projetos (PyCharm, VS Code, Spyder, Sublime text, etc.).

Esta é uma linguagem *case sensitive*, voltada para os conceitos de classes e heranças (orientada a objetos), com uma sintaxe intuitiva e simples de aprender, que permite implementar programas para Web, desktop, dispositivos móveis, sistemas embutidos, etc.

Existem algumas plataformas que já realizam a transformação de código automática para a linguagem Python, como é o caso do Simulink [44] e CPN-Tools (*Coloured Petri Net Tools*) [45], que se baseiam respetivamente no formalismo de diagramas de blocos [44] e redes de Petri coloridas [45] para construir os modelos.

Como anteriormente referido, a linguagem Python tem uma popularidade muito grande entre a comunidade de programadores. Devido a estas razões, esta linguagem de programação é vista como uma mais-valia para a ferramenta IOPT-Flow.net, pois a sua versatilidade permite executar os programas em placas SBC (*Single board Computer*), desde que possuam interpretadores de Python (como é o caso das placas *Raspberry Pi* [46]). A versão usada para a realização das atividades experimentais neste relatório foi Python 3.9.2.

2.5.2 Fundamentos do Python

A linguagem de programação Python é considerada uma linguagem de *script*, pois os programas em Python são executados diretamente usando um interpretador sem haver necessidade de compilação. Esta linguagem também permite realizar processamento interativo, em que o utilizador escreve frases com instruções pelo teclado que o computador executa imediatamente e apresenta o resultado no monitor. Após a ação estar realizada, o utilizador poderá fornecer outra frase ao computador e repetir este ciclo [43].

Os tipos de dados da linguagem Python são classificados por tipos elementares e tipos estruturados (subcapítulo: 2.3.2.4 Tipos de Dados Estruturados). Dentro do grupo dos tipos elementares, existem tipos de dados para inteiros, tipos de dados para números reais e tipos de dados lógicos (Tabela 9).

Tipo Inteiro	**int**	**Números inteiros sem parte decimal**
Tipo Real	**float**	**Números reais com parte decimal**
Tipo Lógico	**Bool**	**Assume dois valores *True* ou *False***
Tipo *string*	**Str**	**Composto por caracteres que formam palavras**

Tabela 9: Tipos de variáveis

A linguagem Python é constituída operadores aritméticos, operadores de atribuição, operadores lógicos, operadores de comparação, operadores de identidade, operadores bit-a-bit [53].

Durante a execução de um programa, é necessário angariar valores provenientes do exterior (*input*), para realizar manipulações de informação. A obtenção de dados do exterior é realizada através de operações de leitura, que transmitem a informação para o programa. Após a realização da manipulação de informação, é essencial o computador comunicar ao exterior os resultados alcançados. A comunicação do programa é realizada através de operações de escrita, que transmitem a informação do programa [43].

2.3.2.1 Ambiente de Desenvolvimento

Para facilitar a execução dos *scripts* gerados, é necessário instalar um ambiente de desenvolvimento para a plataforma de hardware usada. Existem diversas aplicações que permitem editar e implementar código Python, pelo que o utilizador poderá optar por trabalhar nas seguintes aplicações:

IDLE – Ambiente de desenvolvimento de programas em Python. Nesta plataforma o utilizador poderá realizar os seus programas e no fim executá-los no numa placa eletrónica. Uma das vantagens desta aplicação é a indentação automática [10].

PyCharm – Ambiente de desenvolvimento integrado (IDE) para programação Python. Este ambiente contém duas versões, uma versão gratuita (*Community*) e outra com custos (*Professional*). Uma das vantagens desta aplicação é o auxílio na construção de código [49].

VS Code – Editor de código desenvolvido pela Microsoft com suporte para operações de *debug* e execução de tarefas. Esta aplicação é gratuita e de livre acesso por qualquer utilizador. Tem a vantagem de possuir diversas extensões que aumentam as funcionalidades do programa [50].

Spyder – Ambiente de desenvolvimento *open-source* de código Python. Composto por diversas ferramentas como explorador de variáveis, painel gráfico de exibição de Figuras e imagens estáticas, consola para interação com o dado dentro do interpretador, ferramenta de *debugging* e ferramenta de ajuda [51].

Jupyter Notebook – Ambiente de desenvolvimento Web *open-source*. Tem a vantagem de gerar e partilhar ficheiros de código para trabalho online e em equipa [52].

2.3.2.2 Instruções de Seleção

A instrução *if* [43] [44] permite tomar decisões de escolha de execução de instruções. Conforme a condição verdadeira ou falsa, o código Python implementa uma instrução diferente (Figura 18).

```
if (condicao):
    Instrucao
elif (condicao):
    Instrucao
else:
    Instrucao
```

Figura 18: Instrução if

A versão 3.10 do Python veio trazer uma nova abordagem às instruções de seleção múltipla. A nova instrução "match/case" simplifica a implementação de seleções mais complexas, oferecendo uma boa legibilidade e limpeza no código.

2.3.2.3 Ciclos

Em programação, o objetivo de um ciclo é executar uma sequência de instruções de forma repetida. Um ciclo [43] [48] [53] é composto por uma sequência de instruções, denominada por corpo do ciclo e por uma estrutura de controlo que indica o número de vezes que o corpo do ciclo é implementado. Existem duas instruções que especificam os ciclos na linguagem Python:

– Instrução *While*: Realiza um conjunto de instruções repetidamente se a expressão for verdadeira (*while*), caso contrário realiza a instrução opcional *else* (Figura 19).

```
while (expressao):
    Instrucao
else:
    Instrucao
```

Figura 19: Instrução While

– Instrução for: ciclo que pode ser definido por um intervalo de valores ou lista de itens, que opcionalmente pode realizar um *else* quando a sequência termina (Figura 20).

```
for variavel in sequencia:
    Instrucao
else:
    Instrucao
```

Figura 20: Instrução for

Para controlar a execução dos ciclos, a linguagem Python possui as seguintes instruções:

– *break*: Termina o ciclo mesmo se a condição *while* for verdadeira

– *continue*: Termina a presente iteração e passa para a ordem seguinte

– *pass*: Usado para quando o corpo do ciclo não contém instruções

2.3.2.4 Tipo de dados estruturados

Um tipo de dados estruturados [43] [48] [53] é composto por um conjunto de elementos. Estes elementos podem ser constituídos por valores de texto (*str*), valores inteiros (*int*), valores reais (*float*) ou valores booleanos (*bool*). Existem várias categorias de tipos de dados estruturados, que permite realizar operações diferentes. Os principais tipos de estrutura de dados são Sets/Conjuntos, Listas, Dicionários e Tuplos (Tabela 10).

Tipo de estrutura de dados	Especificação em Python	Tipos Mutáveis e Imutáveis	Exemplo de declaração de estrutura
Listas	*list*	Mutável	`l = list (("a","b","c"))` `l = ["a","b","c"]`
Tuplos	*tuple*	Imutável	`t = tuple (("a","b","c"))` `t = ("a","b","c")`
Sets/Conjuntos	*set*	Imutável	`s = set (("a","b","c"))` `s = {"a","b","c"}`
Dicionários	*dict*	Mutável	`d = dict (a=1,b=2,c=3)` `d = {"a": 1,"b":2,"c":3}`

Tabela 10: Tipos de estrutura de dados

Listas – Tem o objetivo de armazenar diversos elementos (em sequência) numa só variável, em que cada elemento é identificado pela sua posição na lista. O primeiro elemento é denominado por índice zero e por cada elemento inserido na lista, o valor do índice vai aumentando. Esta estrutura é mutável, logo é possível alterar, adicionar e remover elementos [43] [53].

Tuplos – À semelhança das listas, os tuplos permitem armazenar vários elementos em sequência de diversos tipos de dados numa só variável. Os elementos inseridos nesta estrutura de dados, são imutáveis, pois não podem ser alterados depois de definidos. [43] [53].

Sets/Conjuntos – Usado para armazenar vários elementos numa variável única. Esta estrutura possui um conjunto de elementos de colocação desordenada e não pode conter elementos duplicados. Os Sets/Conjuntos são parcialmente imutáveis, isto é, os elementos definidos não podem ser alterados, mas é possível removê-los e adicionar novos [53].

Dicionário – Armazena múltiplos valores em pares "chave:valor", numa variável única. A chave funciona como um identificador imutável. Esta estrutura possui uma coleção de elementos parcialmente imutáveis, colocados de forma desordenada, não permitindo elementos com chaves duplicadas [43] [53].

Com características Imutáveis também se pode considerar os números e as *strings*.

2.3.2.5 Tratamento de Erros

Existem dois tipos distintos de erros: erros de sintaxe e erros de exceção. Os erros de sintaxe são mais comuns e quando são encontrados, o erro é reportado com uma cópia da linha do

problema e indica o local do erro. Os erros de exceção são detetados durante a execução, independentemente de a expressão associada estar ou não correta.

O tratamento de erros é suportado pelas instruções principais *try* e *except* e pelas instruções secundárias *else* e *finally*. A palavra *try* permite definir blocos de instruções que ao serem implementadas, podem originar erros. A palavra *except* define as instruções que são corridas quando o erro acontece. A palavra *else* corre o código quando não existe ocorrência de erros. A palavra *finally* efetua o código independentemente do resultado dos blocos *try* e *except* (Figura 21) [48] [53].

```
try:
    print("a")
except:
    print("b")
else:
    print("c")
finally:
    print("d")
```

Figura 21: Tratamento de erros

2.3.2.6 Funções

As funções [48] [53] são pedaços de código que implementam uma determinada tarefa sempre que são chamados. As funções têm capacidade de receber argumentos como entradas provenientes de outras instruções e retornar dados como resultados. Uma função é definida através da expressão *def*, seguida pelo nome da função e dos parâmetros formais, seguindo-se o corpo da função contendo as respetivas instruções. Tudo o que se passa dentro do corpo de uma função, só é conhecido dentro dessa mesma função (Figura 22).

```
def nome(parâmetro):
    print("ola")
```

Figura 22: Função exemplo

2.3.2.7 Módulo

Um módulo [48] [53] é um arquivo composto por definições e instruções em Python, com o sufixo "*.py*", que funciona como uma biblioteca. Um modulo, pode conter instruções executáveis e possuir definições de variáveis e funções. As definições de um módulo podem ser importadas por outros módulos, permitindo uma melhor organização e divisão de conteúdo

programático. Para criar um módulo basta guardar o programa num ficheiro com a extensão“*.py*”. Para importar o módulo, deve-se usar instrução *import* e o nome do ficheiro a importar (Figura 23).

exemplo.py: nome do arquivo

```
def expressao (numero):
    print("exemplo:"+numero)
```

Importar arquivo

```
import exemplo

exemplo.expressao("1")
```

Figura 23: Módulo exemplo

2.3.2.8 Classes e Objetos

Uma das características da linguagem Python é a sua orientação a objetos, fazendo com que quase tudo em Python seja considerado um objeto com as suas propriedades e métodos (Figura 24). Estes objetos possuem um conjunto de variáveis internas e um conjunto de operações (métodos). As classes [48] [53] têm o objetivo de agrupar dados e funcionalidades numa só entidade (objetos).

Criar Classe

```
class exemplo:

    X = "a"
```

Criar Objeto

```
nome = exemplo()

print(nome.X)
```

Figura 24: Classe e Objeto exemplo

As classes possuem uma função de iniciação que é executada automaticamente sempre que novos objetos são criados. Esta função denomina-se por “__init__()”, que tem como objetivo atribuir valores às propriedades do objeto [48] [53]. A palavra *self* é usada para identificar uma instância da classe e tem o objetivo de permitir o acesso às variáveis pertencentes à classe respetiva (Figura 25).

Criar Classe

```
class exemplo:

    def __init__(self, a, b):
        self.a = a
        self.b = b
```

Criar Objeto

```
nome = exemplo(1,2)

print(nome.a)
print(nome.b)
```

Figura 25: Função init exemplo

O utilizador pode interagir com os objetos através de métodos, que são funções definidas na classe e realizam tarefas relacionadas com o objeto, respondendo às solicitações do utilizador (Figura 26).

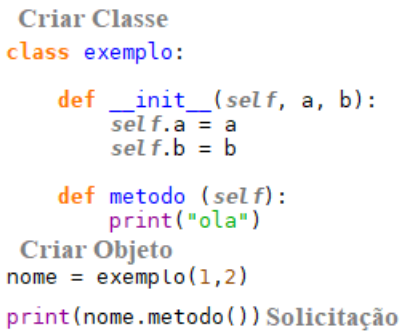


Figura 26: Método exemplo

2.6 Raspberry Pi

O *Raspberry Pi* é um dispositivo de placa única (SBC), com características de um computador, construído sobre uma placa de circuitos eletrónicos. O que define esta placa SBC é o facto de ser um computador relativamente poderoso, de baixo custo e que qualquer um pode simplesmente ligar a dispositivos de *hardware* para começar a programar. Um dos objetivos principais desta placa é a habilidade de poder tornar a programação mais simples e por esta razão, os seus criadores escolheram a linguagem Python como uma parte integral do sistema operativo do *Raspberry Pi* [46] [54].

A interação entre a placa *Raspberry Pi* com o utilizador, é feita através de dispositivos de entrada e saída como: um teclado, rato, um monitor e cartão SD (*Secure Digital*). Com isto, o utilizador pode ter acesso a qualquer aplicação que esteja disponível no sistema operativo. O cartão de memória SD funciona como um disco rígido para armazenar os dados e o sistema operativo para inicializar o computador. O utilizador também tem a capacidade de poder instalar aplicações se forem compatíveis com o ambiente operativo definido. O *Raspberry Pi* é suportado por diversos sistemas operativos, mas o utilizado para realizar a parte experimental foi *Raspberry Pi OS* (antigo *Raspbian* [47]).

Existem diversos modelos de placas *Raspberry Pi*, mas as principais são: *Raspberry Pi Zero, Raspberry Pi 1, Raspberry Pi 2B, Raspberry Pi 3, Raspberry Pi 4* e *Raspberry Pi 400*. O modelo utilizado para testar o código produzido no gerador de código foi *Raspberry Pi 4 Model B 2GB RAM*, mas qualquer dispositivo da *Raspberry* que tenha instalada as ferramentas da linguagem Python, consegue correr o código gerado automaticamente. Cada placa possui *hardware* com características diferentes, mas há uma base comum que assegura a compatibilidade de *software* [54] (Tabela 11).

Características	Definição
CPU	Unidade central de processamento corresponde ao cérebro do *Raspberry Pi*, que tem a funcionalidade de implementar instruções usando operações lógicas e matemáticas
HDMI	Porta de Interface multimédia de alta-definição que permite transmissão de conteúdo áudio e vídeo, através da conexão entre um dispositivo *Raspberry Pi* e um monitor.
Memória RAM	Memória de acesso aleatório. Local de armazenamento temporário

	de informações (memória volátil) mas com rápida recuperação.
Ethernet	Disponível apenas nos modelos B do *Raspberry Pi*, permite conectar a placa, a uma rede de internet utilizando fios. O conector utilizado é RJ45
Ranhura Cartão SD	*Slot* que permite o encaixe do cartão de memória SD (*Secure Digital*)
GPU	Unidade de processamento gráfico, tem a finalidade de manipular e alterar a memória para a criação de imagens do *Raspberry Pi*
GPIO	Pinos que têm a função de controlar sinais elétricos de outros dispositivos, de acordo com a programação definida pelo utilizador
LEDs	Díodos emissores de luz que mostram o estado atual da placa *Raspberry Pi*
Porta USB	Portas que permitem a conexão a dispositivos de *hardware* como o teclado e o rato.
Fonte de energia	A fonte de alimentação utiliza um cabo micro-USB que alimenta a placa a uma tensão de 5v.

Tabela 11: Características das placas Raspberry Pi

O SoCs é um circuito que integra todos os componentes necessários de um sistema eletrónico ou de computador num só chip. As placas *Raspberry Pi* utilizam um sistema num chip (SoC) com uma unidade de processamento central (CPU) compatível com a arquitetura ARM, uma unidade de processamento gráfico (GPU), portas de entrada e saída, memória interna e blocos de entradas e saídas analógicas [55].

2.7 Considerações finais

Os subcapítulos anteriores exibem artigos de relevantes para a realização deste trabalho. Estes artigos apresentam aspetos específicos que foram cruciais para a criação da ferramenta no ambiente IOPT-Flow. Esses conceitos permitem que qualquer pessoa adote as mesmas referências e utilize-as para criar outro tipo de ferramenta de geração automática para outros ambientes de desenvolvimento. Ao comparar os objetivos da dissertação com os trabalhos referenciados, é possível notar algumas semelhanças.

A dissertação [4] é de grande importância, pois apresenta a geração automática de código, partindo de modelos DS-Pnet, para linguagens Javascript, C e VHDL para aplicações em sistemas ciber-físicos, que utilizam o mesmo ambiente Web. Este artigo foi fundamental pois serviu como guia para compreender os conceitos relacionados à geração automática de código e as características do ambiente Web utilizado (IOPT-Flow).

Para o desenvolvimento deste trabalho foi necessário adquirir algumas bases sobre redes de Petri. O livro [3] apresenta uma introdução sobre as características das redes de Petri e os seus elementos principais, tais como arcos, transições e lugares. Além disso este artigo exibe as principais razões que fundamentam a modulação das redes de Petri e algumas classes alusivas à mesma linguagem.

Foi crucial ter domínio de outras ferramentas de geração automática de código. Por exemplo, o ambiente o IOPT-Tools, que segue a mesma abordagem do IOPT-Flow e foi utilizado como base para a criação da ferramenta que, a partir de agora, permitirá gerar código Python automaticamente. Os artigos [27] [28] apresentam um *framework* que usa redes de Petri com características IOPT para o desenvolvimento de controladores de sistemas embutidos. Estes artigos descrevem as ferramentas utilizadas no ambiente IOPT-Tools e o sistema de geração automática de código.

Para implementar a transformação que gera código Python, foi necessários adquirir conhecimentos sobre alguns conceitos fundamentais. Para tal, recorreu-se a um livro bastante interessante [43], que apresenta uma introdução a esta linguagem bem com os fundamentos essenciais que a definem.

Qualquer plataforma de hardware, que possua interpretadores de código Python, pode executar código gerado automaticamente. Para validar os *scripts* criados, foi utilizada a placa Raspberry Pi 4. O artigo [46] contém informação acerca da placa Raspberry Pi e a utilização da linguagem Python nesta placa.

Capítulo 3. Geração Automática de Código

Neste capítulo será descrito o processo de transformação da linguagem de formalismo DS-Pnet para a linguagem de alto nível Python, realizada através de transformação XSL a partir de documentos XML com modelos DS-Pnet. São ainda exibidas as listagens dos módulos usados no modelo exemplo bem como os troços de código em XML e XSL usados para originar os excertos em Python apresentados em cada uma das listagens.

3.1 Modelo Exemplo e Fases de Transformação

Nas IOPT-Flow existem diversos geradores de código automático, que implementam transformações de modelos gráficos DS-Pnet para uma dada linguagem de programação de alto nível. Todos os geradores de código disponíveis no ambiente Web utilizam a mesma transformação para gerar documentos XML contendo as regras da semântica de execução dos modelos. De seguida, cada gerador de código disponível, utiliza uma transformação diferente para produzir código fonte seguindo as regras de sintaxe da respetiva linguagem. Desta forma, garante-se que todos os geradores produzem código com o mesmo comportamento. O processo de conversão foi inspirado em trabalhos anteriores, onde foram implementados geradores de código para outras linguagens de programação [4].

Figura 27: Processo de Transformação

A Figura 27 apresenta um exemplo de um modelo disponível no editor Web IOPT-Flow, que apresenta quase todos os elementos que compõem a linguagem de modelação DS-Pnet. Este modelo ("test") será utilizado como base para descrever o processo de geração de código Python.

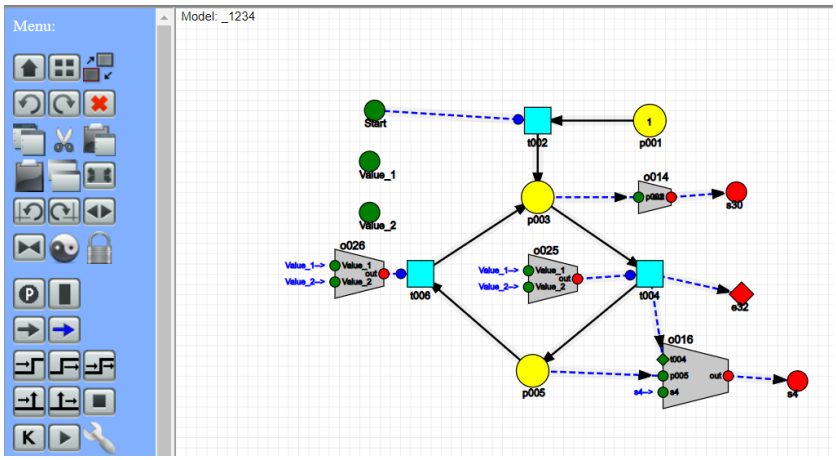


Figura 28: Exemplo de modelo IOPT-Flow

O modelo apresentado na Figura 28 é composto por 4 operações de fluxo de dados (trapezoides cinzentos) para processar os sinais de entrada e calcular os resultados de saída dos modelos, através de manipulações matemáticas e lógicas. As operações de fluxo de dados “o025” e “o026” realizam comparações entre os sinais de entrada de forma a controlar o disparo das transições. A operação de fluxo de dados “o014” aplica um atraso de 5 segundos na saída “s30”, sempre que esta muda de estado. A operação “o016”, conta o tempo em que a saída “s4” permanece ativa.

O modelo exibe uma rede de Petri composta por 3 lugares (círculos amarelos), 3 transições (quadrados azuis), 3 sinais de entrada (círculos verdes) e 2 sinais de saída (círculos vermelhos) e um evento de saída (losango vermelho). Em cada instante, a marcação dos lugares define o estado do sistema, cuja evolução é controlada através do disparo das *transições*.

A marcação inicial do modelo tem apenas uma marca no lugar “p001”. Esta marca, em conjunto com a entrada “Start”, são as condições necessárias para disparar a primeira transição “t002”.

O lugar “p003” recebe uma marca proveniente do arco de saída da transição “t002”. O operador “o014” inicia uma contagem decrescente, que fará ativar a saída “s30” passados 5 segundos.

Para disparar a transição “t004”, é necessário haver uma marca no lugar “p003” e a condição “o025” tem de ser verdadeira, isto é, o valor de “Value_1” tem de ser inferior ao valor de “Value_2”. Quando a transição “t004” dispara, limpa o contador “s4=0”, ativa o evento de saída “e32” e a marca do lugar “p003” é eliminada inserindo uma nova marca no lugar “p005” através do arco da transição seguinte. Esta nova marca inicia o processo de contagem da operação “o016”, que conta o tempo em que a marca permanece no lugar “p005”.

Para disparar a transição “t006”, é necessário haver uma marca no lugar “p005” e a condição “o026” tem de ser verdadeira, isto é, o valor de “Value_1” tem de ser superior ao valor de “Value_2”. Quando a transição “t006” dispara, remove a marca do lugar “p005”, e insere uma marca nova no lugar “p003”, fazendo com que este ative a saída “s30” passados 5 segundos, recomeçando assim o processo.

A geração automática de código é realizada em duas fases:

1ª Fase – Semântica de execução: Utiliza um modelo DS-Pnet para aplicar uma primeira transformação, de forma a gerar um documento XML com as regras necessárias para correr o modelo. Esta fase é comum para todos os geradores de código existentes no ambiente Web.

2ª Fase – sintaxe da linguagem alvo: Pega no XML com a semântica de execução construída, e aplica uma segunda transformação para gerar o código fonte pretendido

É importante referir que antes de aplicar as transformações que executam a geração de código, é necessário executar um passo preliminar, em que se analisam as dependências entre as operações de Dataflow para determinar a sequência correta de execução.

O gerador automático de código apresentado neste trabalho, corresponde à 2ª fase, que utiliza transformações XSLT para produzir código Python. O código Python gerado pelo modelo, está disponível no Anexo II deste relatório.

3.2 Processo de geração automática de código Python

Um modelo criado no ambiente Web do IOPT-Flow é composto por um conjunto de elementos que podem incluir nós de redes de Petri e operações de *Dataflow*s. A cada tipo de elemento correspondem determinadas regras e essas regras constituem a semântica de execução do modelo. Por exemplo, o disparo das transições deve verificar as pré-condições de marcação dos lugares de entrada e dos sinais e eventos associados a essa transição, que inibem o seu disparo. Os nós de *Dataflow*s, aplicam operações matemáticas aos valores de entrada, de forma a calcular os valores de saída (Listagem 1).

Listagem 1:Instrução "if"

3.3 Geração Automática

3.3.1 Módulos Importados

Para que o código Python corra nos dispositivos de *hardware*, os programas gerados importam alguns módulos. Os módulos são arquivos ".py" constituídos por pequenos excertos de código, que podem fazer parte da biblioteca padrão do Python ou podem ser criados pelo utilizador (Figura 29).

```
<xsl:template name="libraries">
    <xsl:text>
from Input_Output_raspberrypi import pinMode,digitalwrite,digitalread
from time import sleep
    </xsl:text>
```

Figura 29: Módulos do Python

"Input_Output_Raspberrypi" – *Script* construído manualmente, que configura e realiza a leitura e escrita dos pinos de i*nput* e *output* da placa *Raspberry Pi* (Anexo I). Este script foi inspirado no ficheiro "model_io.c" construído no gerador de código C, que define todas as operações de entrada e saída do *hardware.* Este ficheiro em código C, implementa as operações de configuração de pinos (pinMode) e leitura e escrita de pinos (*digitalRead*/*digitalWrite*). A atribuição dos pinos de entrada e a saída aos sinais e eventos dos modelos, é feita pelo utilizador, editando as respetivas propriedades no ambiente Web [4].

O módulo importado (Figura 29) implementa 3 funções :

- **def pinMode** – Função que configura o modo de leitura/escrita de um pino;
- **def digitalwrite** – Função que escreve um valor num pino do *Raspberry Pi*;
- **def digitalread** – Função que lê o valor de um pino;

Para além de definir o modo de leitura e escrita dos pinos, a função "pinMode()" foi melhorada para permitir a ativação de resistências internas de *pull-up* e *pull-down* que existem nos pinos GPIO do *Raspberry Pi*.

O *hardware* do *Raspberry Pi* permite configurar cada pino de entrada para *pull-up*, *pull-down* ou nada. A definição do pino de saída com estes atributos é inválida, pois é a lógica do modelo que define o valor de cada saída. A função "pinMode(pino,mode)" recebe como parâmetro de entrada o número e o modo do pino, provenientes da semântica XML.

A escolha das resistências de *pull-up* e *pull-down* é definida na interface Web do editor IOPT-Flow, ao adicionar a *strings* ";PU" e ";PD" ao número do pino. Por exemplo, pino "5;PU". Esta *string* é depois interpretada pela função pinMode() durante a execução do código Python gerado.

"from Time import Sleep" – Importa a função *sleep()* da biblioteca padrão *Time* do Python, usada para definir temporizações no código gerado.

3.3.2 Objetos e Classes

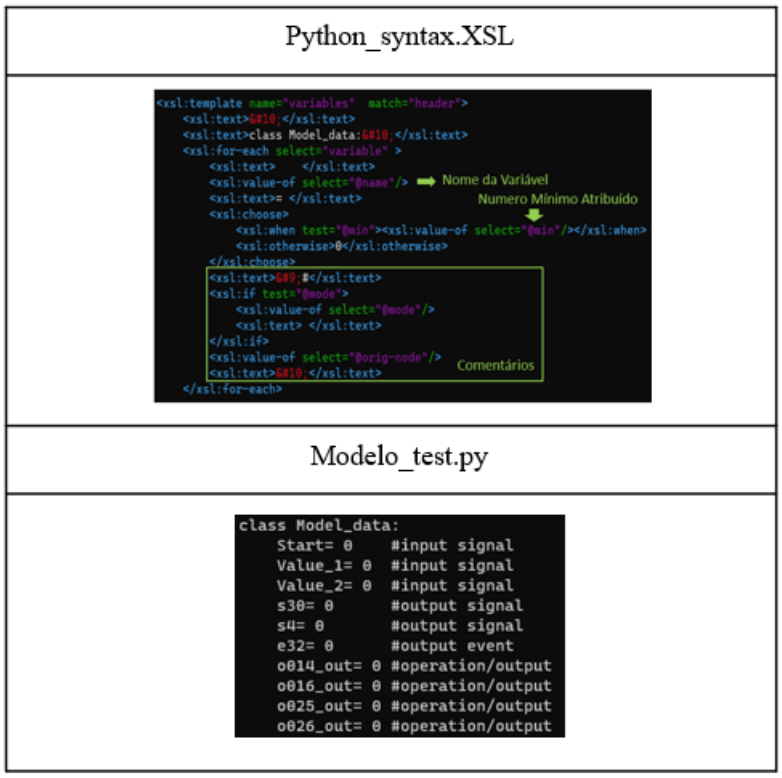
As classes presentes no código gerado, armazenam os dados do modelo DS-Pnet. Ao agrupar os dados por classes, a passagem de parâmetros para as funções ao longo do código fica facilitada, bastando apenas passar um objeto em vez de muitas variáveis (Figura 30).

```
data = Model_data()
marking = Model_marking()
new_marking = Model_new_marking()
avail_marking = Model_avail_marking()
transition_fired = Model_transition_fired()
shift_reg = Model_shift_reg()
```

Figura 30: Objetos das classes

O código gerado contém as seguintes classes:

Class Model_data: Classe que contém as variáveis correspondentes aos sinais e operações de *Dataflow* do modelo. Cada uma das variáveis possui um tipo de dados, que pode ser booleano ou uma gama de inteiros. O utilizador define no editor do modelo os limites para a gama de inteiros de cada sinal e operação. As operações (Figura 28) "o025/o026" e as variáveis/eventos de *input* e *output* estão definidos como booleanos, enquanto as operações "o016" e "o014" possuem gamas de inteiros (Listagem 2).



Listagem 2: Classe Model_data

Pseudocódigo da Transformação XSL:

TEMPLATE 'header' NOME 'variables'
ESCREVE 'class Model_data:'
PARA CADA elemento 'variable' no xml
EXIBE nome da variável + ESCREVE '='
QUANDO existe valor mínimo ('@min')
ESCREVE valor mínimo
CASO CONTRÁRIO
ESCREVE '0'
ESCREVE '#' para iniciar comentário + modo + origem do nó

Class Model_marking: São criados três objetos com a marcação do lugares do modelo: marking, avail_marking e new_marking. O objeto "new_marking" é usado para guardar temporariamente os *tokens* produzidos pelo disparo das transições. Devido a esta razão, os valores deste objeto crescem à medida que as transições disparam. O objeto "avail_marking" guarda a contagem de *tokens* que ainda não foram consumidos pelo disparo de transições e por isso o seu valor vai decrescendo à medida que as transições disparam. O objeto "marking" contém o resultado da marcação dos lugares no fim de cada passo de execução. A transformação "*Python_syntax.XSL*", implementa um ciclo para processar todas as estruturas definidas no ficheiro de semântica "Modelo_test.XML" de forma a poder gerar o ficheiro "Modelo_test.py" (Listagem 3).

modelo_test.XML

```
<struct name="marking">
  <field name="p001" type="range" min="0" max="255" node
  <field name="p003" type="range" min="0" max="255" node
  <field name="p005" type="range" min="0" max="255" node
</struct>
<struct name="new_marking">
  <field name="p001" type="range" min="0" max="255"/>
  <field name="p003" type="range" min="0" max="255"/>
  <field name="p005" type="range" min="0" max="255"/>
</struct>
<struct name="avail_marking">
  <field name="p001" type="range" min="0" max="255"/>
  <field name="p003" type="range" min="0" max="255"/>
  <field name="p005" type="range" min="0" max="255"/>
</struct>
```

Python_syntax.XSL

```
<xsl:for-each select="struct">
    <xsl:text>class Model_</xsl:text>
    <xsl:value-of select="@name"/>
    <xsl:text>:&#10;</xsl:text>
    <xsl:choose>
        <xsl:when test="field">
            <xsl:for-each select="field">
                <xsl:text>    </xsl:text>
                <xsl:value-of select="@name"/>
                <xsl:text>= 0 </xsl:text>
                <xsl:text>&#10;</xsl:text>
            </xsl:for-each>
        </xsl:when>
        <xsl:otherwise>
            <xsl:text>    pass</xsl:text>
        </xsl:otherwise>
    </xsl:choose>
    <xsl:text>&#10;&#10;</xsl:text>
</xsl:for-each>
```

Ciclo de classes

Nome da estrutura

Nome do campo

Modelo_test.py

```
class Model_marking:
    p001= 0
    p003= 0
    p005= 0


class Model_new_marking:
    p001= 0
    p003= 0
    p005= 0


class Model_avail_marking:
    p001= 0
    p003= 0
    p005= 0
```

Listagem 3: Classe: Model_marking; Model_new_marking; Model_avail_marking Model_avail_marking

Pseudocódigo da Transformação XSL:

PARA CADA atributo ‘struct’ no xml

ESCREVE ‘class Model_’ + EXIBE nome da classe + ESCREVE ‘:’

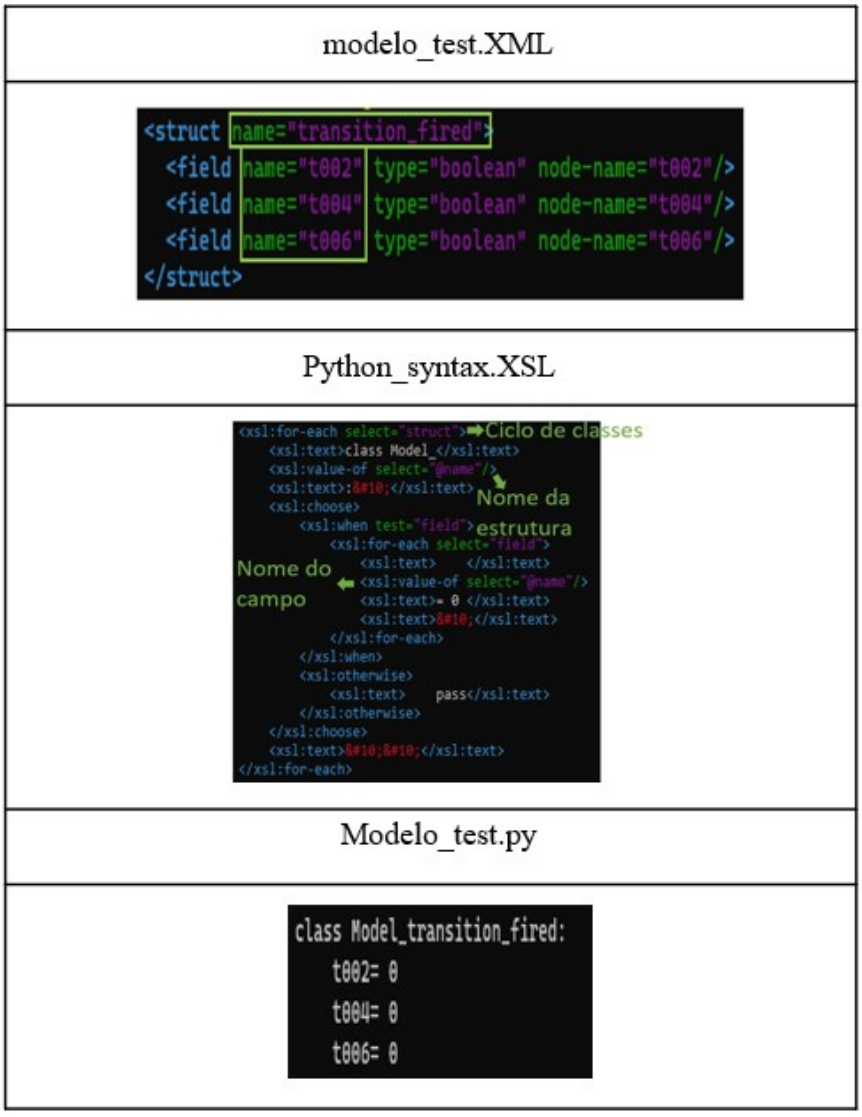
QUANDO existe um elemento ‘field’

EXIBE nome do elemento + ESCREVE ‘= 0’

CASO CONTRÀRIO

ESCREVE ‘pass’

Class_model_transition_fired: Classe usada para guardar informações de tipo booleano, acerca do disparo das transições em cada passo de execução. A Listagem 4 mostra que a transformação implementa um ciclo (Python_syntax.XSL), utilizando os parâmetros disponíveis na semântica, para criar variáveis correspondentes a todas as transições.



Listagem 4: Classe Model_transition_fired

Pseudocódigo da Transformação XSL:

PARA CADA atributo ‘struct’ no xml

ESCREVE ‘class Model_’ + EXIBE nome da classe + ESCREVE ‘:’

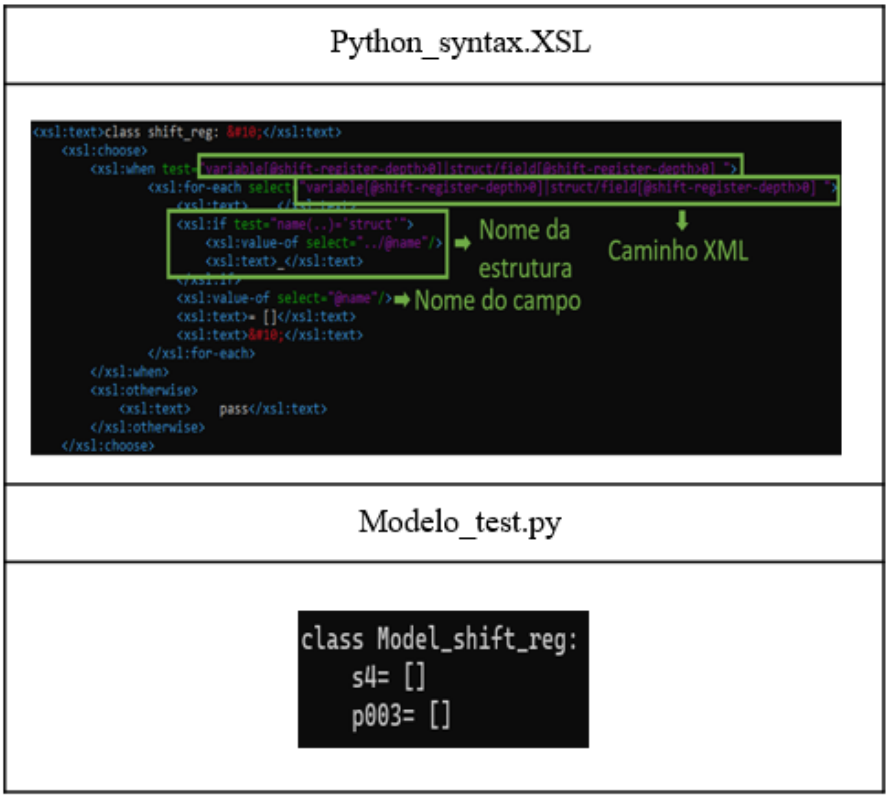
QUANDO existe um elemento ‘field’

EXIBE nome do elemento + ESCREVE ‘= 0’

CASO CONTRÀRIO

ESCREVE ‘pass’

Class shift_reg: Classe que guarda o histórico de valores dos sinais dos passos de execução anteriores do modelo. Para cada valor é criado um registo de deslocamento que é atualizado a cada passo de execução. Esta classe é necessária para permitir que as expressões matemáticas possam aceder a valores de sinais em passos de execução anteriores. Só são guardados os valores efetivamente usados pelo modelo (Listagem 5).



Listagem 5: Classe Shift_reg

Pseudocódigo da Transformação XSL:

ESCREVE ‘class Shift_reg:’

QUANDO o atributo ‘Shift-register-depth’ do xml for >0

EXIBE nome da variável + ESCREVE ‘= []’

CASO CONTRÀRIO

ESCREVE ‘pass’

3.3.3 Funções

−**def main ()**: Função principal do programa, que estabelece a ordem de execução das restantes funções definidas em código Python (Figura 31).

```
def main():
    init()
    setup_io()
    while( 1 ):
        readInputs()
        executionStep()
        writeOutputs()
        delayPause()
```

Figura 31: Main

A função inicializa as variáveis e define a configuração dos pinos. A partir daí, entra num ciclo infinito e faz a leitura dos pinos de entrada, calcula um novo passo de execução e aplica os resultados nos pinos de saída (Listagem 6).

Python_syntax.XSL

```
<xsl:param name="generate-main" select="1" />
<xsl:template match="loop">
    <xsl:text>    setup_io()&#10;</xsl:text>
    <xsl:text>    while( 1 ): &#10;</xsl:text>
    <xsl:apply-templates select="*">
        <xsl:with-param name="level" select="2" />
    </xsl:apply-templates>
    <xsl:text>    &#10;</xsl:text>
</xsl:template>
<xsl:template match="call-procedure">
    <xsl:param name="level" select="1"/>
    <xsl:if test="$level = 2">
        <xsl:text>    </xsl:text>
    </xsl:if>
    <xsl:text>    </xsl:text>
    <xsl:value-of select="@name" />
    <xsl:text>(</xsl:text>
    <xsl:for-each select="@*[name(.) != 'name']">
        <xsl:choose>
            <xsl:when test="name(.) = 'variable' or string(number(.)) != 'NaN'">
                <xsl:value-of select="." />
            </xsl:when>
            <xsl:otherwise>
                <xsl:text>"</xsl:text>
                <xsl:value-of select="."/>
                <xsl:text>"</xsl:text>
            </xsl:otherwise>
        </xsl:choose>
        <xsl:if test="position() != last()">
            <xsl:text>, </xsl:text>
        </xsl:if>
    </xsl:for-each>
    <xsl:text>)&#10;</xsl:text>
</xsl:template>
```

Definição do parâmetro "main"

Função de configuração de pinos

Ciclo Infinito

Colocação das funções dentro do ciclo while

Indentação das funções

Parâmetros de entrada das funções

modelo_test.XML

```
<procedure name="main">
  <call-procedure name="init"/>
  <loop>
    <call-procedure name="readInputs"/>
    <call-procedure name="executionStep"/>
    <call-procedure name="writeOutputs"/>
    <call-procedure name="delayPause"/>
  </loop>
</procedure>
```

Listagem 6: Função main

Pseudocódigo da Transformação XSL:

TEMPLATE ‘loop’

ESCREVE função ‘setup_io’

ESCREVE ciclo ‘while (1):’

SELECIONA todos os elementos existentes em ‘loop’ no xml

TEMPLATE ‘call-procedure’

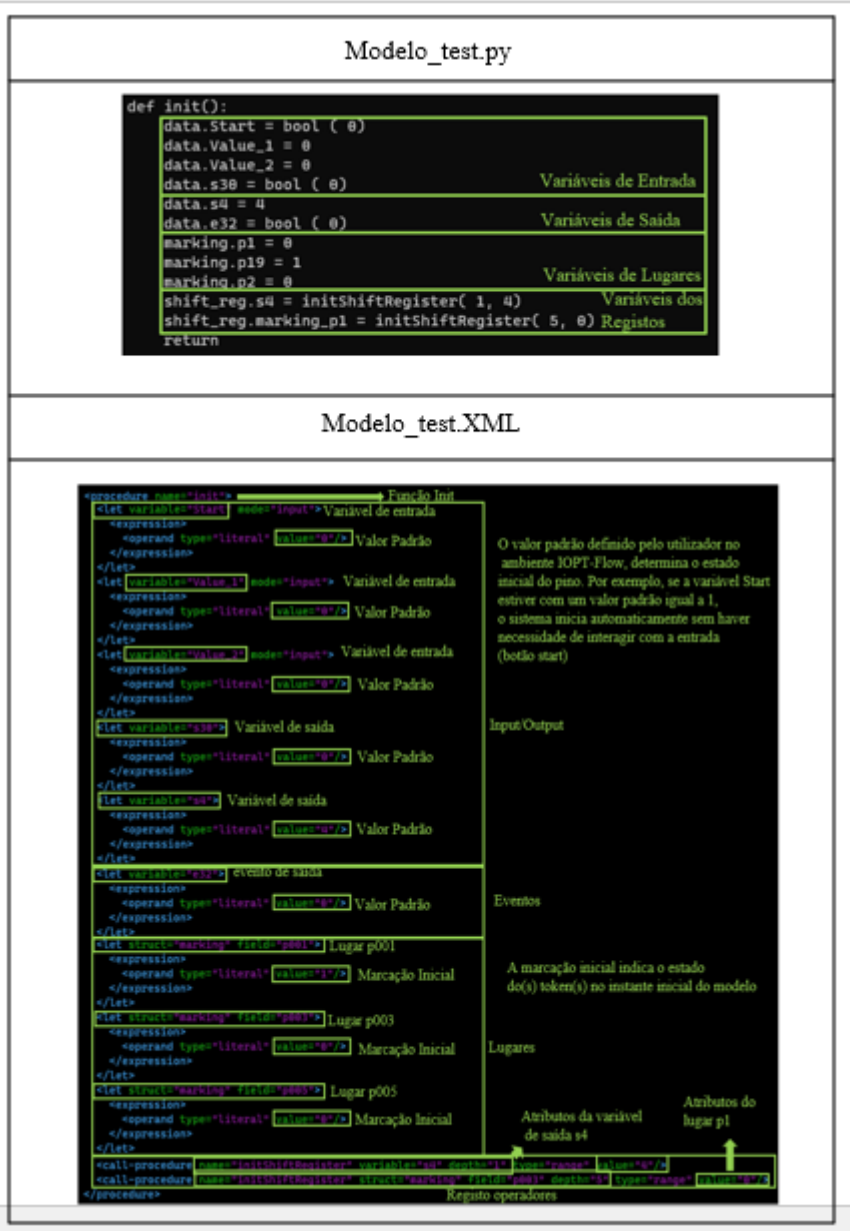
APLICA a indentação às funções

CHAMA cada função

ESCREVE ‘(‘ + parâmetros de entrada de cada função + ‘)’

−**def init ()**: Função que inicializa as variáveis. Esta função contém uma lista de instruções, cujos valores iniciais são definidos no modelo da Figura 28. A associação entre as expressões do ficheiro XSL (Listagem 8) com as expressões do ficheiro XML e o seu resultado em código Python (listagem 7). O *template* “procedure” gera a estrutura da função de inicialização das variáveis (“def init”) e define os seus parâmetros de entrada. O template “call-procedure” declara as variáveis de registo das operações, que neste caso são “Shift_reg.s4” e “Shift_reg.marking_p1”. O *template* ”let” inicializa as restantes variáveis e eventos de entrada, saída e marcação de lugares.

Listagem 7:Função init formato XSL



Listagem 8: Função init

Pseudocódigo da Transformação XSL:

TEMPLATE procedure

ESCREVE ‘def’ + EXIBE nome do elemento + ESCREVE ‘(‘ + DEFINE parâmetros de entrada + ESCREVE ‘)’

SELECIONA todos os elementos existentes no procedure

ESCREVE ‘return’ para todas as funções excepto a função main

TEMPLATE ‘call-procedure’

ESCREVE ‘shift_reg.”

QUANDO existe atributo de lugar (‘struct’)

EXIBE nome da estrutura (‘struct’) + ESCREVE ‘_’ + EXIBE nome do lugar (‘field’)

CASO CONTRÁRIO

EXIBE nome da variável (‘variable’)

ESCREVE ‘ = initShiftRegister(‘ + EXIBE valor anterior + ESCREVE ‘,’ + MOSTRA valor pre-definido no modelo + ESCREVE ‘)’

TEMPLATE ’let’

DECLARAÇÃO da variável ‘dest’

QUANDO existe atributo de lugar (‘struct’)

EXIBE nome da estrutura de marcação (‘struct’) + ESCREVE ‘.’ + EXIBE nome do lugar (‘field’)

CASO CONTRÁRIO

ESCREVE ‘data.’ + EXIBE nome da variável (‘variable’)

VARIÁVEL do elemento ‘dest’ + ESCREVE ‘=’

QUANDO tipo de variável for booleano

ESCREVE ‘bool (‘ + EXIBE expressão valor pré-definido no modelo + ESCREVE ‘)’

CASO CONTRÁRIO

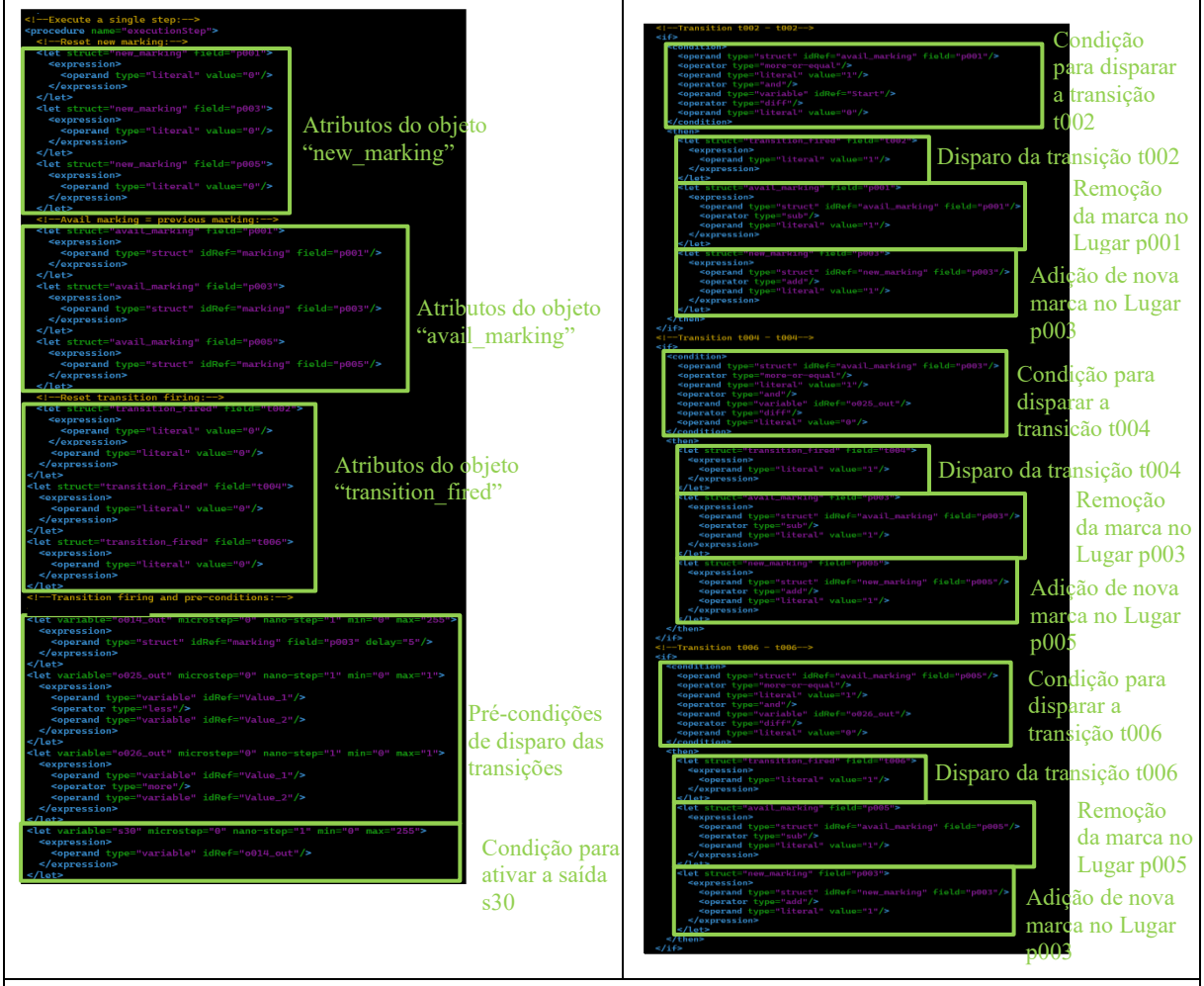
EXIBE expressão do valor pré-definido no modelo

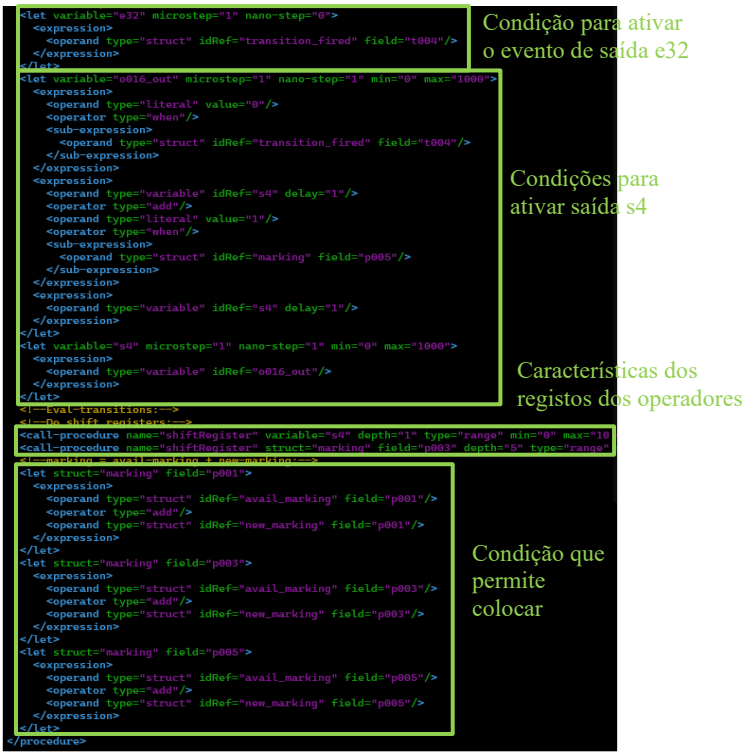
−def executionStep() : Função que desempenha os passos de execução do modelo da Figura 28. Esta função é composta por uma sequência de instruções que definem as ações do modelo. A Figura 32 apresenta as tarefas do modelo construído, ou seja, todas as ações definidas no modelo são executadas por esta função tais como: as condições de disparo de cada transição, os atributos de cada uma das operações, as condições de ativação dos eventos e sinais de saída bem como todo o processo de transferência de marcas entre lugares.

A Listagem 9 contém uma representação do funcionamento do modelo da Figura 27. Esta representação XML contém todas as instruções necessárias para que seja possível construir o código Python, utilizando a linguagem de transformação apresentada na Listagem 10 (XSLT). O código da listagem 10 (XSL) vai buscar os atributos ao troço da semântica de execução apresentada na listagem 9 e gera automaticamente a função exibida na Figura 32. Este código em XSL é composto por um *template* “let”, que realiza a construção das variáveis apresentadas na função da Figura 32.

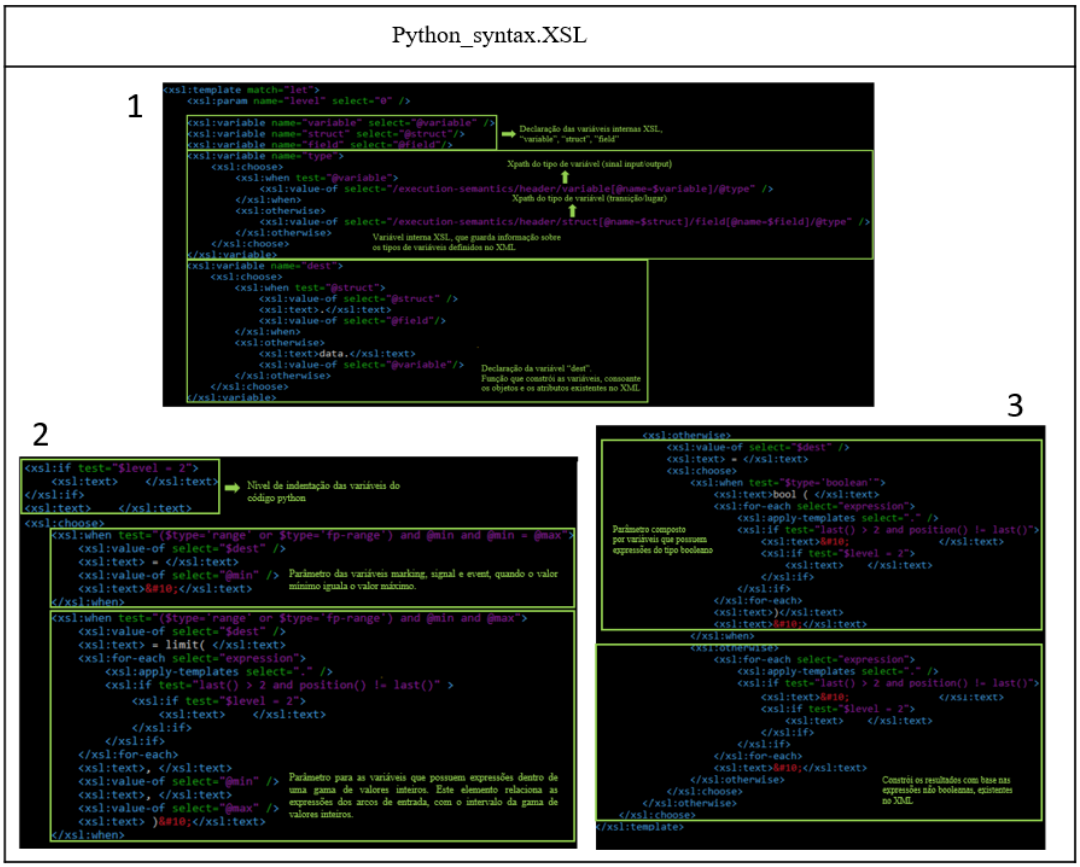
Figura 32: Função ExecutionStep

modelo_test.XML





Listagem 9:Modelo_test.XML - Função executionStep



Listagem 10: Python_syntax.XSL - Função executionStep

Pseudocódigo da Transformação XSL:

TEMPLATE ‘let’

DECLARAÇÃO variáveis internas (‘variable’,’struct’,’field’)

DECLARAÇÃO variável ‘field’

QUANDO existe elemento ‘variable’ no xml

DEFINE o tipo de variável como input ou output

CASO CONTRÁRIO

DEFINE o tipo de variável como um lugar ou marcação

DECLARAÇÃO da variável ‘dest’

QUANDO existe atributo de lugar (‘struct’)

EXIBE nome da estrutura de marcação (‘struct’) + ESCREVE ‘.’ + EXIBE nome do lugar (‘field’)

CASO CONTRÁRIO

ESCREVE ‘data.’ + EXIBE nome da variável (‘variable’)

QUANDO tipo de variável é igual a ‘range’ ou ‘fp-range’ e o valor min = valor max

EXIBE nome da variável + ESCREVE ‘=’ + EXIBE valor min

QUANDO tipo de varável é igual a ‘range’ ou ‘fp-range’ e tem valor min e valor max

EXIBE nome da variável + ESCREVE ‘= limit(‘

EXIBE variável associada ao objeto de entrada + ESCREVE ‘,’ + EXIBE valor mínimo do intervalo + ESCREVE ‘,’ + EXIBE valor máximo do intervalo + ESCREVE ‘)’

CASO CONTRÁRIO

EXIBE nome da variável + ESCREVE ‘=’

QUANDO tipo de variável for booleano

ESCREVE ‘bool (‘ + EXIBE expressão valor pré-definido no modelo + ESCREVE +)’

CASO CONTRÁRIO

EXIBE expressão do valor pré-definido no modelo

−**def setup_io()**: Função que define a configuração dos Pinos, de acordo com os atributos de cada sinal e evento de entrada/saída. Esta função recebe os parâmetros dos pinos do *Raspberry Pi*, que são usados para invocar a função “pinMode()”, que é implementada no módulo “*Input_Output_Raspberrypi.py*”(Anexo I). Os pinos definidos com modo 1, são considerados como *inputs*, enquanto que os pinos definidos por 0 são considerados como *outputs* (Listagem 11).

modelo_test.XML

```
<!--Signals:-->
<variable name="Start" orig-node="signal" mode="input" type="boolean" def_value="0" io_pin="10;PD"/>
<variable name="Value_1" orig-node="signal" mode="input" type="range" min="0" max="100" def_value="0" io_pin="11;PD"/>
<variable name="Value_2" orig-node="signal" mode="input" type="range" min="0" max="100" def_value="0" io_pin="12;PD"/>
<variable name="s30" orig-node="signal" mode="output" type="range" min="0" max="255" def_value="0" io_pin="2"/>
<variable name="s4" orig-node="signal" mode="output" type="range" min="0" max="1000" shift-register-depth="1" def_value="4" io_pin="4"/>
<!--Events:-->
<variable name="e32" orig-node="event" mode="output" type="boolean" io_pin="3"/>
```

Atributos dos pinos

Python_syntax.XSL

```
<xsl:template name="setup_pinmode">
    <xsl:text>&#10;def setup_io():&#10;</xsl:text>
    <xsl:choose>
        <xsl:when test="header/variable[@io_pin]">
            <xsl:for-each select="header/variable[@io_pin]">
                <xsl:text>    pinMode("</xsl:text>
                <xsl:value-of select="@io_pin" />
                <xsl:text>",</xsl:text>
                <xsl:choose>
                    <xsl:when test="@mode='input'">1</xsl:when>
                    <xsl:when test="@mode='output'">0</xsl:when>
                </xsl:choose>
                <xsl:text>)</xsl:text>
                <xsl:text>&#9;#Pino: </xsl:text>
                <xsl:if test="@mode">
                    <xsl:value-of select="@mode"/>
                    <xsl:text> </xsl:text>
                </xsl:if>
                <xsl:text>&#10;</xsl:text>
            </xsl:for-each>
            <xsl:text>&#10;</xsl:text>
        </xsl:when>
        <xsl:otherwise>
            <xsl:text>    return</xsl:text>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Configuração dos Pinos

Comentário que indica se o pino é de entrada ou de saída

Modelo_test.py

```
def setup_io():
    pinMode("10;PD",1)  #Pino: input
    pinMode("11;PD",1)  #Pino: input
    pinMode("12;PD",1)  #Pino: input
    pinMode("2",0)      #Pino: output
    pinMode("4",0)      #Pino: output
    pinMode("3",0)      #Pino: output
```

Listagem 11: Função setup_io

Pseudocódigo da Transformação XSL:

TEMPLATE ‘setup_pinmode’

ESCREVE ‘def setup_io()’

QUANDO existe atributo ‘io_pin’ no xml

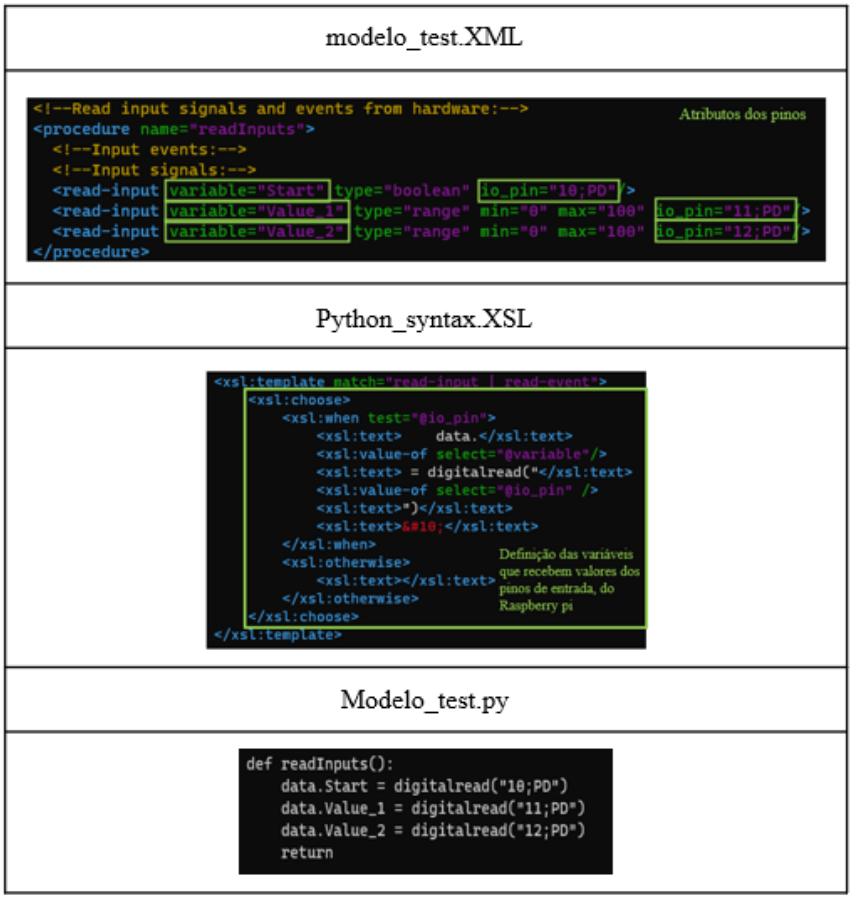
ESCREVE ‘pinmode(“ ‘ + número e característica do pino + modo + ‘)’

ESCREVE ‘#Pino: ’ + nome correspondente ao modo da identificação associada (1- Input, 0 – Output)

CASO CONTRÁRIO

ESCREVE ‘return’

–**def readInputs()**: Função que lê os pinos de entrada através da função “digitalRead()” definida no módulo “*Input_Output_Raspberrypi.py*” (Anexo I). Esta função é chamada sempre que se inicia um novo passo de execução (Listagem 12).



Listagem 12: Função readInputs

Pseudocódigo da Transformação XSL:

TEMPLATE ‘read inputs’

QUANDO existe o atributo ‘io_pin’ no xml

ESCREVE ‘data’ + nome do sinal de entrada definido no modelo

ESCREVE ‘= digitalread (‘ + número e característica do pino + ‘)’

CASO CONTRÁRIO

NÂO ESCREVE NADA

–**def writeOutputs()**: Função que escreve valores de saída nos pinos do *Raspberry Pi* através da função “digitalWrite()” definida no módulo “*Input_Output_Raspberrypi.py*” (Anexo I). Esta função é chamada no final de cada passo de execução (Listagem 13).

modelo_test.XML

```
<!--Write output signals and events to hardware:-->
<procedure name="writeOutputs">
  <!--Output signals:-->
  <write-output variable="s30" type="range" min="0" max="255" io_pin="2"/>
  <write-output variable="s4" type="range" min="0" max="1000" io_pin="4"/>
  <!--Output events:-->
  <write-event variable="e32" io_pin="3"/>
</procedure>
```

Atributos dos pinos

Python_syntax.XSL

```
<xsl:template match="write-output | write-event">
    <xsl:choose>
        <xsl:when test="@io_pin">
            <xsl:text>    digitalwrite("</xsl:text>
            <xsl:value-of select="@io_pin" />
            <xsl:text>", data.</xsl:text>
            <xsl:value-of select="@variable"/>
            <xsl:text>)</xsl:text>
            <xsl:text>&#10;</xsl:text>
        </xsl:when>
        <xsl:otherwise>
            <xsl:text></xsl:text>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Instruções que escrevem valores nos pinos de saída do Raspberry pi

Modelo_test.py

```
def writeOutputs():
    digitalwrite("2", data.s30)
    digitalwrite("4", data.s4)
    digitalwrite("3", data.e32)
    return
```

Listagem 13: Função writeOutput

Pseudocódigo da Transformação XSL:

TEMPLATE ‘write output’

QUANDO existe o atributo ‘io_pin’ no xml

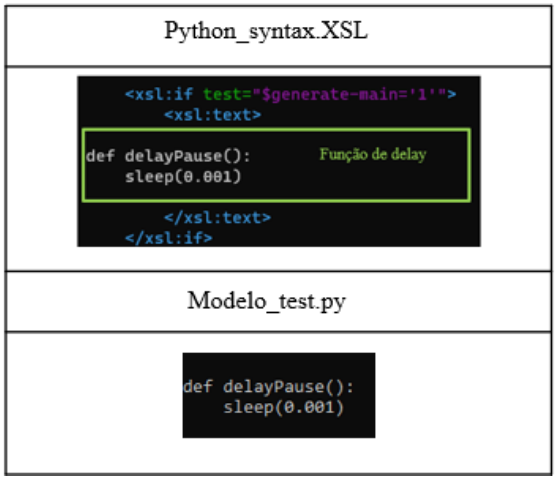
ESCREVE ‘digitalwrite (“ ‘ + número do pino de saida

ESCREVE ‘ ”, data. ’ + nome do sinal de saída + ‘)’

CASO CONTRÁRIO

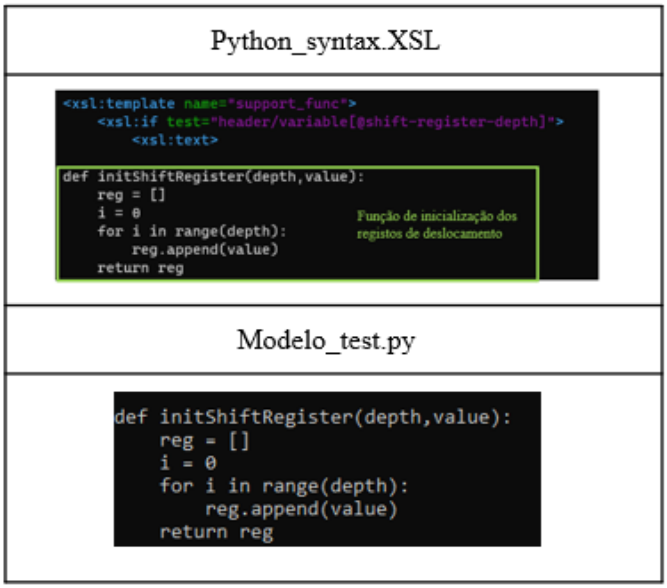
NÂO ESCREVE NADA

– **def delayPause()**: Aplica uma pausa entre passos de execução. O *Delay* é programado para 1ms por omissão, mas esse tempo pode ser alterado. O valor de 1ms é semelhante ao utilizado nos ciclos de execução de autómatos/PLC (Listagem 14).



Listagem 14: Função delayPause

–**def initShiftRegister()**: Função escrita manualmente, adicionada de forma automática pela transformação XSL, que recebe parâmetros provenientes da função "init" para inicializar os registos de deslocamento, de acordo com a sua dimensão e o valor inicial (Listagem 15).



Listagem 15: Função initShiftRegister

–**def shiftRegister()**: Função que implementa o deslocamento dos valores contidos num registo de deslocamento. Remove o último elemento da variável "reg" e retorna o último elemento removido. Esse valor removido é trocado pelo valor do parâmetro "value", que será inserido na posição zero da variável "reg" (Listagem 16).

Python_syntax.XSL

```
def shiftRegister(reg,value):
    reg.pop()
    reg.insert(0, value)
```

Função de deslocamento de valores provenientes do registo reg

```
        </xsl:text>
    </xsl:if>
```

Modelo_test.py

```
def shiftRegister(reg,value):
    reg.pop()
    reg.insert(0, value)
```

Listagem 16: Função ShiftRegister

–**def limit ()**: Função que limita um valor a um intervalo definido por dois parâmetros "min" e "max". Esta função é usada para garantir que o resultado das expressões matemáticas não excede a gama de valores permitidos para o sinal onde será guardado no final (Listagem 17).

Python_syntax.XSL

```
    <xsl:text>
def limit(value,min,max):
    if (value &lt; min):
        return  min
    elif (value &gt; max):
        return max
    else:
        return value
    </xsl:text>
```

Modelo_test.py

```
def limit(value,min,max):
    if (value < min):
        return  min
    elif (value > max):
        return max
    else:
        return value
```

Listagem 17: Função Limit

A tabela 12 apresenta as *Tags* principais utilizadas na semântica de execução XML:

<execution-semantics>	Nome do modelo
<header>	Declaração das variáveis
<variable>	Declaração das variáveis de entrada/saída
<struct>	Declaração das variáveis estruturadas ou classes
<code>	Contém todo o código gerado
<procedure>	Declara uma função/procedimento
<let>	Atribui valores a uma variável
<operand>	Valores/Variáveis utilizados nas expressões matemáticas
<operator>	Operadores utilizados nas expressões
<expression>	Expressões matemáticas
<read-inputs>	Lê sinais e eventos a partir do Hardware
<write-outputs>	Escreve sinais e eventos a partir do Hardware

<if>	Execução condicional
<condition>	Execução condicional
<then>	Execução condicional
<else>	Execução condicional
<call-procedure>	Executa um procedimento/função
<loop>	Ciclo de elementos

Tabela 12: Tabela de Tags da semântica de execução

3.4 Acesso aos *Scripts* no ambiente do *Raspberry Pi*

Após determinar a sequência de execução das operações de Dataflow, o processo de geração de código começa pela aplicação de uma transformação XSL ("exec_sem.xsl") sobre o documento XML do modelo, de forma a gerar outro ficheiro XML com a semântica de execução. De seguida, outra transformação "python_syntax.xsl", converte o resultado da primeira transformação em código Python.

A Figura 33 apresenta uma sessão de trabalho remota, usando o protocolo ssh, para aceder à placa *Raspberry Pi* a partir de um computador pessoal. O utilizador necessita conhecer o endereço IP e os dados de autenticação (*user*/*password*) para poder entrar. O upload dos ficheiros de código fonte, produzido pelo gerador de código Python pode ser realizado através do protocolo ssh, PenDrive USB ou cartão SD, diretamente na placa Raspberry pi. Observando a Figura 33, é possível encontrar os documentos dentro da pasta "/Tese/modelo test". Através do protocolo ssh, também foi possível inserir os scripts escritos à mão criados, em XSL e código Python, para dentro do servidor do ambiente Web IOPT-Flow.

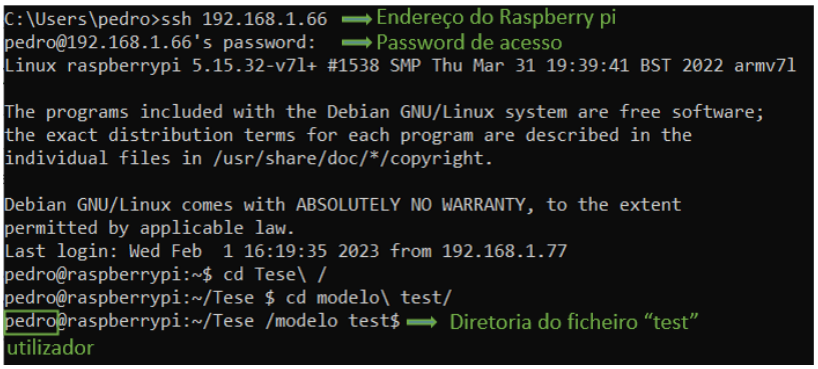


Figura 33: Localização do modelo "test", dentro do Raspberry Pi

Capítulo 4. Modelos de Validação

O presente capítulo apresenta os dois modelos de validação criados, que demonstram a viabilidade da ferramenta desenvolvida. O primeiro modelo usa um teclado numérico universal para realizar funções de comando de porta. O segundo modelo utiliza um braço robótico que pode ser controlado manualmente ou de forma automática.

4.1 Alarme de porta

Este modelo de validação serviu de protótipo para validar o desempenho do gerador de código automático, baseado num controlador para ler um teclado numérico universal. Este teclado é composto por 3 colunas (saídas) e 4 linhas (entradas), ao todo emprega 7 pinos de ligação. Por cada tecla premida, ativa-se a ligação entre um pino de uma linha e um pino de uma coluna. O funcionamento do teclado consiste na junção das linhas com as colunas. Cada vez que uma das saídas é ativada, faz-se uma leitura de cada uma das entradas de forma a verificar se alguma tecla dessa coluna foi premida. Se alguma das entradas estiver a 1, identifica-se a origem da tecla correspondente (Figura 34).

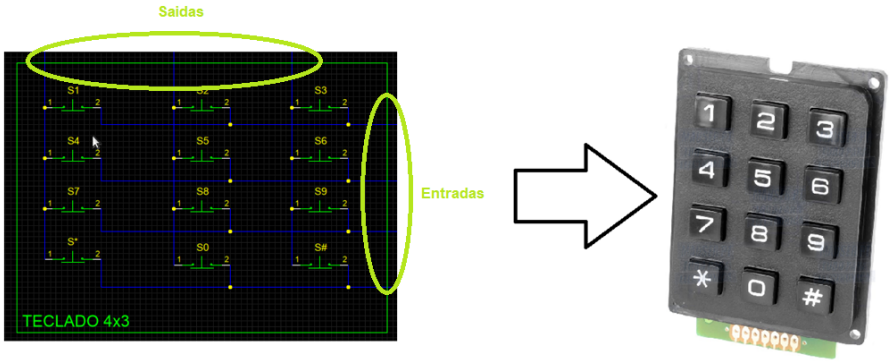


Figura 34: Esquema Teclado Universal

A Figura 35 mostra o esquema de ligações do modelo. Neste esquema é possível ver que os pinos das 4 entradas são ligados diretamente ao *Raspberry pi*, aproveitando os *pull-downs* internos, permitindo que os pinos estejam a zero se nenhum dos 4 botões for pressionado. As 3 saídas para as colunas, são conectadas aos díodos retificadores, que se encontram ligados entre as saídas do *Raspberry Pi* e o teclado. No esquema apresentado na Figura 35 existe também um *push-button* que simula uma intrusão no sistema, um led vermelho que serve de alarme sempre que o botão de intrusão é pressionado e um led verde que acende sempre que a porta se encontra aberta.

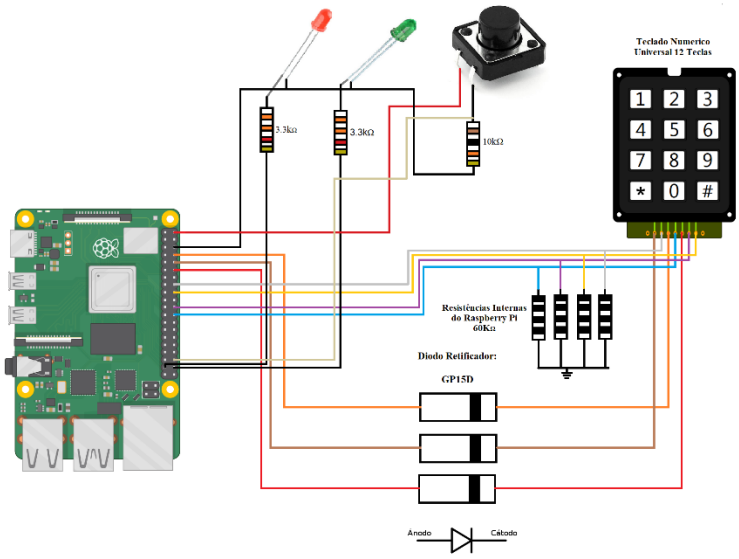


Figura 35: Esquema Elétrico Alarme de Porta

A utilização do díodo retificador na montagem da Figura 35 é bastante importante, pois ajuda a evitar danos nos pinos de saída, evitando causar curto-circuitos entre saídas com valores diferentes (0 e 1). Cada vez que se põe uma saída do *Raspberry Pi* a 1, o díodo recebe 3.3v no ânodo (terminal positivo) e passa a haver corrente a circular na coluna indicada. A tensão que existe na entrada do teclado, será o valor da tensão do pino do *Raspberry Pi*, menos a queda de tensão aos terminais do díodo (que pode ser de 0,5v/0,7v/1v). Se a saída do *Raspberry Pi* passar para 0, a coluna indicada deixa de ser alimentada e o díodo entra em alta impedância. Pode-se dizer que o díodo oferece duas alternativas:

– Ligar a coluna a alta impedância (0V)

– Ligar a coluna a 3.3v (Pino do *Raspberry Pi* ativo)

O objetivo deste modelo de validação é simular a abertura de uma porta, utilizando uma combinação inserida através do teclado numérico. Para além disso, este modelo também tem a capacidade de poder alterar a senha de acesso para um código escolhido pelo utilizador e ainda tem a possibilidade de simular uma intrusão no sistema.

O modelo de alarme de porta divide-se em 3 submodelos:

– **Modelo de comando do teclado universal**: Modelo que lê o teclado e regista o último valor premido pelo utilizador. O modelo da Figura 36 percorre ciclicamente todas as colunas do teclado, ativando uma coluna de cada vez, para verificar se alguma tecla está premida. Quando isso acontece regista o valor da tecla correspondente

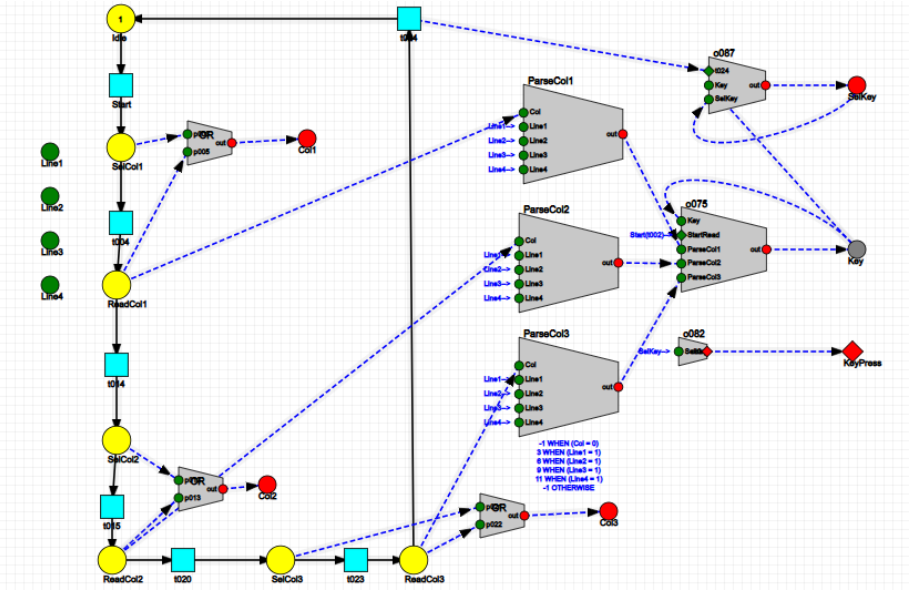


Figura 36: Modelo IOPT-Flow: Teclado Universal

Para ativar as colunas do teclado numérico basta haver *tokens* nos lugares "*SelCol1,2,3*." ou "*ReadCol1,2,3*"(Figura 36). As saídas "Col1", "Col2" e "Col3" são do tipo booleano e representam a ativação dos pinos de saída do *Raspberry Pi* que correspondem às colunas do teclado universal (Figura 34). Quando cada uma das colunas se encontra no estado ativo, alimenta o polo positivo do díodo com 3.3v e o díodo fica pronto para conduzir.

O sinal interno "*key*" indica o número da tecla que está a ser pressionada. Esta variável possui uma gama de valores inteiros que vai de -1 até 11. Cada um destes valores corresponde ao número de caracteres do teclado universal, onde:

- -1 – Nenhuma tecla carregada;
- 0–9 Dígitos;
- 10 – Corresponde a *;
- 11 – Corresponde a #;

As operações "ParseCol1", "ParseCol2, "ParseCol3" possuem os valores das teclas de cada coluna do teclado. Operação "o075" recebe e envia os números correspondentes de cada linha selecionada, às restantes operações do modelo. Operação "*KeyPress*" produz um evento que fica ativo sempre que uma tecla nova é pressionada. A saída "*SelKey*" recebe o número da tecla proveniente do sinal interno "*key*", que é atualizado no fim do ciclo.

– **Modelo que lê uma sequência de caracteres**: Este modelo (Figura 37) lê e memoriza uma sequência de 17 caracteres (teclas) seguidos. Esta sequência de 17 dígitos tem duas funções, para além de guardar os valores associados à senha de acesso a abertura da porta, também armazena os caracteres que permitem fazer a alteração do código de acesso.

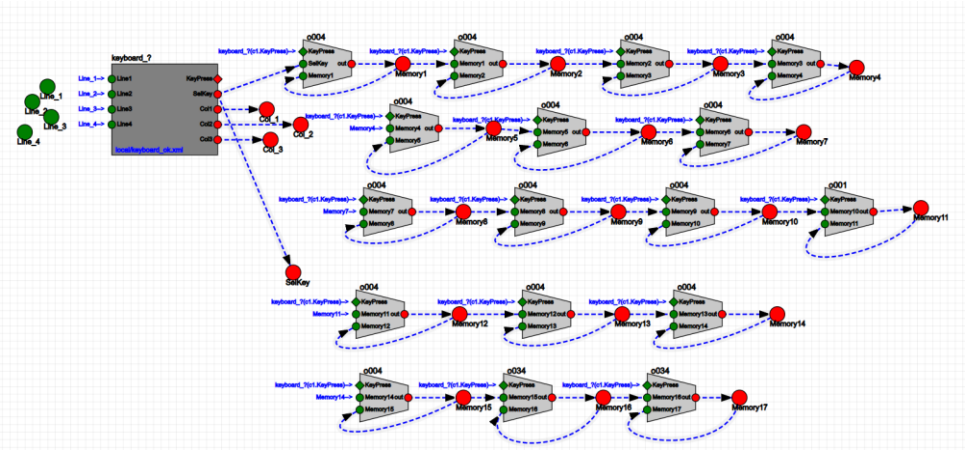


Figura 37: Modelo IOPT-Flow: Sequência de Caracteres

Para descrever o funcionamento do modelo da Figura 37, irá ser usado como caso-de-uso a aplicação da senha para abrir a porta. A senha é um código composto por 4 caracteres numéricos seguidos pela tecla #, que valida os caracteres colocados anteriormente. O modelo da Figura 39, utiliza uma combinação inicial pré-definida (1234#) (Tabela 13) que simula o funcionamento do sistema. A sequência de teclas inseridas funciona como um registo de deslocamento. Sempre que o utilizador insere uma nova tecla, as anteriores são deslocadas uma posição para a frente. O modelo da Figura 37 verifica a posição especifica das teclas na sequência. Quando não existe eventos de "*keypress*", as saídas das memórias mantêm os valores antigos, permanecendo com os valores das últimas teclas pressionadas. Isto faz com que a análise seja vista de trás para afrente, a posição do primeiro valor colocado será a última na memória, enquanto que a posição do último valor será a primeira (Tabela 13).

Posição na memória	Memória 1	Memória 2	Memória 3	Memória 4	Memória 5
Palavra Pass	11 (#)	4	3	2	1
Ordem da colocação de valores	5°	4°	3°	2°	1°

Tabela 13: Aplicação da Senha

– **Modelo de Alarme de Porta**: As funções modelo de alarme de porta consistem, tal como mencionado anteriormente, no comando de um teclado universal para controlar a abertura de uma porta, na alteração de código se o utilizador preferir e simulação de uma intrusão. Este modelo pode ser aplicado em diversas estruturas do dia-a-dia, tal como casas, prédios, garagens, portões, etc. O mecanismo também pode ser utilizado em fechaduras eletrónicas sem chave, fazendo com que o pin seja a "chave" de acesso à divisão (Figura 38).

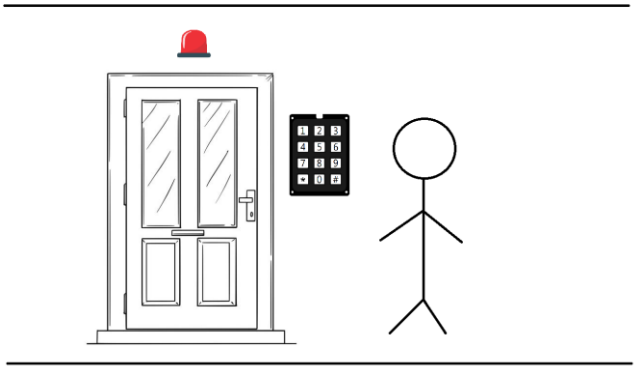


Figura 38: Exemplo de aplicação do modelo Alarme de Porta

O modelo principal do alarme de porta é formado por um componente interno que contém os dois modelos mencionados anteriormente, o comando do teclado universal (Figura 36) e a memória da sequência de caracteres (Figura 37). Este componente é composto por 4 entradas e 21 saídas. As entradas descrevem as linhas do teclado. As saídas memorizam os valores provenientes de "*SelKey*" (17), controlam as colunas do teclado (3) e indicam o último valor selecionado na máquina de estados da Figura 36 (1). A Figura 39 mostra os 3 sistemas que descrevem o funcionamento final do modelo.

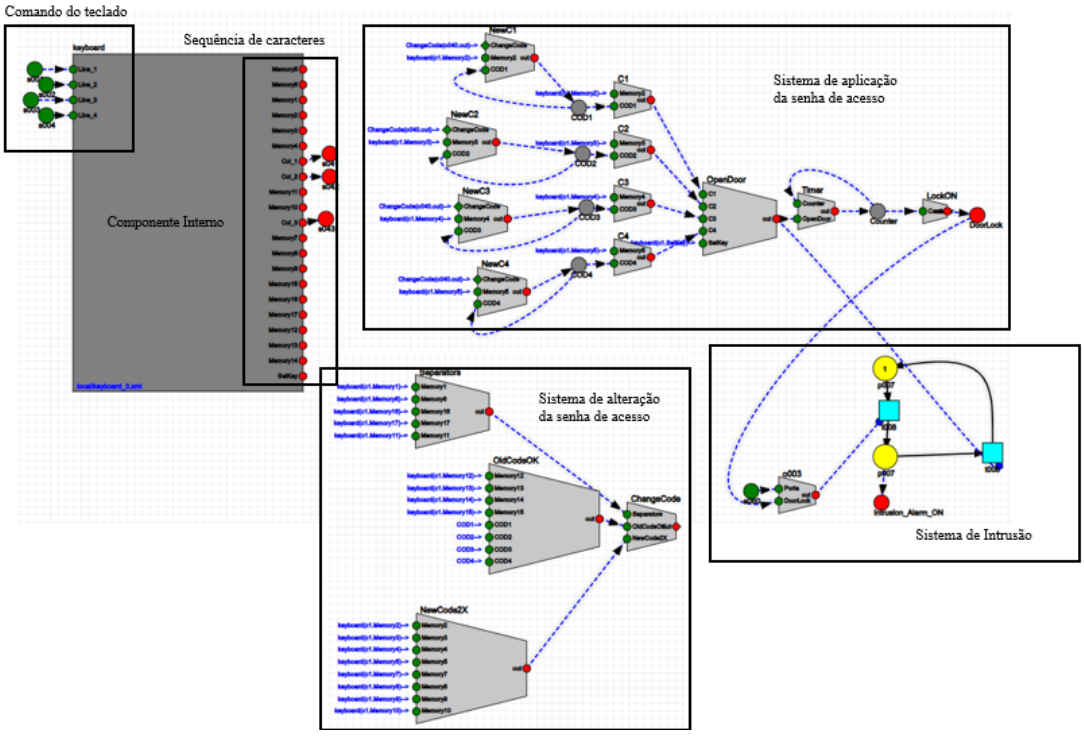


Figura 39: Modelo Alarme de Porta

Descrição do modelo da Figura 39 :

- **Abertura de porta usando senha**: tal como mencionado no tópico "**Modelo que lê uma sequência de caracteres"**, é um processo que compara as 5 variáveis ("COD1", "COD2", "COD3", "COD4","*SelKey*") com as 5 primeiras memórias. Se os 5 valores forem idênticos, o trinco da porta fica ativo por um tempo limitado ("*DoorLock*") e é iniciada uma contagem decrescente ("Timer") para trancar a fechadura automaticamente (200ms).

Para voltar a abrir a porta é necessário inserir a senha correta de novo. O modelo possui uma sequência pré-definida para ativar a senha, cujos caracteres são compostos por 1,2,3,4,# (Tabela 13). Esta combinação permite ensaiar o modelo, mas se for necessário, os caracteres numéricos podem ser alterados de acordo com a preferência do utilizador (Figura 40).

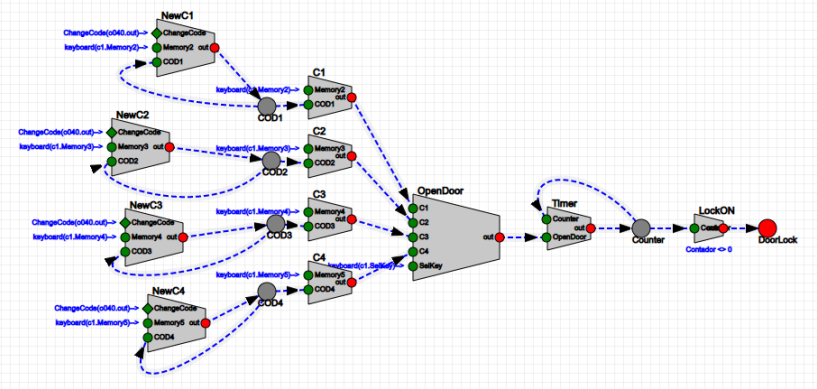


Figura 40: Troço de abertura de porta

- **Sistema de Intrusão**: Para simular uma intrusão, usou-se o *input* "s060". Esta entrada é um sensor (Ex: botão) que aciona o alarme e indica se a fechadura foi aberta sem que o código tenha sido inserido (segurança comprometida). Para desativar o alarme, basta introduzir a combinação correta do código atual e pressionar # (tecla 11). A máquina de estado memoriza o estado do alarme até que o utilizador insira de novo o código (Figura 41).

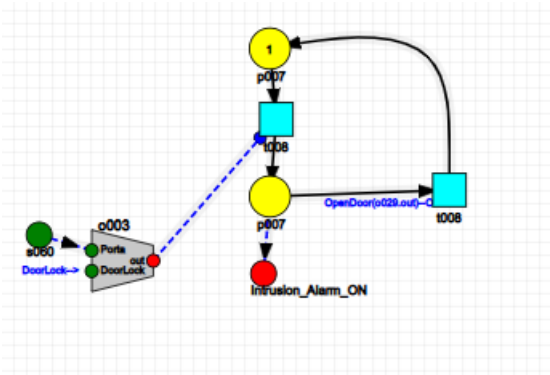


Figura 41: Troço de intrusão

- **Sistema de alteração da senha**: Para alterar a senha, é necessário inserir uma combinação composta pelo código antigo (1234), em conjunto com o código novo (5678) e a sua confirmação (Figura 42).

Figura 42: Exemplo de alteração de senha

Como se pode ver pela Figura 42, a sequência de alteração de código é realizada por 3 operações de *Dataflow* (Figura 43):

– "Separadores": Operação que verifica se os separadores (*) estão nas posições corretas, de forma a efetuar a validação da mudança de código;

– "*OldCodeOK*": Compara os valores do código antigo (que se encontra entre as posições 12 até 15 da memória), com o código anterior ([-1]) que está nas variáveis "COD1", "COD2", "COD3", "COD4";

– "*NewCode2X*": Operação que recebe a nova combinação de código, que deve de ser inserida duas vezes. A validação do novo código só ocorre se ambas as cópias forem idênticas.

A sequência de alteração senha é construída da seguinte forma (Figura 43):

1º lugar : Insere-se dois separadores (* : tecla 10), para sinalizar o início do processo de alteração de código;

2º lugar : Coloca-se a sequência de código antigo e confirma-se com um separador (*);

3º lugar : Insere-se a nova combinação pretendida e confirma-se com um separador (*);

4º lugar : Insere-se a repetição da combinação do código novo, que será comparada com a cópia anterior, e confirma-se com um separador (*);

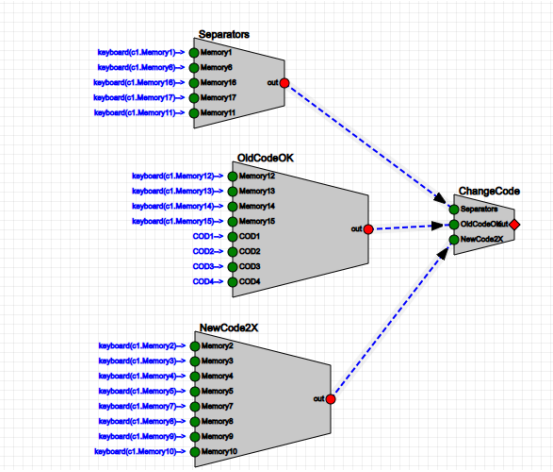


Figura 43: Troço de alteração de senha

De notar que a combinação escolhida é da preferência do utilizador, se o utilizador iniciar a combinação pelos valores de abertura da porta, este tem de terminar a combinação com o caracter "#", se o utilizador pretender fazer alteração à combinação existente, este terá de iniciar combinação com o caracter "*".

No Anexo III, apresenta-se o código Python gerado automaticamente para o modelo da Figura 39. Este código foi depois aplicado na placa *Raspberry Pi* e testado utilizando um teclado universal de 12 botões.

4.2 Comando de um Braço robótico

Este capítulo exibe os modelos de validação que comandam um protótipo que emprega um braço robótico, demonstrando o funcionamento do gerador automático de código Python. Os modelos apresentados usam o braço robótico para realizar tarefas de *pick and place*, tanto de forma manual como de forma automática. O primeiro modelo ("teste_servos") permite comandar o dispositivo utilizando botões de pressão. O segundo modelo ("pick_place_9") permite comandar o dispositivo segundo as trajetórias estipuladas no ambiente IOPT-Flow, sempre que o sensor for acionado.

Para construir o protótipo do braço robótico foi necessário adquirir um kit com as peças da estrutura do robô, uma placa de expansão de portas PWM para o *Raspberry Pi* (PCA9685), um *Raspberry Pi* e 4 motores servo de abertura até 180º (Figura 44).

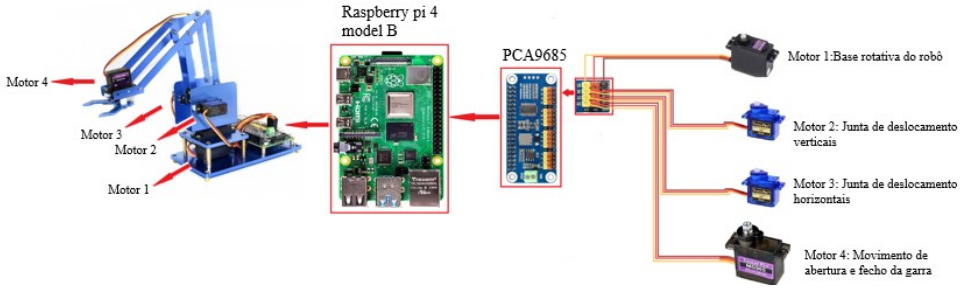


Figura 44: Esquema elétrico Braço Robótico

A Figura 44 mostra o esquema de ligações do protótipo. Neste esquema é possível ver que a placa PCA9685 é o elo de ligação entre os motores servos com a placa *Raspberry pi,* através de protocolo I2C. Esta placa de expansão de portas PWM foi utilizada pois o *Raspberry Pi* é bastante pobre nesse aspeto, oferecendo apenas 2 canais de PWM internos. Este dispositivo possui um chip próprio da *Adafruit* com capacidade de gerar sinais até 16 canais com 12 bits de precisão, mas nesta aplicação só foi necessário usar 4 canais para controlar os motores. Apesar do esquema da Figura 44 não exibir a existência de botões de pressão (sensores), deve-se ao fato de ambos os modelos de validação usufruírem desse componente, mas em quantidades diferentes. Para movimentar o braço robótico de forma automática, utiliza-se 1 botão para simular um sensor, cuja função é detetar a presença de objetos e iniciar o processo. O modelo que movimenta o braço robótico de forma manual, utiliza 8 botões de pressão em que 4 deles comandam o sentido horário e os restantes comandam o sentido anti-horário dos motores.

Os motores servo são comandados através de pulsos PWM, em que a duração dos impulsos define a posição do motor. Quando um motor recebe o sinal de PWM, este desloca-se até a posição definida e fica estático nessa posição até que novos sinais de pulso sejam enviados e façam este motor mudar de posição.

Os motores servos utilizados têm uma gama de posições entre -90º a 90º, e recebem sinais de PWM com uma frequência de 50 Hz (t=1/50 = 20 ms de pulso). De acordo com a folha de dados

da placa PCA9685, para obter uma frequência de 50Hz a partir de um oscilador de 25MHz, é necessário definir o seguinte divisor de relógio:

$$\left(\frac{25MHz}{4096*50}\right) - 1 = 122 \rightarrow (0x7A) \quad (1)$$

Uma vez definida a frequência PWM, é possível movimentar cada servo motor para diferentes posições de acordo com a largura dos impulsos (Figura 45). Para determinar a duração de cada pulso, utiliza-se a fórmula :

$$Pulsos = \left(\frac{4096*duração}{20ms}\right) \quad (2)$$

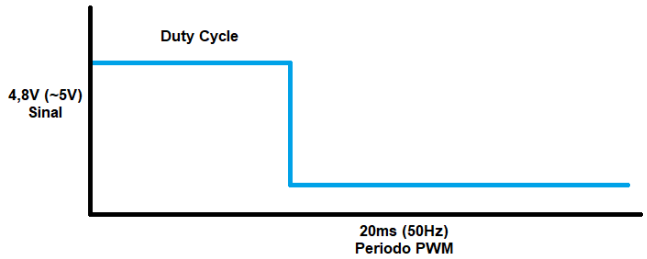


Figura 45: Gráfico PWM

A largura de pulso deverá variar entre os 0,5 ms até 2,5 ms, que representa o *duty-cycle* num período de 20ms (Figura 45).

Aplicando a fórmula (2) para determinar a largura dos pulsos, correspondente à posição angular desejada do motor, obtém-se os seguintes resultados:

– 0,5ms → 102 pulsos → 2,5% de 4096 →+90º

– 2,5ms → 512 pulsos → 12,5% de 4096 →-90º

Para verificar a linearidade da escala de rotação dos motores, foi definido uma gama de valores com 8 divisões iguais. O objetivo era dividir os 180º por 8 intervalos de forma a verificar a precisão da rotação dos motores. Com base neste teste, foi possível concluir que um deslocamento de 22,5º da posição do motor corresponde aproximadamente a 51 pulsos (Figura 46).

Pulsos	102	153	204	255	306	357	408	459	510
Ângulo do motor	+90º (+22,5º)	+67,5º	+45º	+22,5º	+0º	-22,5º	-45º	-67,5º	-90º

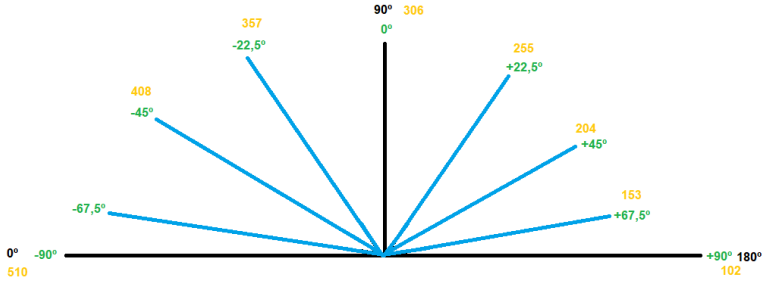


Figura 46: Escala de linearidade dos motores

Partindo deste resultado foi definida uma fórmula para calcular a largura dos impulsos correspondente a cada posição do motor. Esta fórmula que posteriormente foi aplicada no ficheiro "external_components" (Listagem 18), utiliza os ângulos em graus para determinar o PWM.

$$PWM_{DC} = 306 - \frac{204 * Ângulo}{90} \quad (3)$$

Componente Externo: Para programar a placa PCA9685, foi necessário adicionar novas funcionalidades à transformação XSL existente, de forma a conseguir suportar componentes externos em linguagem Python. Um componente externo ("extern/Foreign") é uma funcionalidade das IOPT-Flow que permite importar ficheiros escritos à mão, de uma dada linguagem, para dentro do programa gerado automaticamente. Esta funcionalidade possui a vantagem de oferecer mais versatilidade às aplicações geradas pelos modelos desenvolvidos no ambiente IOPT-Flow.

O modelo IOPT-Flow da Figura 48, utiliza arcos de leitura para transmitir a informação sobre a posição dos motores ao componente externo. Este componente de características externas, recebe a informação proveniente dos arcos de leitura e converte a posição do motor (em graus) para largura de impulsos PWM. Desta forma o modelo IOPT-Flow define a posição desejada em graus, mas a conversão para a duração de impulsos e a comunicação I2C é feita pelo código escrito à mão que se encontra no ficheiro "external_components.py" (Listagem 18).

Python_syntax.XSL

```
<xsl:if test="//component[@target = 'external']">
    <xsl:text>
from external_components import </xsl:text>
    <xsl:for-each select="header/component-list/component[@class]">
        <xsl:value-of select="translate(@class,'/-+. ','_____')" />
        <xsl:text>_init</xsl:text>
        <xsl:text>,</xsl:text>
        <xsl:value-of select="translate(@class,'/-+. ','_____')" />
        <xsl:text>_step</xsl:text>
        <xsl:if test="position() != last()">, </xsl:if>

    </xsl:for-each>
</xsl:if>
```

Importação da biblioteca "external_componentes"

model_code.py

```
from external_components import local_pwm_servo_4mtr_xml_init,local_pwm_servo_4mtr_xml_step
```

Listagem 18: Biblioteca external_components

Pseudocódigo XSL:

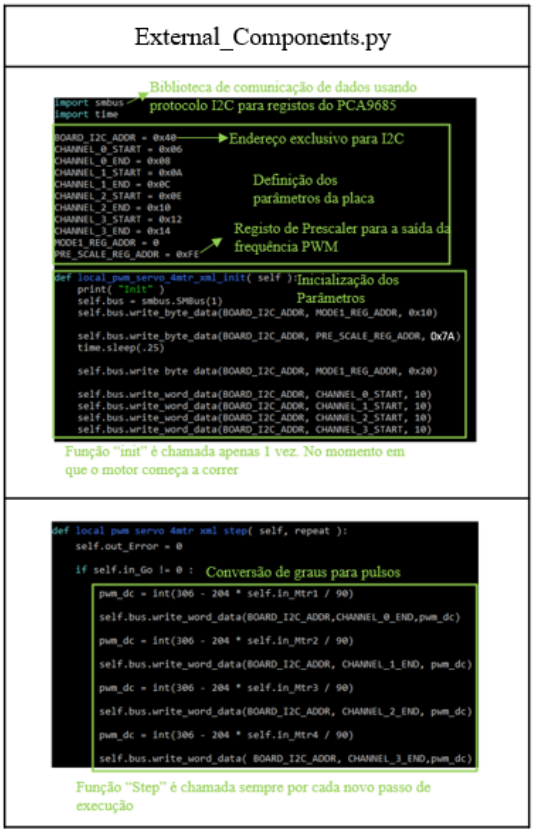
SE existir componente externo no modelo
  ESCREVE 'from external_components import'
  PARA CADA elemento 'component [@class]'
    APLICA e traduz o atributo xml + ESCREVE '_init' + ESCREVE ','
    APLICA e traduz o atributo xml + + ESCREVE '_step'

O código gerado automaticamente preenche o valor das entradas do componente externo com os parâmetros que vêm do modelo através de arcos, que depois chamam a função "step()" e "init()" do ficheiro escrito manualmente (Anexo IV). O ficheiro escrito à mão

(“external_componenets.py”), contém as funções “local_pwm_servo_4mtr_XML_init(self)” e “local_pwm_servo_4mtr_XML_step(self, repeat)” (Listagem 19).

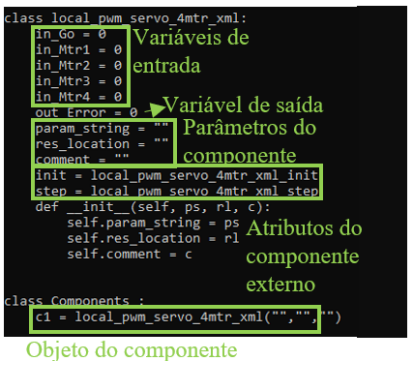
“local_pwm_servo_4mtr_XML_init(self)” – Função que inicializa os parâmetros PWM, SMBus e a placa PCA9685. A biblioteca SMBus, encontra-se incluída na instalação padrão do ambiente *Raspberry Pi OS* (antigo *Raspbian* [47]);

“local_pwm_servo_4mtr_XML_step(self, repeat)” – Função que faz a conversão de graus para pulsos PWM e ainda comunica com o controlador PWM via I2C. Esta função lê o valor das entradas do componente (valor em graus) e envia o valor de *duty-cycle* calculado para a placa PCA9685, através do protocolo I2C;



Listagem 19: Ficheiro externo "External_Components”

Para cada classe de componentes externos, é criado uma nova classe de código Python. Cada classe de componentes irá possuir: variáveis correspondentes a cada entrada e saída do componente externo do modelo IOPT-Flow; parâmetros de *string*, localização de componentes e comentários; funções do documento importado “external_compoents.py” (Listagem 20).

Python_syntax.XSL	model_code.py
	

Listagem 20: Classe componentes externos

Pseudocódigo XSL:

SE existir componente externo no modelo

ORDENAR os nomes da classe por ordem alfabética

ESCREVE ‘class’ + EXIBE e transforma o nome da classe + ESCREVE ‘:’

POR CADA variável de entrada no bloco de componentes externos

ESCREVE ‘in_’ + EXIBE nome da variável + ESCREVE ‘=0’

POR CADA variável de saida no bloco de componentes externos

ESCREVE ‘out_’ + EXIBE nome da variável + ESCREVE ‘=0’

ESCREVE parâmetros dos componentes externos definidos no modelo

ESCREVE ‘init = ‘ + EXIBE e transforma o nome da classe + ESCREVE ‘_init’

ESCREVE ‘step = ‘ + EXIBE e transforma o nome da classe + ESCREVE ‘_step

DEFINE função construtora que recebe e guarda os parâmetros definidos no modelo

ESCREVE “class componentes :”

POR CADA componente externo

EXIBE ‘id’ do componente + ESCREVE ‘=’ + EXIBE e transforma o nome da classe

ESCREVE ‘(“ ‘ + EXIBE e transforma o nome de ‘param_string’

ESCREVE ‘ “,“ ‘ + EXIBE e transforma o nome de ‘res_location’

ESCREVE ‘ “,“ ‘ + EXIBE e transforma o nome de ‘comment’ + ESCREVE ‘ “)’

A aplicação do componente externo no modelo de validação tem o objetivo de realizar o comando dos motores. Devido a esta situação, este componente tem de possuir um número de entradas equivalente ao número de motores utilizados, que neste caso serão 4. O tipo de dados de cada entrada corresponde ao intervalo de rotação dos motores; -90º a +90º

Modelo "test_servos_2": O modelo da Figura 47 realiza o controlo do robô de forma manual, através de 8 botões, em que 4 rodam os motores no sentido dos ponteiros do relógio (CW) e os restantes rodam no sentido anti-horário (CCW). O componente tem como entradas o evento "Go" e como saída os sinais que comandam os 4 motores. O comando "Go" inicia o processo de funcionamento dos motores, se a posição anterior de qualquer um deles for diferente da atual. Se esta condição se confirmar, o componente externo recebe os parâmetros dos 4 motores em graus e calcula a duração dos impulsos PWM utilizando a fórmula (3) (Figura 47).

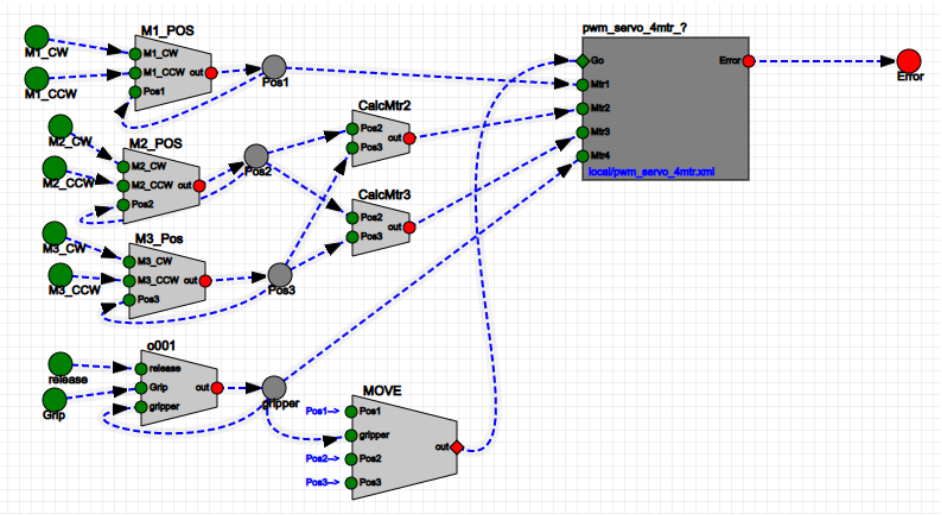


Figura 47: Controlo do robô por botões

Neste modelo o controlo dos motores é feito de forma independente (Figura 47), usando os seguintes sinais de entrada:

M1_CW e M1_CCW – Comandam os movimentos do motor 1, que se localiza na base do braço robótico. Realiza movimentos rotativos em torno do eixo de Z.

M2_CW e M2_CCW – Comandam os movimentos do motor 2, que se localiza na lateral do braço robótico. Realiza movimentos verticais em torno do eixo de X para o eixo de Y e vice-versa.

M3_CW e M3_CCW – Comandam os movimentos do motor 3, que se localiza na parte lateral do braço robótico. Realiza movimentos horizontais ao longo do eixo de X.

Grip e Release – Comandam os movimentos do motor 4, que se localiza na garra do robô. Realiza movimentos de abertura e fecho da garra.

Os sinais apresentados são usados para interpretar as ações do utilizador exerce sobre os botões, para em seguida, implementar os movimentos desejados. Para evitar o sobreaquecimento dos motores 2 e 3, foi utilizada uma estratégia de controlo que permite minimizar o seu esforço. Esta estratégia foi definida devido ao fato dos motores 2 e 3 estarem montados paralelamente e qualquer movimento que um dos motores realize, irá gerar um pequeno esforço sobre o outro, causando um sobreaquecimento que pode provocar danos permanentes. Para resolver esta situação, foi adicionado ao modelo da Figura 47 duas operações de fluxo de dados que calculam a posição dos dois motores ("CalcMtr2" e "CalcMtr3") de forma indireta. Esta estratégia faz com que as deslocações horizontais sejam realizadas movimentando ambos os motores na mesma direção e as deslocações verticais sejam feitas movimentando os motores em sentidos opostos, dividindo sempre o esforço entre os dois atuadores elétricos. As operações que determinam a posição do motor 2 e motor 3, têm o objetivo de minimizar qualquer tipo de sobrecarga nos atuadores, quando está a ser realizado um movimento do robô. A fórmula que define cada uma destas variáveis é:

- "CalcMtr2": Pos2 – Pos3 / 2

- "CalcMtr3": Pos2 + Pos3 / 2

O Anexo V mostra o código Python gerado automaticamente através do modelo da Figura 47. Este código foi aplicado e testado na placa *Raspberry Pi*, demonstrando a validade do gerador desenvolvido.

Modelo "pick_place_9": O modelo da Figura 48 comanda um robô de 4 graus de liberdade, com um motor servo em cada junta. Neste modelo, as trajetórias foram pré-programadas e o movimento é realizado de forma automática.

Para definir as posições do robô, utilizou-se, como auxílio, o programa "test_servo_2" (*Teaching*), que permite realizar o comando do braço robótico através de botões de pressão. Estes botões controlam as juntas do robô de forma praticamente independente, dando liberdade ao utilizador de definir uma sequência de posições que pretende que o robô atinja automaticamente. Para que isso aconteça, o utilizador tem de registar de forma manual os ângulos de cada motor. Este modelo de validação pode ser acedido no ambiente Web IOPT-Flow através do modelo "pick_place_9"(Figura 48).

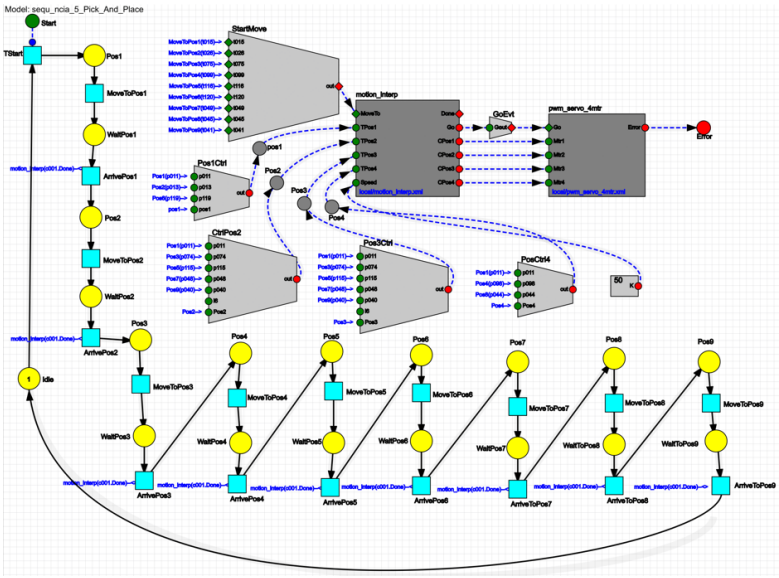


Figura 48: Modelo pick_place_9

O modelo experimental de *pick and place* é apresentado na Figura 48. É importante notar que cada lugar está associado a uma operação que define uma posição diferente no robô. Sendo assim foram traçados os seguintes pontos, que determinam os movimentos que o robô cumpre durante a execução do modelo da Figura 48 (Tabela 14).

	Pos1	Pos2	Pos3	Pos4	Pos5	Pos6	Pos7	Pos8	Pos9
Mtr1	+1º	-79º				+90º			
Mtr2	-10º		+26º		-19º		+28º		+58º
Mtr3	+59º		-8º		+67º		-30º		+18º
Mtr4	-77º			-46º				-77º	

Tabela 14: Tabela final das posições do robô em Graus

Pos 1: Posição inicial (casa) do robô;

Pos 2: Desloca o motor 1 (localizado na base do robô), para a posição onde irá apanhar a peça;

Pos 3: Desloca o motor 2 verticalmente e o motor 3 horizontalmente de maneira a posicionar-se para agarrar a peça (Motores localizados na lateral do robô);

Pos 4: Motor 4 (localizado na garra do robô), que realiza a ação de apanhar a peça;

Pos 5: Motor 2 e Motor 3 realizam o recuo do braço robótico, esta ação é implementada simultaneamente, enquanto que o motor 4 mantém a garra fechada ;

Pos 6: Motor 1 realiza uma rotação de aproximadamente 180º, deixando o braço robótico em posição de poder soltar a peça no local indicado;

Pos 7: Motor 2 e Motor 3 implementa a ação de extensão do braço robótico, de forma a que este consiga alcançar a posição definida para soltar a peça;

Pos 8: Motor 4 realiza a ação de abertura da garra;

Pos 9:Motor 2 e Motor 3 realizam o recuo do braço robótico, após a colocação da peça no local determinado;

O modelo da Figura 48 é composto por uma máquina de estados, que define a sequência de funcionamento do processo. Ao existirem 18 estados, o processo terá 9 estados que iniciam os movimentos dos motores associados e os restantes 9 estados aguardam pela conclusão dos mesmos movimentos. De acordo com o modelo descrito, o método de deslocamento é definido através do componente "motion_interp", o ângulo de deslocação do motor é definido pelas operações "Pos1Ctrl", "CtrlPos2", "Pos3Ctrl" e "PosCtrl4" e ordem de posições é definida pela Rede de Petri.

O modelo da Figura 48 permite mover os motores servos de forma individual ou em simultâneo. Estas ações dos movimentos, dependem da forma de como os pontos foram traçados e com base no deslocamento, seleciona-se o número necessário de motores que são capazes de concluir os movimentos. Para ter um melhor controlo sobre os movimentos de cada motor, o ambiente IOPT-Flow possui um modelo que realiza deslocações passo-a-passo de cada um dos motores ("motion_Interp"). Este modelo tem o objetivo de efetuar movimentos suaves dos motores, desempenhando deslocamentos de 1 grau de cada vez, de forma a relacionar a posição atual com a posição desejada, para o qual o motor se pretende deslocar (Figura 50).

O anexo VI mostra o resultado da transformação do modelo da Figura 48, construído em DS-Pnet através do ambiente Web IOPT-Flow, para código Python. O código Python gerado, foi posteriormente aplicado no protótipo físico, de forma a confirmar a validade da geração de código. O modelo da Figura 49 ilustra um exemplo prático, para a utilização do código extraído no ambiente IOPT-Flow num processo industrial. O objetivo é fazer com que o braço robótico realize a passagem de latas de tinta da passadeira rolante, para uma zona de empacotamento, sempre que o sensor deteta uma lata nova.

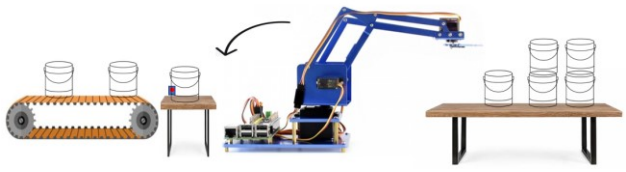


Figura 49: Aplicação Braço Robótico

Modelo "motion_Interp": Modelo que calcula a distância entre os pontos de cada um dos motores e de seguida altera a sua posição (simultaneamente ou individualmente), 1 grau de cada vez, realizando assim, uma interpolação entre todos os eixos, de maneira a que seja possível todos chegarem ao destino final simultaneamente. A distância tem bastante influência na velocidade do motor, a posição mais afastada será a primeira a iniciar e a efetuar os movimentos de forma mais rápida. Este modelo desempenha um papel bastante importante na gestão das posições intermédias do braço robótico e possui uma saída que informa quando os motores atingiram as posições definidas (Figura 50).

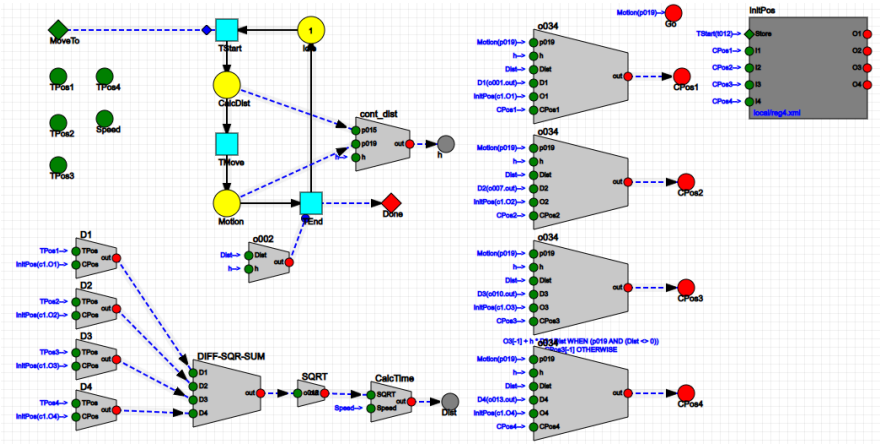


Figura 50:Modelo Motion Interpolation

O modelo "motion_interp" (Figura 50) é composto por 4 entradas que estão associadas a cada um dos motores. Essas entradas recebem o valor da posição de cada um dos motores em graus. O modelo também contém um evento "MoveTo" que dá ordem de início dos movimentos e possui também uma entrada que recebe uma constante que define a velocidade ("Speed") dos movimentos numa gama de 0 a 100.

De acordo com a Figura 50, as operações D1,D2,D3 e D4 calculam a diferença entre a nova posição e a posição antiga (TPos...: nova posição; CPos...: Posição antiga), operação "DIFF_SQR_SUM" realiza a soma dos quadrados das posições provenientes de D1,D2,D3 e D4.

A operação "SQRT" realiza a raiz quadrada do resultado de "DIFF_SQR_SUM", determinando a distância que os motores se irão deslocar. A operação "CalcTime" multiplica o valor da distância determinada na operação "SQRT" por 40 e divide pela constante da velocidade definida pelo utilizador, de maneira a descobrir o valor final da distância total que os motores têm de percorrer. A variável "Dist" corresponde ao resultado das operações D1,D2,D3 e D4 em conjunto com a operação "DIFF_SQR_SUM" e "SQRT", que neste caso corresponde ao valor da raiz quadrada dos quadrados das diferenças entre as posições novas e as antigas. A Figura 51 exibe um exemplo de funcionamento deste grupo de operações. Neste exemplo, as operações recebem parâmetros de posições novas (TPos) e os cálculos serão realizados de acordo como os novos dados inseridos:

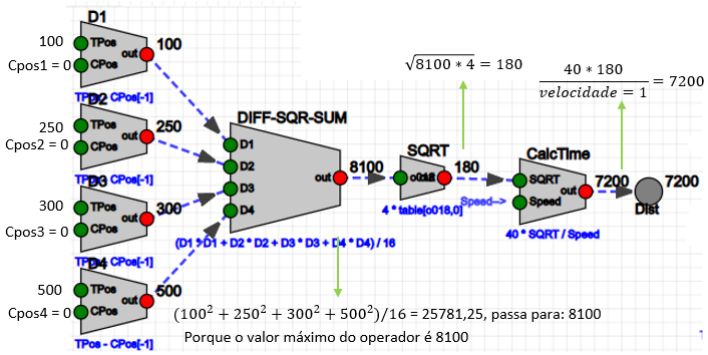


Figura 51: Exemplo de cálculo da distância

A operação “cont_dist” (Figura 50), é um contador que realiza uma contagem crescente desde 0 até à distância máxima determinada na variável “Dist” (Figura 51). À medida que o contador aumenta, vai sendo aplicada a fórmula (4) :

$$pos = pos\ inicial + h * \frac{diff\_pos}{dist} \qquad (4)$$

A “pos” corresponde à posição final, o “h” corresponde ao sinal interno do contador, “dist” é a distância a alcançar e “diff_pos” corresponde à diferença entre a posição final e a inicial.

Modelo “initPos”: O componente “initPos” da Figura 50, funciona como um registo do sistema. Este registo memoriza as coordenadas iniciais do motor, antes destes começarem a andar (“Cpos1”, “Cpos2”, “Cpos3”, “Cpos4”) (Figura 52).

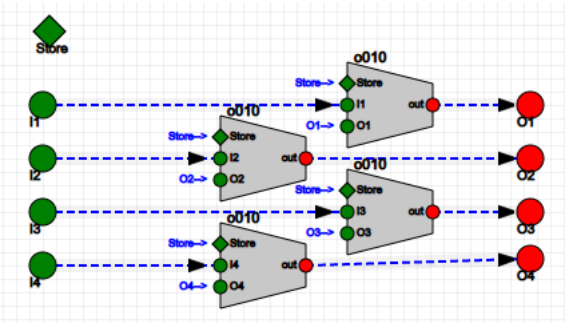


Figura 52: Modelo InitPos

Finalmente é importante referir que o *Raspberry Pi* utiliza um sistema operativo multitarefa que partilha o tempo de CPU por múltiplas aplicações que correm em simultâneo, causando irregularidades nos movimentos do braço robótico, exibindo muita vibração. Para resolver este problema foi necessário correr o código gerado em modo real-time. Desta forma, o controlador do braço robótico corre com mais prioridade, efetuando os movimento de forma suave e sem perturbações.

4.3. Notas Finais

Com a existência desta nova ferramenta de geração automática de código Python, é possível fornecer ao utilizador mecanismos de programação mais apelativos do que o código em formato de texto. O novo gerador de código está integrado no ambiente IOPT-Flow, permitindo tirar partido das ferramentas de simulação e validação, evitar possíveis avarias que seriam provocadas por erros de codificação e erros que são detetados na simulação, antes de executar o código no protótipo.

Quando o utilizador não domina as linguagens de programação e necessitar de uma aplicação para um dado projeto, esta ferramenta de geração automática de código permite que este construa um modelo que depois poderá exportar para essa linguagem. Desta forma, o utilizador apenas tem de aplicar o código gerado automaticamente diretamente no dispositivo de hardware sem qualquer problema e sem erros.

Para quem procura enveredar pelo mundo do software, ou quem não possua nenhuma base sobre programação, o gerador de código automático existente no ambiente IOPT-Flow é uma boa opção para quem quer iniciar. O processo de construção e geração de código fornece bases lógicas de programação ao utilizador, permitindo que este, depois de exportar o modelo para a linguagem escolhida, consiga associar as tarefas que se encontram na linguagem pretendida com as ações exibidas no modelo.

A escrita normal de *software* é habitualmente um processo demorado. A linguagem DS-Pnet fornece ao utilizador a possibilidade de realizar algumas tarefas que poderiam ser escritas à mão, mas de uma forma rápida e simples. A figura 53 apresenta duas especificações diferentes de um sistema que comanda um Led utilizando um botão. A imagem do lado esquerdo exibe o código Python desenvolvido manualmente e do lado direito apresenta o modelo DS-Pnet equivalente. Ao comparar o programa escrito em código Python com o modelo construído em DS-Pnet, é possível notar que o modelo apresentado no ambiente Web do IOPT-Flow, é mais simples, rápido e atraente para utilizadores que não têm conhecimento de programação em linguagens tradicionais.

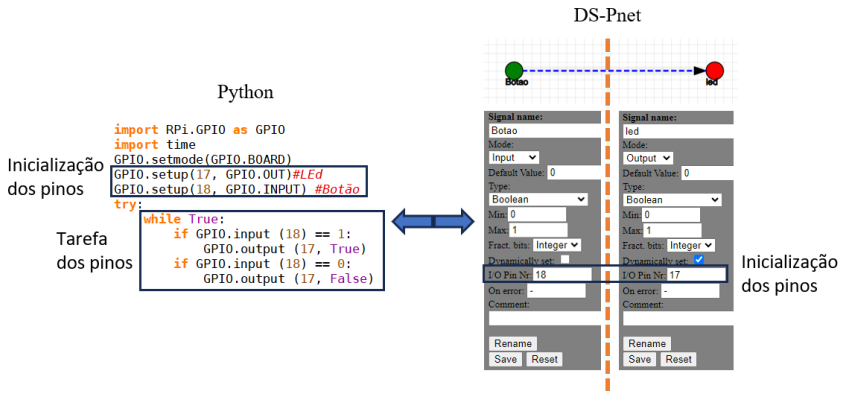


Figura 53: Exemplo de especificação de comando de um led utilizando um Botão

A associação dos pinos na imagem do lado esquerdo é realizada manualmente onde o utilizador define as características do pino e atribui tarefas ao mesmo. A associação de pinos a sinais na imagem do lado direito é realizada através da janela de trabalho (Figura 8). Na coluna de propriedades dos elementos é definido um número de GPIO nas propriedades de cada sinal de entrada e saída.

Observando o código gerado automaticamente para os modelos de validação apresentados neste capítulo, verificamos que a complexidade do código gerado é substancialmente maior do que os modelos DS-Pnet originais. Por exemplo, para o modelo de Alarme de porta, o código apresentado no Anexo III contém 590 linhas de código (cerca 10 páginas.) que correspondem ao 3 submodelos DS-Pnet (figuras 36, 37 e 39). No modelo de validação de comando de um braço robótico, a geração automática de código produziu 2734 linhas (Anexo VI – cerca de 42 páginas), das quais 2049 correspondem a uma tabela de valores (cerca de 32 páginas), que correspondem aos 3 submodelos criados em DS-Pnet (figuras 48, 50 e 52). Com esta comparação é possível concluir que é vantajoso construir modelos em DS-Pnet e gerar código automaticamente, em detrimento da escrita manual de código, mesmo que o código escrito manualmente seja mais sucinto do que o código gerado automaticamente.

Capítulo 5. Conclusão

A popularidade da linguagem Python tem vindo a aumentar ao longo destes últimos anos e por esse motivo existe uma grande comunidade de utilizadores, que se servem desta linguagem de alto nível para os diversos tipos de aplicações. A ferramenta proposta permite tirar partido de bibliotecas e funcionalidades oferecidas pela linguagem Python a partir do ambiente Web IOPT-Flow, possibilitando o desenvolvimento de aplicações sem recurso a escrita manual de código. Desta forma, o gerador de código apresentado neste documento simplifica a construção de controladores para sistemas embutidos e ciber-físicos, contribuindo para reduzir o tempo de desenvolvimento e minimizar os erros de codificação.

Os controladores são criados através do desenho de modelos gráficos, que posteriormente são simulados e verificados usando a interface Web do ambiente IOPT-Flow. Após os modelos estarem testados usando as ferramentas anteriores, a geração automática de código produz programas que correm diretamente nas plataformas de *hardware* usadas nos sistemas ciber-físicos/embutidos. O simulador e o gerador automático de código Python partilham a mesma semântica de execução, contribuindo para evitar que o comportamento dos sistemas finais seja diferente ao que havia sido observado durante a simulação.

As aplicações apresentadas neste relatório utilizam a linguagem Python, em conjunto com as bibliotecas disponíveis, para controlar o *hardware* dos modelos de validação, como o sistema de alarme de porta e o braço robótico. Os modelos de validação tiram partido das diversas funcionalidades oferecidas pela linguagem, como por exemplo as bibliotecas para ler e escrever pinos de entrada/saída e para comunicar com a placa controladora de PWM usando um barramento I2C.

Os modelos de validação foram selecionados para mostrar a versatilidade da ferramenta, podendo ser usada para desenvolver sistemas de controlo industriais, sistemas para uso doméstico, entretenimento e muitos outros. Ao longo do processo de conceção da ferramenta automática, foram concebidos diversos modelos simples não descritos neste relatório, que serviram também como protótipos de validação. Todos os protótipos baseados no código gerado, funcionaram corretamente, demonstrando assim a validade da ferramenta desenvolvida.

A partir de agora, qualquer utilizador que pretenda gerar código Python automaticamente para manipular um sistema ciber-físico ou embutido recorrendo a uma linguagem de modelação gráfica, pode utilizar o ambiente Web IOPT-Flow. Contudo, é necessário que estes dispositivos tenham interpretadores de Python para correr o código gerado, como é no caso do *Rasperry Pi*.

A realização deste trabalho abre diversas possibilidades para projetos futuros. Um dos possíveis projetos consiste no desenvolvimento de ferramentas gráficas para implementar as interfaces gráficas com o utilizador (GUI), utilizando as bibliotecas disponíveis na linguagem Python, para fazer interface com ecrãs táteis. Esta aplicação seria bastante útil para criar consolas gráficas para controlo de máquinas ou *dashboards* para monitorizar processos industriais, sistemas SCADA, etc. Em alternativa às interfaces gráficas a correr em *hardware* local, a linguagem Python também oferece bibliotecas que no futuro permitirão implementar interfaces remotas via Web. Para

além disso, seria útil criar uma biblioteca de componentes externos para fazer interface com diversos dispositivos e recursos oferecidos pelo sistema operativo, como por exemplo acesso a bases de dados, aceder ao relógio do computador e comunicação via rede para permitir criar sistemas ciber-físicos distribuídos.

Finalmente o trabalho apresentado neste documento também pode servir como base para a criação de novos geradores de código para outras linguagens como é o caso de texto estruturado para PLCs ou Matlab.

Referências

[1] Reisig, W. (1985). Petri nets: an introduction (Vol. 4). Springer Science & Business Media. ISBN978-3-642-69968-9

[2] Pinto, J. R. C. (3ª edição:2016, Fevereiro). Técnicas de automação. *ETEP- Edições Técnicas e Profissionais*. ISBN 978-972-8480-26-4

[3] francês, C. R. L. (2003, Agosto). Introdução às redes de Petri. *Laboratório de Computação Aplicada, Universidade Federal do Pará*. URL: https://www.dca.ufrn.br/~affonso/DCA0409/pdf/redes_de_Petri.pdf (acedido em: 14-10-2022)

[4] Pereira, F. J. G. (2017). *The DS-Pnet modeling formalism for cyber-physical system development* (Doctoral Dissertation, Universidade Nova de Lisboa).

[5] Petri, C. A. (1980). Introduction to general net theory. In *Net theory and applications* (pp. 1-19). Springer, Berlin, Heidelberg. URL: https://link.springer.com/content/pdf/10.1007/3-540-10001-6_21.pdf

[6] Campos-Rebelo, R., Pereira, F., Moutinho, F., & Gomes, L. (2011, July). From IOPT Petri nets to C: An automatic code generator tool. In *2011 9th IEEE International Conference on Industrial Informatics* (pp. 390-395). doi: 10.1109/INDIN.2011.6034908.

[7] What is Arduino? *Arduino Documentation.* (2015). URL:https://search.iczhiku.com/paper/TFzDJhGhd6VMaDsI.pdf. (acedido em: 11-10-2022)

[8] Arduino DOCS. (2006). MicroPython with Arduino Boards. URL: https://docs.Arduino.cc/learn/programming/Arduino-and-Python.(acedido em: 11-10-2022)

[9] Wei-Peng, Z., Li-Sha, C., Er-Min, L., & Feng-Chun, Z. (2020, June). Design and Production of Tracking System based on OpenMV Image Recognition. In *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)* (pp. 1198-1202). doi: 10.1109/ITOEC49072.2020.9141560.

[10] Python Software Foundation. (2001). IDLE. *Python 3.11.0 documentation.* URL:https://docs.Python.org/3/library/idle.html (acedido em:11-10-2022)

[11] Rosa, E. A. M. (2018). *Decomposição de Redes de Petri IOPT*. FCT: DEE - Dissertações de Mestrado. URL: https://run.unl.pt/bitstream/10362/56307/1/Rosa_2018.pdf

[12] Rocha, J. I., Gomes, L., & Dias, O. P. (2011, July). Dataflow model property verification using Petri net translation techniques. In *2011 9th IEEE International Conference on Industrial Informatics* (pp. 783-788).doi: 10.1109/INDIN.2011.6034993.

[13] Maciel, P. R., Lins, R. D., & Cunha, P. R. (1996, Julho). Introdução às redes de Petri e aplicações. *UNICAMP-Instituto de Computacao.*URL: https://www.researchgate.net/profile/Paulo-Maciel-/publication/247935030_Introducao_as_redes_de_Petri_e_aplicacoes/links/53fe1afc0cf283c3583b6a83/Introducao-as-redes-de-Petri-e-aplicacoes.pdf

[14] Oliveira, W. R., & Ferramola, M. L. (1999, Dezembro). Características e tipos de Petri Nets. *UFPE-Centro de Informática Linguagens e Máquinas*. URL: https://www.cin.ufpe.br/~if114/Monografias/Petri%20Nets/tipos.html (acedido em: 18-10-2022)

[15] Barros, J. P. M. P. (2001). Introdução à modelação de sistemas utilizando redes de Petri. *Instituto Politécnico de Beja. Escola Superior de Tecnologia e Gestão. Portugal*.

[16] Kovalenko, I., Moyne, J., Bi, M., Balta, E. C., Ma, W., Qamsane, Y.,. .. & Barton, K. (2022). Toward an Automated Learning Control Architecture for Cyber-Physical Manufacturing Systems. *IEEE Access*, *10*, 38755-38773. doi**:** 10.1109/ACCESS.2022.3165551

[17] Harrison, R., Vera, D., & Ahmad, B. (2016, March). Engineering methods and tools for cyber–physical automation systems. *Proceedings of the IEEE*, *104*(5), 973-985. doi: 10.1109/JPROC.2015.2510665.

[18] Shafi, Q. (2012, June). Cyber physical systems security: A brief survey. In *2012 12th International Conference on Computational Science and Its Applications* (pp. 146-150).doi: 10.1109/ICCSA.2012.36.

[19] Shangguan, L., & Gopalswamy, S. (2019). Health monitoring for cyber physical systems. *IEEE Systems Journal*, *14*(1), 1457-1467. doi: 10.1109/JSYST.2019.2922982.

[20] Krishnamurthy, P., Khorrami, F., Karri, R., Paul-Pena, D., & Salehghaffari, H. (2018). Process-aware covert channels using physical instrumentation in cyber-physical systems. *IEEE Transactions on Information Forensics and Security*, *13*(11), 2761-2771. doi: 10.1109/TIFS.2018.2833063.

[21] Sacomano, J. B., Gonçalves, R. F., Bonilla, S. H., da Silva, M. T., & Sátyro, W. C. (2018). Indústria 4.0: Conceitos e Fundamentos. *Editora Blucher*. ISBN: 978-85-212-1371-0

[22] Pereira, F., & Gomes, L. (2016). The IOPT-Flow modeling framework applied to power electronics controllers. *IEEE Transactions on Industrial Electronics*, *64*(3), 2363-2372. doi: 10.1109/TIE.2016.2620101

[23] Ferreira, J. A. F. (2015, February). IEC 61131-3 development environment in Eclipse. *Doctoral dissertation, Universidade do Porto*. URL: https://paginas.fe.up.pt/~ee11055/dissertation/Doc/PDI.pdf

[24] Bhattacharyya, S. S., Murthy, P. K., & Lee, E. A. (1999). Synthesis of embedded software from synchronous dataflow specifications. *Journal of VLSI signal processing systems for signal, image and video technology*, *21*, 151-166

[25] Pereira, F., & Gomes, L. (2016, October). The IOPT-Flow framework pairing Petri nets and Dataflows for embedded controller development. *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society* (pp. 4832-4837). doi: 10.1109/IECON.2016.7794152.

[26] Pereira, F., Moutinho, F., & Gomes, L. (2014, July). IOPT-Tools—Towards cloud design automation of digital controllers with Petri nets. *In 2014 International Conference on Mechatronics and Control (ICMC)* (pp. 2414-2419). doi: 10.1109/ICMC.2014.7232002.

[27]Gomes, L., Moutinho, F., & Pereira, F. (2013, September). IOPT-Tools—A Web based tool framework for embedded systems controller development using Petri nets. *In 2013 23rd International Conference on Field programmable Logic and Applications* (pp. 1-1). doi: 10.1109/FPL.2013.6645633.

[28]Pereira, F., Moutinho, F., & Gomes, L. (2012, May). Model-checking framework for embedded systems controllers development using IOPT Petri nets. *In 2012 IEEE International Symposium on Industrial Electronics* (pp. 1399-1404). doi: 10.1109/ISIE.2012.6237295.

[29] Moutinho, F., & Gomes, L. (2014, November). Asynchronous-channels within Petri net-based GALS distributed embedded systems modeling. *IEEE Transactions on Industrial Informatics*, *10*(4), (pp. 2024-2033). doi: 10.1109/TII.2014.2341933.

[30] Moreira, T. G., Wehrmeister, M. A., Pereira, C. E., Petin, J. F., & Levrat, E. (2010, July). Automatic code generation for embedded systems: From UML specifications to VHDL code. *In 2010 8th IEEE International Conference on Industrial Informatics* (pp. 1085-1090). doi: 10.1109/INDIN.2010.5549590.

[31] Krizan, J., Ertl, L., Bradac, M., Jasansky, M., & Andreev, A. (2014, May). Automatic code generation from MATLAB/Simulink for critical applications. *In 2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)* (pp. 1-6). doi: 10.1109/CCECE.2014.6901058.

[32] MathWorks. (1984). MATLAB Coder Generate C and C++ code from MATLAB code. URL: https://www.mathworks.com/products/matlab-coder.html (acedido em: 24-11-2022)

[33] MathWorks. (2022). HDL Coder Generate VHDL and Verilog code for FPGA and ASIC designs. URL: https://www.mathworks.com/products/hdl-coder.html (acedido em: 24-11-2022)

[34] Pastravanu, O. C., & Matcovschi, M. H. (2008, March). Petri Net Toolbox for MATLAB: Petri-net-based approaches to discrete event and hybrid systems in MATLAB-Simulink. *Iasi, Politehnium, 2008.* ISBN: 978-973-621-149-2

[35] Mahulea, C., Matcovschi, M. H., & Pastravanu, O. C. (16-05-2008). PETRI NET TOOLBOX VERSION 2.4.. Department of Automatic Control and Applied Informatics of the Technical University "Gh. Asachi" of Iasi, Romania. URL: http://www.pntool.ac.tuiasi.ro/ (acedido em: 24-11-2022)

[36] MathWorks. (1984). Embedded Coder Generate C and C++ code optimized for embedded systems. URL: https://www.mathworks.com/products/embedded-coder.html. (acedido em: 24-11-2022)

[37] MathWorks. (1984). Simulink Coder Generate C and C++ code from Simulink and Stateflow models. URL: https://www.mathworks.com/products/Simulink-coder.html (acedido em: 24-11-2022)

[38] National Instruments Corporation. (2001, November). LabVIEW Basics 1 Course Manual. Version 6.0. Part Number 322682A-01

[39] Vavilina, E., & Gaigals, G. (2015, November). Improved LabVIEW code generation. *2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)* (pp. 1-4). doi: 10.1109/AIEEE.2015.7367304.

[40] National Instruments (2013, June). LabVIEW C Generator Module. Part Number: 373144C-01. URL: https://www.ni.com/docs/en-US/bundle/labview-c-generator-module/page/lvcgenhelp/cgen_lvcgenhelpdocinfo.html (acedido em: 30-11-2022)

[41] Gomes, D., Campos-Rebelo, R., & Moutinho, F. (2019, October). Web-based Editor for Signal Interpretation Models. In *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society* (Vol. 1, pp. 5892-5897). **DOI:** 10.1109/IECON.2019.8927437.

[42] Cass, S.(23– 08-2022). Top Programming Languages 2022.*IEEE Spectrum*. URL: https://spectrum.ieee.org/top-programming-languages-2022 (acedido em 12-12-2022)

[43] Martins, J. P. (3ª edição: 2019, Fevereiro). Programação em Python: Introdução à Programação Utilizando Múltiplos Paradigmas. Editor IST - Instituto Superior Técnico. Coleção Ensino da Ciência e da Tecnologia. ISBN:978-8481-47-4

[44] Bonet, A. From Simulink to C and Python. *isae-supaero - institut supérieur de l'aéronautique et de l'espace*. URL:https://tristan-ka.github.io/IBOAT_RL/_downloads/SIMULINK_TO_C__PYTHON.pdf (acedido em: 12-12-2022)

[45] Hussain, R. M. (2021). Automatic Code Generation of Petri Net-Based Business Process Models. *COMSATS University Islamabad, Lahore Campus Lahore-Pakistan*

[46] Donat, W., & Krause, C. (2ª edição:2018, July). Learn Raspberry Pi Programming with Python.Learn to Programo n the World's Most Popular Tiny Computer. Apress Berkeley, CA. ISBN: 978-1-4842-3768-7

[47] Piltch, A. (2020, May). Raspberry Pi OS: Why it's no longer called 'Raspbian'. *The oficial Pi operating system is now called 'Raspberry Pi Pi OS'* URL:https://www.tomshardware.com/news/Raspberry-pi-os-no-longer-raspbian (acedido em: 14-12-2022)

[48] Python Software Foundation. (2001). The Python Tutorial. URL:https://docs.Python.org/3/tutorial/ (acedido em: 14-12-2022)

[49] JetBrains s.r.o.(2010).PyCharm. *Get Started: Choose the best PyCharm for you*. URL:https://www.jetbrains.com/help/pycharm/quick-start-guide.html (acedido em: 14-12-2022)

[50] Microsoft.(2015, April). Visual Studio Code. *Getting Started Whit Python in VS Code*. URL:https://code.visualstudio.com/docs/Python/Python-tutorial (acedido em: 14-12-2022)

[51] Raybaut, P. (2009). Spyder. *The Scientific Python Development Environment*. URL:https://www.spyder-ide.org/ (acedido em: 14-12-2022)

[52] Jupyter Team. (2014).Jupyter. *Free software, open standards, and Web services for interactive computing across all programming languages*. URL:https://jupyter.org/ (acedido em: 14-12-2022)

[53] Refsnes Data.(1999). Python Tutorial. URL:https://www.w3schools.com/Python/default.asp (acedido em: 15-12-2022)

[54] Basumallicks, C. (2022, August). What Is Raspberry Pi? Models, Features, and Uses. URL:https://www.spiceworks.com/tech/networking/articles/what-is-Raspberry-pi/ (acedido em: 19-12-2022)

[55] Wikipedia contributors. (2022, Dec 19). System on a chip. In *Wikipedia, The Free Encyclopedia*. Retrieved 18:23, July 20, 2023, from https://en.wikipedia.org/w/index.php?title=System_on_a_chip&oldid=1163919838 (acedido em: 19-12-2022)

ANEXOS

Anexo I

Ficheiro “Input_Output_raspberrypi.py”.

```
import RPi.GPIO as GPIO

def pinMode (pino,modo): #Função que diz o número do pino e o modo
    GPIO.setwarnings(False)#ativa ou desativa mensagens do sistema (False ou True)
    GPIO.setmode(GPIO.BCM)#Define que tipo de portas usar ( Portas da Board ou portas GPIO(BCM))

    if pino.endswith(";PU"):
        GPIO.setup(int(pino [:-3]),modo,pull_up_down=GPIO.PUD_UP)  #Configuração do pino para Pull-Up (PUD_UP) / int(pino [:-3])-Transforma o valor da string num inteiro e elimina o ";PU"
    elif pino.endswith(";PD"):
        GPIO.setup(int(pino [:-3]),modo,pull_up_down=GPIO.PUD_DOWN)  #Configuração do pino para Pull-Down (PUD_DOWN) / int(pino [:-3])-Transforma o valor da string num inteiro e elimina o ";PD"
    else:
        GPIO.setup(int(pino),modo) # define se o pino funciona como entrada ou saida(GPIO.OUT(0) ou GPIO.IN(1)) entrada:Digitalread Saida:Digitalwrite

def digitalwrite(pino,valor):  #valor da saida do pino (GPIO.HIGH(1) ou GPIO.LOW(0))
    GPIO.output(int(pino), valor)

def digitalread(pino):  #valor da entrada do pino (True(1) ou False(0))
    return GPIO.input(int(pino [:-3]))  #retorna 0 se estiver desligado e 1 se estiver ligado
```

Anexo II

Código Python do modelo da Figura 27.

```
##############################################
 ## Model:_1234                          ##
 ## IOPT-Flow Python Automatic code Generator ##
 ## 2022 (C) Pedro Vale                  ##
 ##############################################


from Input_Output_raspberrypi import pinMode,digitalwrite,digitalread
from time import sleep


class Model_data:
    Start= 0          #input signal
    Value_1= 0      #input signal
    Value_2= 0      #input signal
    s30= 0            #output signal
    s4= 0  #output signal
    e32= 0            #output event
    o014_out= 0    #operation/output
    o016_out= 0    #operation/output
    o025_out= 0    #operation/output
    o026_out= 0    #operation/output

class Model_marking:
    p001= 0
    p003= 0
    p005= 0


class Model_new_marking:
    p001= 0
    p003= 0
    p005= 0


class Model_avail_marking:
    p001= 0
    p003= 0
    p005= 0


class Model_transition_fired:
    t002= 0
    t004= 0
    t006= 0


class Model_shift_reg:
    s4=  []
    p003=  []




def initShiftRegister(depth,value):
    reg =  []
    i = 0
    for i in range(depth):
        reg.append(value)
    return reg

def shiftRegister(reg,value):
    reg.pop()
    reg.insert(0, value)
```

```
def limit(value,min,max):
    if (value < min):
        return  min
    elif (value > max):
        return max
    else:
        return value


def delayPause():
    sleep(0.001)


def setup_io():
    pinMode("10;PD",1)      #Pino: input
    pinMode("11;PD",1)      #Pino: input
    pinMode("12;PD",1)      #Pino: input
    pinMode("2",0)          #Pino: output
    pinMode("4",0)          #Pino: output
    pinMode("3",0)          #Pino: output



def init():
    data.Start = bool ( 0)
    data.Value_1 = 0
    data.Value_2 = 0
    data.s30 = 0
    data.s4 = 4
    data.e32 = bool ( 0)
    marking.p001 = 1
    marking.p003 = 0
    marking.p005 = 0
    shift_reg.s4 = initShiftRegister( 1, 4)
    shift_reg.marking_p003 = initShiftRegister( 5, 0)
    return


def readInputs():
    data.Start = digitalread("10;PD")
    data.Value_1 = digitalread("11;PD")
    data.Value_2 = digitalread("12;PD")
    return


def writeOutputs():
    digitalwrite("2", data.s30)
    digitalwrite("4", data.s4)
    digitalwrite("3", data.e32)
    return


def executionStep():
    new_marking.p001 = 0
    new_marking.p003 = 0
    new_marking.p005 = 0
    avail_marking.p001 = marking.p001
    avail_marking.p003 = marking.p003
    avail_marking.p005 = marking.p005
    transition_fired.t002 = bool ( 0)
    transition_fired.t004 = bool ( 0)
    transition_fired.t006 = bool ( 0)
    data.o014_out = limit( shift_reg.marking_p003 [4], 0, 255 )
```

```
    data.o025_out = bool ( data.Value_1 < data.Value_2)
    data.o026_out = bool ( data.Value_1 > data.Value_2)
    data.s30 = limit( data.o014_out, 0, 255 )
    if (avail_marking.p001 >= 1 and data.Start != 0) :
        transition_fired.t002 = bool ( 1)
        avail_marking.p001 = avail_marking.p001 - 1
        new_marking.p003 = new_marking.p003 + 1

    if (avail_marking.p003 >= 1 and data.o025_out != 0) :
        transition_fired.t004 = bool ( 1)
        avail_marking.p003 = avail_marking.p003 - 1
        new_marking.p005 = new_marking.p005 + 1

    if (avail_marking.p005 >= 1 and data.o026_out != 0) :
        transition_fired.t006 = bool ( 1)
        avail_marking.p005 = avail_marking.p005 - 1
        new_marking.p003 = new_marking.p003 + 1

    data.e32 = bool ( transition_fired.t004)
    data.o016_out = limit( 0 if ( (transition_fired.t004) ) else\
        shift_reg.s4 [0] + 1 if ( (marking.p005) ) else\
        shift_reg.s4 [0], 0, 1000 )
    data.s4 = limit( data.o016_out, 0, 1000 )
    shiftRegister( shift_reg.s4, data.s4)
    shiftRegister( shift_reg.marking_p003, marking.p003)
    marking.p001 = avail_marking.p001 + new_marking.p001
    marking.p003 = avail_marking.p003 + new_marking.p003
    marking.p005 = avail_marking.p005 + new_marking.p005
    return


def main():
    init()
    setup_io()
    while( 1 ):
        readInputs()
        executionStep()
        writeOutputs()
        delayPause()


data = Model_data()
marking = Model_marking()
new_marking = Model_new_marking()
avail_marking = Model_avail_marking()
transition_fired = Model_transition_fired()
shift_reg = Model_shift_reg()
main()
```

Anexo III

Código Python do modelo da Figura 39.

```
##############################################
 ## Model:AlarmeSimples                     ##
 ## IOPT-Flow Python Automatic code Generator ##
 ## 2022 (C) Pedro Vale                     ##
 ##############################################


from Input_Output_raspberrypi import pinMode,digitalwrite,digitalread
from time import sleep


class Model_data:
    s001= 0          #input signal
    s002= 0          #input signal
    s003= 0          #input signal
    s004= 0          #input signal
    s060= 0          #input signal
    COD1= 0          #internal signal
    COD2= 0          #internal signal
    COD3= 0          #internal signal
    COD4= 0          #internal signal
    Counter= 0       #internal signal
    c1_Col_1= 0      #internal signal
    c1_Col_2= 0      #internal signal
    c1_Col_3= 0      #internal signal
    c1_Line_1= 0     #internal signal
    c1_Line_2= 0     #internal signal
    c1_Line_3= 0     #internal signal
    c1_Line_4= 0     #internal signal
    c1_Memory1= -1          #internal signal
    c1_Memory10= -1         #internal signal
    c1_Memory11= -1         #internal signal
    c1_Memory12= -1         #internal signal
    c1_Memory13= -1         #internal signal
    c1_Memory14= -1         #internal signal
    c1_Memory15= -1         #internal signal
    c1_Memory16= -1         #internal signal
    c1_Memory17= -1         #internal signal
    c1_Memory2= -1          #internal signal
    c1_Memory3= -1          #internal signal
    c1_Memory4= -1          #internal signal
    c1_Memory5= -1          #internal signal
    c1_Memory6= -1          #internal signal
    c1_Memory7= -1          #internal signal
    c1_Memory8= -1          #internal signal
    c1_Memory9= -1          #internal signal
    c1_SelKey= -1 #internal signal
    c1_c1_Col1= 0#internal signal
    c1_c1_Col2= 0#internal signal
    c1_c1_Col3= 0#internal signal
    c1_c1_Key= -1            #internal signal
    c1_c1_Line1= 0           #internal signal
    c1_c1_Line2= 0           #internal signal
    c1_c1_Line3= 0           #internal signal
    c1_c1_Line4= 0           #internal signal
    c1_c1_SelKey= -1         #internal signal
    DoorLock= -32768         #output signal
    Intrusion_Alarm_ON= 0            #output signal
    s041= 0          #output signal
    s042= 0          #output signal
    s043= 0          #output signal
    c1_c1_KeyPress= 0        #internal event
    c1_c1_o037_out= 0        #operation/output
    c1_c1_o041_out= 0        #operation/output
    c1_c1_o050_out= -1       #operation/output
```

```
    c1_c1_o057_out= -1      #operation/output
    c1_c1_o064_out= -1      #operation/output
    c1_c1_o075_out= -1      #operation/output
    c1_c1_o082_out= 0       #operation/output
    c1_c1_o087_out= -1      #operation/output
    c1_c1_op1_out= 0        #operation/output
    c1_o001_out= -1         #operation/output
    c1_o002_out= -1         #operation/output
    c1_o003_out= -1         #operation/output
    c1_o004_out= -1         #operation/output
    c1_o009_out= -1         #operation/output
    c1_o020_out= -1         #operation/output
    c1_o021_out= -1         #operation/output
    c1_o034_out= -1         #operation/output
    c1_o098_out= -1         #operation/output
    c1_o099_out= -1         #operation/output
    c1_o102_out= -1         #operation/output
    c1_o103_out= -1         #operation/output
    c1_o105_out= -1         #operation/output
    c1_o123_out= -1         #operation/output
    c1_o126_out= -1         #operation/output
    c1_o127_out= -1         #operation/output
    c1_o129_out= -1         #operation/output
    o001_out= 0    #operation/output
    o004_out= 0    #operation/output
    o012_out= -1   #operation/output
    o014_out= 0    #operation/output
    o015_out= -1   #operation/output
    o018_out= -1   #operation/output
    o021_out= -1   #operation/output
    o026_out= 0    #operation/output
    o029_out= -1   #operation/output
    o040_out= 0    #operation/output
    o044_out= -1   #operation/output
    o046_out= 0    #operation/output
    o050_out= -32768        #operation/output
    o059_out= 0    #operation/output
    o061_out= 0    #operation/output
    o063_out= 0    #operation/output

class Model_marking:
    c1_c1_p001= 0
    c1_c1_p003= 0
    c1_c1_p005= 0
    c1_c1_p012= 0
    c1_c1_p013= 0
    c1_c1_p021= 0
    c1_c1_p022= 0
    p009= 0
    p011= 0


class Model_new_marking:
    c1_c1_p001= 0
    c1_c1_p003= 0
    c1_c1_p005= 0
    c1_c1_p012= 0
    c1_c1_p013= 0
    c1_c1_p021= 0
    c1_c1_p022= 0
    p009= 0
    p011= 0


class Model_avail_marking:
```

```
    c1_c1_p001= 0
    c1_c1_p003= 0
    c1_c1_p005= 0
    c1_c1_p012= 0
    c1_c1_p013= 0
    c1_c1_p021= 0
    c1_c1_p022= 0
    p009= 0
    p011= 0


class Model_transition_fired:
    c1_c1_t002= 0
    c1_c1_t004= 0
    c1_c1_t014= 0
    c1_c1_t015= 0
    c1_c1_t020= 0
    c1_c1_t023= 0
    c1_c1_t024= 0
    t010= 0
    t012= 0


class Model_shift_reg:
    s060=  []
    COD1=  []
    COD2=  []
    COD3=  []
    COD4=  []
    Counter=  []
    c1_Memory1=  []
    c1_Memory10=  []
    c1_Memory11=  []
    c1_Memory12=  []
    c1_Memory13=  []
    c1_Memory14=  []
    c1_Memory15=  []
    c1_Memory16=  []
    c1_Memory17=  []
    c1_Memory2=  []
    c1_Memory3=  []
    c1_Memory4=  []
    c1_Memory5=  []
    c1_Memory6=  []
    c1_Memory7=  []
    c1_Memory8=  []
    c1_Memory9=  []
    c1_c1_Key=  []
    c1_c1_SelKey=  []
    o029_out=  []




def initShiftRegister(depth,value):
    reg =  []
    i = 0
    for i in range(depth):
        reg.append(value)
    return reg

def shiftRegister(reg,value):
    reg.pop()
    reg.insert(0, value)
```

```
def limit(value,min,max):
    if (value < min):
        return  min
    elif (value > max):
        return max
    else:
        return value


def delayPause():
    sleep(0.001)


def setup_io():
    pinMode("23;PD",1)      #Pino: input
    pinMode("24;PD",1)      #Pino: input
    pinMode("25;PD",1)      #Pino: input
    pinMode("8;PD",1)       #Pino: input
    pinMode("16;PD",1)      #Pino: input
    pinMode("20",0)         #Pino: output
    pinMode("26",0)         #Pino: output
    pinMode("14",0)         #Pino: output
    pinMode("15",0)         #Pino: output
    pinMode("18",0)         #Pino: output



def init():
    data.COD1 = 4
    data.COD2 = 3
    data.COD3 = 2
    data.COD4 = 1
    data.Counter = bool ( 0)
    data.DoorLock = 0
    data.Intrusion_Alarm_ON = 0
    data.c1_Col_1 = bool ( 0)
    data.c1_Col_2 = bool ( 0)
    data.c1_Col_3 = bool ( 0)
    data.c1_Line_1 = bool ( 0)
    data.c1_Line_2 = bool ( 0)
    data.c1_Line_3 = bool ( 0)
    data.c1_Line_4 = bool ( 0)
    data.c1_Memory1 = 0
    data.c1_Memory10 = 0
    data.c1_Memory11 = 0
    data.c1_Memory12 = 0
    data.c1_Memory13 = 0
    data.c1_Memory14 = 0
    data.c1_Memory15 = 0
    data.c1_Memory16 = 0
    data.c1_Memory17 = 0
    data.c1_Memory2 = 0
    data.c1_Memory3 = 0
    data.c1_Memory4 = 0
    data.c1_Memory5 = 0
    data.c1_Memory6 = 0
    data.c1_Memory7 = 0
    data.c1_Memory8 = 0
    data.c1_Memory9 = 0
    data.c1_SelKey = -1
    data.c1_c1_Col1 = bool ( 0)
    data.c1_c1_Col2 = bool ( 0)
    data.c1_c1_Col3 = bool ( 0)
```

```
    data.c1_c1_Key = 0
    data.c1_c1_Line1 = bool ( 0)
    data.c1_c1_Line2 = bool ( 0)
    data.c1_c1_Line3 = bool ( 0)
    data.c1_c1_Line4 = bool ( 0)
    data.c1_c1_SelKey = -1
    data.s001 = bool ( 0)
    data.s002 = bool ( 0)
    data.s003 = bool ( 0)
    data.s004 = bool ( 0)
    data.s041 = bool ( 0)
    data.s042 = bool ( 0)
    data.s043 = bool ( 0)
    data.s060 = bool ( 0)
    data.c1_c1_KeyPress = bool ( 0)
    marking.c1_c1_p001 = 1
    marking.c1_c1_p003 = 0
    marking.c1_c1_p005 = 0
    marking.c1_c1_p012 = 0
    marking.c1_c1_p013 = 0
    marking.c1_c1_p021 = 0
    marking.c1_c1_p022 = 0
    marking.p009 = 0
    marking.p011 = 1
    shift_reg.COD1 = initShiftRegister( 1, 4)
    shift_reg.COD2 = initShiftRegister( 1, 3)
    shift_reg.COD3 = initShiftRegister( 1, 2)
    shift_reg.COD4 = initShiftRegister( 1, 1)
    shift_reg.Counter = initShiftRegister( 1, 0)
    shift_reg.c1_Memory1 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory10 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory11 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory12 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory13 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory14 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory15 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory16 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory17 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory2 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory3 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory4 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory5 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory6 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory7 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory8 = initShiftRegister( 1, 0)
    shift_reg.c1_Memory9 = initShiftRegister( 1, 0)
    shift_reg.c1_c1_Key = initShiftRegister( 1, 0)
    shift_reg.c1_c1_SelKey = initShiftRegister( 1, -1)
    shift_reg.s060 = initShiftRegister( 1, 0)
    shift_reg.o029_out = initShiftRegister( 1, -1)
    return


def readInputs():
    data.s001 = digitalread("23;PD")
    data.s002 = digitalread("24;PD")
    data.s003 = digitalread("25;PD")
    data.s004 = digitalread("8;PD")
    data.s060 = digitalread("16;PD")
    return


def writeOutputs():
    digitalwrite("20", data.DoorLock)
    digitalwrite("14", data.s041)
```

```
    digitalwrite("15", data.s042)
    digitalwrite("18", data.s043)
    digitalwrite("26", data.Intrusion_Alarm_ON)
    return


def executionStep():
    new_marking.c1_c1_p001 = 0
    new_marking.c1_c1_p003 = 0
    new_marking.c1_c1_p005 = 0
    new_marking.c1_c1_p012 = 0
    new_marking.c1_c1_p013 = 0
    new_marking.c1_c1_p021 = 0
    new_marking.c1_c1_p022 = 0
    new_marking.p009 = 0
    new_marking.p011 = 0
    avail_marking.c1_c1_p001 = marking.c1_c1_p001
    avail_marking.c1_c1_p003 = marking.c1_c1_p003
    avail_marking.c1_c1_p005 = marking.c1_c1_p005
    avail_marking.c1_c1_p012 = marking.c1_c1_p012
    avail_marking.c1_c1_p013 = marking.c1_c1_p013
    avail_marking.c1_c1_p021 = marking.c1_c1_p021
    avail_marking.c1_c1_p022 = marking.c1_c1_p022
    avail_marking.p009 = marking.p009
    avail_marking.p011 = marking.p011
    transition_fired.c1_c1_t002 = bool ( 0)
    transition_fired.c1_c1_t004 = bool ( 0)
    transition_fired.c1_c1_t014 = bool ( 0)
    transition_fired.c1_c1_t015 = bool ( 0)
    transition_fired.c1_c1_t020 = bool ( 0)
    transition_fired.c1_c1_t023 = bool ( 0)
    transition_fired.c1_c1_t024 = bool ( 0)
    transition_fired.t010 = bool ( 0)
    transition_fired.t012 = bool ( 0)
    data.Intrusion_Alarm_ON = limit( marking.p009, 0, 1 )
    data.c1_c1_o037_out = bool ( marking.c1_c1_p012 or marking.c1_c1_p013)
    data.c1_c1_o041_out = bool ( marking.c1_c1_p021 or marking.c1_c1_p022)
    data.c1_c1_op1_out = bool ( marking.c1_c1_p003 or marking.c1_c1_p005)
    data.c1_Line_1 = bool ( data.s001)
    data.c1_Line_2 = bool ( data.s002)
    data.c1_Line_3 = bool ( data.s003)
    data.c1_Line_4 = bool ( data.s004)
    data.c1_c1_Col1 = bool ( data.c1_c1_op1_out)
    data.c1_c1_Col2 = bool ( data.c1_c1_o037_out)
    data.c1_c1_Col3 = bool ( data.c1_c1_o041_out)
    data.c1_Col_1 = bool ( data.c1_c1_Col1)
    data.c1_Col_2 = bool ( data.c1_c1_Col2)
    data.c1_Col_3 = bool ( data.c1_c1_Col3)
    data.c1_c1_Line1 = bool ( data.c1_Line_1)
    data.c1_c1_Line2 = bool ( data.c1_Line_2)
    data.c1_c1_Line3 = bool ( data.c1_Line_3)
    data.c1_c1_Line4 = bool ( data.c1_Line_4)
    data.c1_c1_o050_out = limit( -1 if ( (marking.c1_c1_p005 == 0) ) else\
       1 if ( (data.c1_c1_Line1 == 1) ) else\
       4 if ( (data.c1_c1_Line2 == 1) ) else\
       7 if ( (data.c1_c1_Line3 == 1) ) else\
       10 if ( (data.c1_c1_Line4 == 1) ) else\
       -1, -1, 11 )
    data.c1_c1_o057_out = limit( -1 if ( (marking.c1_c1_p013 == 0) ) else\
       2 if ( (data.c1_c1_Line1 == 1) ) else\
       5 if ( (data.c1_c1_Line2 == 1) ) else\
       8 if ( (data.c1_c1_Line3 == 1) ) else\
       0 if ( (data.c1_c1_Line4 == 1) ) else\
       -1, -1, 11 )
    data.c1_c1_o064_out = limit( -1 if ( (marking.c1_c1_p022 == 0) ) else\
```

```
        3 if ( (data.c1_c1_Line1 == 1) ) else\
        6 if ( (data.c1_c1_Line2 == 1) ) else\
        9 if ( (data.c1_c1_Line3 == 1) ) else\
        11 if ( (data.c1_c1_Line4 == 1) ) else\
        -1, -1, 11 )
    data.s041 = bool ( data.c1_Col_1)
    data.s042 = bool ( data.c1_Col_2)
    data.s043 = bool ( data.c1_Col_3)
    if (avail_marking.c1_c1_p001 >= 1) :
        transition_fired.c1_c1_t002 = bool ( 1)
        avail_marking.c1_c1_p001 = avail_marking.c1_c1_p001 - 1
        new_marking.c1_c1_p003 = new_marking.c1_c1_p003 + 1

    if (avail_marking.c1_c1_p003 >= 1) :
        transition_fired.c1_c1_t004 = bool ( 1)
        avail_marking.c1_c1_p003 = avail_marking.c1_c1_p003 - 1
        new_marking.c1_c1_p005 = new_marking.c1_c1_p005 + 1

    if (avail_marking.c1_c1_p005 >= 1) :
        transition_fired.c1_c1_t014 = bool ( 1)
        avail_marking.c1_c1_p005 = avail_marking.c1_c1_p005 - 1
        new_marking.c1_c1_p012 = new_marking.c1_c1_p012 + 1

    if (avail_marking.c1_c1_p012 >= 1) :
        transition_fired.c1_c1_t015 = bool ( 1)
        avail_marking.c1_c1_p012 = avail_marking.c1_c1_p012 - 1
        new_marking.c1_c1_p013 = new_marking.c1_c1_p013 + 1

    if (avail_marking.c1_c1_p013 >= 1) :
        transition_fired.c1_c1_t020 = bool ( 1)
        avail_marking.c1_c1_p013 = avail_marking.c1_c1_p013 - 1
        new_marking.c1_c1_p021 = new_marking.c1_c1_p021 + 1

    if (avail_marking.c1_c1_p021 >= 1) :
        transition_fired.c1_c1_t023 = bool ( 1)
        avail_marking.c1_c1_p021 = avail_marking.c1_c1_p021 - 1
        new_marking.c1_c1_p022 = new_marking.c1_c1_p022 + 1

    if (avail_marking.c1_c1_p022 >= 1) :
        transition_fired.c1_c1_t024 = bool ( 1)
        avail_marking.c1_c1_p022 = avail_marking.c1_c1_p022 - 1
        new_marking.c1_c1_p001 = new_marking.c1_c1_p001 + 1

    data.c1_c1_o075_out = limit( -1 if ( (transition_fired.c1_c1_t002 == 1) ) else\
        data.c1_c1_o050_out if ( (data.c1_c1_o050_out >= 0) ) else\
        data.c1_c1_o057_out if ( (data.c1_c1_o057_out >= 0) ) else\
        data.c1_c1_o064_out if ( (data.c1_c1_o064_out >= 0) ) else\
        shift_reg.c1_c1_Key [0], -1, 11 )
    data.c1_c1_Key = limit( data.c1_c1_o075_out, -1, 11 )
    data.c1_c1_o087_out = limit( data.c1_c1_Key if ( (transition_fired.c1_c1_t024 == 1) ) else\
        shift_reg.c1_c1_SelKey [0], -1, 11 )
    data.c1_c1_SelKey = limit( data.c1_c1_o087_out, -1, 11 )
    data.c1_c1_o082_out = bool ( data.c1_c1_SelKey != shift_reg.c1_c1_SelKey [0] and
data.c1_c1_SelKey >= 0)
    data.c1_SelKey = limit( data.c1_c1_SelKey, -1, 11 )
    data.c1_c1_KeyPress = bool ( data.c1_c1_o082_out)
    data.c1_o001_out = limit( shift_reg.c1_Memory10 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
        shift_reg.c1_Memory11 [0], -1, 11 )
    data.c1_o002_out = limit( shift_reg.c1_Memory4 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
        shift_reg.c1_Memory5 [0], -1, 11 )
    data.c1_o003_out = limit( shift_reg.c1_Memory16 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
        shift_reg.c1_Memory17 [0], -1, 11 )
    data.c1_o004_out = limit( data.c1_c1_SelKey if ( (data.c1_c1_KeyPress == 1) ) else\
        shift_reg.c1_Memory1 [0], -1, 11 )
    data.c1_o009_out = limit( shift_reg.c1_Memory1 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
```

```
      shift_reg.c1_Memory2 [0], -1, 11 )
   data.c1_o020_out = limit( shift_reg.c1_Memory2 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
      shift_reg.c1_Memory3 [0], -1, 11 )
   data.c1_o021_out = limit( shift_reg.c1_Memory3 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
      shift_reg.c1_Memory4 [0], -1, 11 )
   data.c1_o034_out = limit( shift_reg.c1_Memory15 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
      shift_reg.c1_Memory16 [0], -1, 11 )
   data.c1_o098_out = limit( shift_reg.c1_Memory5 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
      shift_reg.c1_Memory6 [0], -1, 11 )
   data.c1_o099_out = limit( shift_reg.c1_Memory6 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
      shift_reg.c1_Memory7 [0], -1, 11 )
   data.c1_o102_out = limit( shift_reg.c1_Memory7 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
      shift_reg.c1_Memory8 [0], -1, 11 )
   data.c1_o103_out = limit( shift_reg.c1_Memory8 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
      shift_reg.c1_Memory9 [0], -1, 11 )
   data.c1_o105_out = limit( shift_reg.c1_Memory9 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
      shift_reg.c1_Memory10 [0], -1, 11 )
   data.c1_o123_out = limit( shift_reg.c1_Memory11 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
      shift_reg.c1_Memory12 [0], -1, 11 )
   data.c1_o126_out = limit( shift_reg.c1_Memory12 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
      shift_reg.c1_Memory13 [0], -1, 11 )
   data.c1_o127_out = limit( shift_reg.c1_Memory13 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
      shift_reg.c1_Memory14 [0], -1, 11 )
   data.c1_o129_out = limit( shift_reg.c1_Memory14 [0] if ( (data.c1_c1_KeyPress == 1) ) else\
      shift_reg.c1_Memory15 [0], -1, 11 )
   data.c1_Memory1 = limit( data.c1_o004_out, -1, 11 )
   data.c1_Memory10 = limit( data.c1_o105_out, -1, 11 )
   data.c1_Memory11 = limit( data.c1_o001_out, -1, 11 )
   data.c1_Memory12 = limit( data.c1_o123_out, -1, 11 )
   data.c1_Memory13 = limit( data.c1_o126_out, -1, 11 )
   data.c1_Memory14 = limit( data.c1_o127_out, -1, 11 )
   data.c1_Memory15 = limit( data.c1_o129_out, -1, 11 )
   data.c1_Memory16 = limit( data.c1_o034_out, -1, 11 )
   data.c1_Memory17 = limit( data.c1_o003_out, -1, 11 )
   data.c1_Memory2 = limit( data.c1_o009_out, -1, 11 )
   data.c1_Memory3 = limit( data.c1_o020_out, -1, 11 )
   data.c1_Memory4 = limit( data.c1_o021_out, -1, 11 )
   data.c1_Memory5 = limit( data.c1_o002_out, -1, 11 )
   data.c1_Memory6 = limit( data.c1_o098_out, -1, 11 )
   data.c1_Memory7 = limit( data.c1_o099_out, -1, 11 )
   data.c1_Memory8 = limit( data.c1_o102_out, -1, 11 )
   data.c1_Memory9 = limit( data.c1_o103_out, -1, 11 )
   data.o001_out = bool ( data.c1_Memory1 == 10 and data.c1_Memory6 == 10 and
data.c1_Memory16 == 10 and data.c1_Memory17 == 10 and data.c1_Memory11 == 10)
   data.o014_out = bool ( data.c1_Memory12 == shift_reg.COD1 [0] and data.c1_Memory13 ==
shift_reg.COD2 [0] and data.c1_Memory14 == shift_reg.COD3 [0] and data.c1_Memory15 ==
shift_reg.COD4 [0])
   data.o026_out = bool ( data.c1_Memory2 == data.c1_Memory7 and data.c1_Memory3 ==
data.c1_Memory8 and data.c1_Memory4 == data.c1_Memory9 and data.c1_Memory5 ==
data.c1_Memory10)
   data.o040_out = bool ( data.o001_out and data.o014_out and data.o026_out)
   data.o004_out = limit( data.c1_Memory5 if ( data.o040_out ) else\
      shift_reg.COD4 [0], 0, 9 )
   data.o046_out = limit( data.c1_Memory2 if ( data.o040_out ) else\
      shift_reg.COD1 [0], 0, 9 )
   data.o059_out = limit( data.c1_Memory3 if ( data.o040_out ) else\
      shift_reg.COD2 [0], 0, 9 )
   data.o063_out = limit( data.c1_Memory4 if ( data.o040_out ) else\
      shift_reg.COD3 [0], 0, 9 )
   data.COD1 = limit( data.o046_out, 0, 9 )
   data.COD2 = limit( data.o059_out, 0, 9 )
   data.COD3 = limit( data.o063_out, 0, 9 )
   data.COD4 = limit( data.o004_out, 0, 9 )
   data.o012_out = limit( data.c1_Memory2 == data.COD1, -1, 11 )
   data.o015_out = limit( data.c1_Memory3 == data.COD2, -1, 11 )
```

```
    data.o018_out = limit( data.c1_Memory4 == data.COD3, -1, 11 )
    data.o021_out = limit( data.c1_Memory5 == data.COD4, -1, 11 )
    data.o029_out = limit( data.o012_out and data.o015_out and data.o018_out and data.o021_out
and (data.c1_SelKey == 11), -1, 11 )
    data.o044_out = limit( 200 if ( data.o029_out == 1 and shift_reg.o029_out [0] == 0 ) else\
        shift_reg.Counter [0] - 1 if ( shift_reg.Counter [0] > 0 ) else\
        0, -1, 11 )
    data.Counter = bool ( data.o044_out)
    data.o050_out = limit( data.Counter != 0, -32768, 32767 )
    data.DoorLock = limit( data.o050_out, -32768, 32767 )
    data.o061_out = limit( 1 if ( (data.s060 == 1) and (shift_reg.s060 [0] == 0) and (data.DoorLock
== 0) ) else\
        0, 0, 1 )
    if (avail_marking.p009 >= 1 and data.o029_out != 0) :
        transition_fired.t010 = bool ( 1)
        avail_marking.p009 = avail_marking.p009 - 1
        new_marking.p011 = new_marking.p011 + 1

    if (avail_marking.p011 >= 1 and data.o061_out != 0) :
        transition_fired.t012 = bool ( 1)
        avail_marking.p011 = avail_marking.p011 - 1
        new_marking.p009 = new_marking.p009 + 1

    shiftRegister( shift_reg.COD1, data.COD1)
    shiftRegister( shift_reg.COD2, data.COD2)
    shiftRegister( shift_reg.COD3, data.COD3)
    shiftRegister( shift_reg.COD4, data.COD4)
    shiftRegister( shift_reg.Counter, data.Counter)
    shiftRegister( shift_reg.c1_Memory1, data.c1_Memory1)
    shiftRegister( shift_reg.c1_Memory10, data.c1_Memory10)
    shiftRegister( shift_reg.c1_Memory11, data.c1_Memory11)
    shiftRegister( shift_reg.c1_Memory12, data.c1_Memory12)
    shiftRegister( shift_reg.c1_Memory13, data.c1_Memory13)
    shiftRegister( shift_reg.c1_Memory14, data.c1_Memory14)
    shiftRegister( shift_reg.c1_Memory15, data.c1_Memory15)
    shiftRegister( shift_reg.c1_Memory16, data.c1_Memory16)
    shiftRegister( shift_reg.c1_Memory17, data.c1_Memory17)
    shiftRegister( shift_reg.c1_Memory2, data.c1_Memory2)
    shiftRegister( shift_reg.c1_Memory3, data.c1_Memory3)
    shiftRegister( shift_reg.c1_Memory4, data.c1_Memory4)
    shiftRegister( shift_reg.c1_Memory5, data.c1_Memory5)
    shiftRegister( shift_reg.c1_Memory6, data.c1_Memory6)
    shiftRegister( shift_reg.c1_Memory7, data.c1_Memory7)
    shiftRegister( shift_reg.c1_Memory8, data.c1_Memory8)
    shiftRegister( shift_reg.c1_Memory9, data.c1_Memory9)
    shiftRegister( shift_reg.c1_c1_Key, data.c1_c1_Key)
    shiftRegister( shift_reg.c1_c1_SelKey, data.c1_c1_SelKey)
    shiftRegister( shift_reg.o029_out, data.o029_out)
    shiftRegister( shift_reg.s060, data.s060)
    marking.c1_c1_p001 = avail_marking.c1_c1_p001 + new_marking.c1_c1_p001
    marking.c1_c1_p003 = avail_marking.c1_c1_p003 + new_marking.c1_c1_p003
    marking.c1_c1_p005 = avail_marking.c1_c1_p005 + new_marking.c1_c1_p005
    marking.c1_c1_p012 = avail_marking.c1_c1_p012 + new_marking.c1_c1_p012
    marking.c1_c1_p013 = avail_marking.c1_c1_p013 + new_marking.c1_c1_p013
    marking.c1_c1_p021 = avail_marking.c1_c1_p021 + new_marking.c1_c1_p021
    marking.c1_c1_p022 = avail_marking.c1_c1_p022 + new_marking.c1_c1_p022
    marking.p009 = avail_marking.p009 + new_marking.p009
    marking.p011 = avail_marking.p011 + new_marking.p011
    return


def main():
    init()
    setup_io()
    while( 1 ):
```

```
        readInputs()
        executionStep()
        writeOutputs()
        delayPause()


data = Model_data()
marking = Model_marking()
new_marking = Model_new_marking()
avail_marking = Model_avail_marking()
transition_fired = Model_transition_fired()
shift_reg = Model_shift_reg()
main()
```

Anexo IV

Biblioteca de componentes externos “external_componenets.py”.

```
import smbus # incluido na instalação padrão do Raspbian. Biblioteca de comunicação de dados usando protocolo I2C para registos do PCA9685
import time

# ver endereços no ficheiro PCA96_Datasheet, capitulo 7.3 Register Definitions ( página 9 do PDF)

#Definição dos parâmetros da placa
BOARD_I2C_ADDR = 0x40 # Endereço exclusivo para I2C ( barramento de comunicação serie síncrono (master/slave))
CHANNEL_0_START = 0x06  # Led 0 On Low -> Motor 1
CHANNEL_0_END = 0x08 # Led 0 Off Low  -> Motor 1
CHANNEL_1_START = 0x0A # Led 1 On Low -> Motor 2
CHANNEL_1_END = 0x0C # Led 1 Off Low -> Motor 2
CHANNEL_2_START = 0x0E # Led 2 On Low -> Motor 3
CHANNEL_2_END = 0x10 # Led 2 Off Low -> Motor 3
CHANNEL_3_START = 0x12 # Led 3 On Low -> Motor 4
CHANNEL_3_END = 0x14 # Led 3 Off Low -> Motor 4
MODE1_REG_ADDR = 0 # Mode Register 1 ( 0x00)
PRE_SCALE_REG_ADDR = 0xFE # Registo de Prescaler para a saida da frequência PWM

# Inicialização do PWM, dos parâmetros da placa PCA9685 e SMBus
#Função init é chamada apenas 1 vez. No momento em que o motor começa a correr. Todas as inicializações pertencem a esta função
def local_pwm_servo_4mtr_XML_init( self ):
   print( "Init" )
   self.bus = smbus.SMBus(1)
   #Enable prescaler change ( registo de clock do prescaler)
   self.bus.write_byte_data(BOARD_I2C_ADDR, MODE1_REG_ADDR, 0x10)
   # Set prescaler to 50Hz from datasheet calculation for a 25MHZ clock oscilator ( 25MHZ / (4096 x 50 Hz)) - 1 = 122 (0 x 7A)
   self.bus.write_byte_data(BOARD_I2C_ADDR, PRE_SCALE_REG_ADDR, 0x7A) # definição da frequência
   time.sleep(.25)
   # Enable word writes

   # Set channel start times              address
   self.bus.write_word_data(BOARD_I2C_ADDR, CHANNEL_0_START, 10)  # tempo de inicio de pulso de cada servo é 10
   self.bus.write_word_data(BOARD_I2C_ADDR, CHANNEL_1_START, 10)
   self.bus.write_word_data(BOARD_I2C_ADDR, CHANNEL_2_START, 10)
   self.bus.write_word_data(BOARD_I2C_ADDR, CHANNEL_3_START, 10)

#Código que executa a conversão dos dados em graus para valores
#Função Step é chamada sempre em cada novo passo de execução
def local_pwm_servo_4mtr_XML_step( self, repeat ):

   self.out_Error = 0

   if self.in_Go != 0 :

      pwm_dc = int(306 - 204 * self.in_Mtr1 / 90)
      print("pwm_dc111:",pwm_dc)
      self.bus.write_word_data(BOARD_I2C_ADDR,CHANNEL_0_END,pwm_dc)

      pwm_dc = int(306 - 204 * self.in_Mtr2 / 90)
      print("pwm_dc222:",pwm_dc)
      self.bus.write_word_data(BOARD_I2C_ADDR, CHANNEL_1_END, pwm_dc)

      pwm_dc = int(306 - 204 * self.in_Mtr3 / 90)
      print("pwm_dc333:",pwm_dc)
      self.bus.write_word_data(BOARD_I2C_ADDR, CHANNEL_2_END, pwm_dc)

      pwm_dc = int(306 - 204 * self.in_Mtr4 / 90)
      print("pwm_dc444:",pwm_dc)
```

self.bus.write_word_data(BOARD_I2C_ADDR, CHANNEL_3_END,pwm_dc)

Anexo V

Código Python do modelo da Figura 47.

```
##############################################
 ## Model:test_servos                       ##
 ## IOPT-Flow Python Automatic code Generator ##
 ## 2022 (C) Pedro Vale                     ##
 ##############################################


from Input_Output_raspberrypi import pinMode,digitalwrite,digitalread
from time import sleep

from external_components import
local_pwm_servo_4mtr_xml_init,local_pwm_servo_4mtr_xml_step

class Model_data:
    Grip= 0          #input signal
    M1_CCW= 0   #input signal
    M1_CW= 0     #input signal
    M2_CCW= 0   #input signal
    M2_CW= 0     #input signal
    M3_CCW= 0   #input signal
    M3_CW= 0     #input signal
    release= 0       #input signal
    Pos1= -80        #internal signal
    Pos2= -80        #internal signal
    Pos3= -80        #internal signal
    gripper= -80     #internal signal
    Error= 0         #output signal
    c001_Go= 0     #output component/input
    c001_Mtr1= -90           #output component/input
    c001_Mtr2= -90           #output component/input
    c001_Mtr3= -90           #output component/input
    c001_Mtr4= -90           #output component/input
    c001_Error= 0 #input component/output
    o001_out= -80 #operation/output
    o002_out= -80 #operation/output
    o003_out= -80 #operation/output
    o010_out= -80 #operation/output
    o022_out= 0     #operation/output
    o027_out= -90 #operation/output
    o033_out= -80 #operation/output

class Model_marking:
    pass

class Model_new_marking:
    pass

class Model_avail_marking:
    pass

class Model_transition_fired:
    pass

class Model_shift_reg:
    Pos1=  []
    Pos2=  []
    Pos3=  []
    gripper=  []


class local_pwm_servo_4mtr_xml:
    in_Go = 0
    in_Mtr1 = 0
    in_Mtr2 = 0
    in_Mtr3 = 0
```

```
    in_Mtr4 = 0
    out_Error = 0
    param_string = ""
    res_location = ""
    comment = ""
    init = local_pwm_servo_4mtr_xml_init
    step = local_pwm_servo_4mtr_xml_step
    def __init__(self, ps, rl, c):
        self.param_string = ps
        self.res_location = rl
        self.comment = c


class Components :
    c001 = local_pwm_servo_4mtr_xml("","","")



def initShiftRegister(depth,value):
    reg =  []
    i = 0
    for i in range(depth):
        reg.append(value)
    return reg

def shiftRegister(reg,value):
    reg.pop()
    reg.insert(0, value)



def limit(value,min,max):
    if (value < min):
        return  min
    elif (value > max):
        return max
    else:
        return value


def delayPause():
    sleep(0.001)


def setup_io():
    pinMode("7;PD",1)       #Pino: input
    pinMode("15;PD",1)      #Pino: input
    pinMode("14;PD",1)      #Pino: input
    pinMode("23;PD",1)      #Pino: input
    pinMode("18;PD",1)      #Pino: input
    pinMode("25;PD",1)      #Pino: input
    pinMode("24;PD",1)      #Pino: input
    pinMode("8;PD",1)       #Pino: input
    pinMode("23",0)         #Pino: output



def init():
    data.Error = bool ( 0)
    data.Grip = bool ( 0)
    data.M1_CCW = bool ( 0)
    data.M1_CW = bool ( 0)
    data.M2_CCW = bool ( 0)
    data.M2_CW = bool ( 0)
    data.M3_CCW = bool ( 0)
```

```
    data.M3_CW = bool ( 0)
    data.Pos1 = 0
    data.Pos2 = 34
    data.Pos3 = 16
    data.gripper = -40
    data.release = bool ( 0)
    shift_reg.Pos1 = initShiftRegister( 1, 0)
    shift_reg.Pos2 = initShiftRegister( 1, 34)
    shift_reg.Pos3 = initShiftRegister( 1, 16)
    shift_reg.gripper = initShiftRegister( 1, -40)
    components.c001.init()
    return


def readInputs():
    data.Grip = digitalread("7;PD")
    data.M1_CW = digitalread("14;PD")
    data.M1_CCW = digitalread("15;PD")
    data.M2_CW = digitalread("18;PD")
    data.M2_CCW = digitalread("23;PD")
    data.M3_CW = digitalread("24;PD")
    data.M3_CCW = digitalread("25;PD")
    data.release = digitalread("8;PD")
    return


def writeOutputs():
    digitalwrite("23", data.Error)
    return


def executionStep():
    data.o001_out = limit( shift_reg.gripper [0] + 1 if ( (data.release == 1 and data.Grip == 0) )
else\
        shift_reg.gripper [0] - 1 if ( (data.Grip == 1 and data.release == 0) ) else\
        shift_reg.gripper [0], -80, 80 )
    data.o010_out = limit( shift_reg.Pos1 [0] + 1 if ( (data.M1_CW == 1 and data.M1_CCW == 0)
) else\
        shift_reg.Pos1 [0] - 1 if ( (data.M1_CCW == 1 and data.M1_CW == 0) ) else\
        shift_reg.Pos1 [0], -80, 80 )
    data.o027_out = limit( shift_reg.Pos2 [0] + 1 if ( (data.M2_CW == 1 and data.M2_CCW == 0)
) else\
        shift_reg.Pos2 [0] - 1 if ( (data.M2_CCW == 1 and data.M2_CW == 0) ) else\
        shift_reg.Pos2 [0], -90, 90 )
    data.o033_out = limit( shift_reg.Pos3 [0] + 1 if ( (data.M3_CW == 1 and data.M3_CCW == 0)
) else\
        shift_reg.Pos3 [0] - 1 if ( (data.M3_CCW == 1 and data.M3_CW == 0) ) else\
        shift_reg.Pos3 [0], -80, 80 )
    data.Pos1 = limit( data.o010_out, -80, 80 )
    data.Pos2 = limit( data.o027_out, -80, 80 )
    data.Pos3 = limit( data.o033_out, -80, 80 )
    data.gripper = limit( data.o001_out, -80, 80 )
    data.o002_out = limit( data.Pos2 + data.Pos3 / 2, -80, 80 )
    data.o003_out = limit( data.Pos2 - data.Pos3 / 2, -80, 80 )
    data.o022_out = bool ( data.Pos1 != shift_reg.Pos1 [0] or data.gripper != shift_reg.gripper [0]
or data.Pos2 != shift_reg.Pos2 [0] or data.Pos3 != shift_reg.Pos3 [0])
    data.c001_Go = bool ( data.o022_out)
    data.c001_Mtr1 = limit( data.Pos1, -90, 90 )
    data.c001_Mtr2 = limit( data.o003_out, -90, 90 )
    data.c001_Mtr3 = limit( data.o002_out, -90, 90 )
    data.c001_Mtr4 = limit( data.gripper, -90, 90 )
    components.c001.in_Go = data.c001_Go
    components.c001.in_Mtr1 = data.c001_Mtr1
    components.c001.in_Mtr2 = data.c001_Mtr2
    components.c001.in_Mtr3 = data.c001_Mtr3
```

```
    components.c001.in_Mtr4 = data.c001_Mtr4
    components.c001.step(0)
    data.c001_Error = components.c001.out_Error
    data.Error = bool ( data.c001_Error)
    shiftRegister( shift_reg.Pos1, data.Pos1)
    shiftRegister( shift_reg.Pos2, data.Pos2)
    shiftRegister( shift_reg.Pos3, data.Pos3)
    shiftRegister( shift_reg.gripper, data.gripper)
    return


def main():
    init()
    setup_io()
    while( 1 ):
        readInputs()
        executionStep()
        writeOutputs()
        delayPause()


data = Model_data()
marking = Model_marking()
new_marking = Model_new_marking()
avail_marking = Model_avail_marking()
transition_fired = Model_transition_fired()
components = Components()
shift_reg = Model_shift_reg()
main()
```

Anexo VI

Código Python do modelo da Figura 48.

```
##############################################
 ## Model: pick_place_9        ##
 ## IOPT-Flow Python Automatic code Generator ##
 ## 2022 (C) Pedro Vale                  ##
 ##############################################


from Input_Output_raspberrypi import pinMode,digitalwrite,digitalread
from time import sleep

from external_components import
local_pwm_servo_4mtr_xml_init,local_pwm_servo_4mtr_xml_step

class Model_data:
    Start= 0          #input signal
    Pos2= -80         #internal signal
    Pos3= -80         #internal signal
    Pos4= -80         #internal signal
    c001_CPos1= -90         #internal signal
    c001_CPos2= -90         #internal signal
    c001_CPos3= -90         #internal signal
    c001_CPos4= -90         #internal signal
    c001_Dist= 0    #internal signal
    c001_Go= 0      #internal signal
    c001_Speed= 1           #internal signal
    c001_TPos1= -90         #internal signal
    c001_TPos2= -90         #internal signal
    c001_TPos3= -90         #internal signal
    c001_TPos4= -90         #internal signal
    c001_c1_I1= -32768      #internal signal
    c001_c1_I2= -32768      #internal signal
    c001_c1_I3= -32768      #internal signal
    c001_c1_I4= -32768      #internal signal
    c001_c1_O1= -32768      #internal signal
    c001_c1_O2= -32768      #internal signal
    c001_c1_O3= -32768      #internal signal
    c001_c1_O4= -32768      #internal signal
    c001_h= 0       #internal signal
    pos1= -90       #internal signal
    Error= 0        #output signal
    c001_Done= 0 #internal event
    c001_MoveTo= 0          #internal event
    c001_c1_Store= 0        #internal event
    c1_Go= 0        #output component/input
    c1_Mtr1= -90    #output component/input
    c1_Mtr2= -90    #output component/input
    c1_Mtr3= -90    #output component/input
    c1_Mtr4= -90    #output component/input
    c1_Error= 0     #input component/output
    c001_c1_o010_out= -32768        #operation/output
    c001_c1_o015_out= -32768        #operation/output
    c001_c1_o020_out= -32768        #operation/output
    c001_c1_o025_out= -32768        #operation/output
    c001_o001_out= -32768           #operation/output
    c001_o002_out= 0        #operation/output
    c001_o007_out= -32768           #operation/output
    c001_o010_out= -32768           #operation/output
    c001_o011_out= 0        #operation/output
    c001_o013_out= -32768           #operation/output
    c001_o018_out= 0        #operation/output
    c001_o028_out= 0        #operation/output
    c001_o034_out= -90      #operation/output
    c001_o037_out= 0        #operation/output
    c001_o046_out= -90      #operation/output
    c001_o054_out= -90      #operation/output
```

```
    c001_o062_out= -90      #operation/output
    o002_out= 0     #operation/output
    o009_K= 50      #operation/output
    o014_out= -90 #operation/output
    o020_out= -80 #operation/output
    o066_out= 0     #operation/output
    o087_out= -80 #operation/output
    o098_out= -80 #operation/output

class Model_marking:
    c001_p011= 0
    c001_p015= 0
    c001_p019= 0
    p011= 0
    p012= 0
    p013= 0
    p014= 0
    p032= 0
    p040= 0
    p042= 0
    p044= 0
    p046= 0
    p048= 0
    p050= 0
    p074= 0
    p076= 0
    p098= 0
    p100= 0
    p115= 0
    p117= 0
    p119= 0
    p121= 0


class Model_new_marking:
    c001_p011= 0
    c001_p015= 0
    c001_p019= 0
    p011= 0
    p012= 0
    p013= 0
    p014= 0
    p032= 0
    p040= 0
    p042= 0
    p044= 0
    p046= 0
    p048= 0
    p050= 0
    p074= 0
    p076= 0
    p098= 0
    p100= 0
    p115= 0
    p117= 0
    p119= 0
    p121= 0


class Model_avail_marking:
    c001_p011= 0
    c001_p015= 0
    c001_p019= 0
    p011= 0
    p012= 0
```

```
    p013= 0
    p014= 0
    p032= 0
    p040= 0
    p042= 0
    p044= 0
    p046= 0
    p048= 0
    p050= 0
    p074= 0
    p076= 0
    p098= 0
    p100= 0
    p115= 0
    p117= 0
    p119= 0
    p121= 0


class Model_transition_fired:
    c001_t012= 0
    c001_t017= 0
    c001_t021= 0
    t015= 0
    t019= 0
    t026= 0
    t030= 0
    t034= 0
    t041= 0
    t043= 0
    t045= 0
    t047= 0
    t049= 0
    t051= 0
    t075= 0
    t077= 0
    t099= 0
    t101= 0
    t116= 0
    t118= 0
    t120= 0
    t122= 0


c001_o028_table=  [
    [ 0 ],
    [ 1 ],
    [ 1 ],
    [ 2 ],
    [ 2 ],
    [ 2 ],
    [ 2 ],
    [ 3 ],
    [ 3 ],
    [ 3 ],
    [ 3 ],
    [ 3 ],
    [ 3 ],
    [ 4 ],
    [ 4 ],
    [ 4 ],
    [ 4 ],
    [ 4 ],
    [ 4 ],
    [ 4 ],
```

[4],
[5],
[5],
[5],
[5],
[5],
[5],
[5],
[5],
[5],
[5],
[6],
[6],
[6],
[6],
[6],
[6],
[6],
[6],
[6],
[6],
[6],
[6],
[6],
[7],
[7],
[7],
[7],
[7],
[7],
[7],
[7],
[7],
[7],
[7],
[7],
[7],
[7],
[8],
[8],
[8],
[8],
[8],
[8],
[8],
[8],
[8],
[8],
[8],
[8],
[8],
[8],
[8],
[8],
[8],
[8],
[8],
[9],
[9],
[9],
[9],
[9],
[9],
[9],
[9],
[9],
[9],
[9],
[9],
[9],

[9],
[9],
[9],
[9],
[9],
[9],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[10],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[11],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],
[12],

[12],
[12],
[12],
[12],
[12],
[12],
[12],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[13],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[14],
[15],
[15],
[15],
[15],

[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[15],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[16],
[17],
[17],
[17],
[17],
[17],
[17],
[17],

[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[17],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[18],
[19],
[19],

[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[19],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],

[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[20],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[21],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],

[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[22],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],

[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[23],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[24],
[25],
[25],
[25],
[25],

[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[25],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],

[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[26],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],

[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[27],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],

[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[28],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],
[29],

[29],
[29],
[29],
[29],
[29],
[29],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],
[30],

[30],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[31],
[32],
[32],

[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[32],
[33],
[33],
[33],

[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[33],
[34],
[34],

[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],
[34],

[34],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],
[35],

[35],
[35],
[35],
[35],
[35],
[35],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],

[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[36],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],

[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[37],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],

[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[38],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],

[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[39],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],

[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[40],
[41],
[41],
[41],
[41],

[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],

[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[41],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],

[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[42],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],

[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[43],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],

[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],

[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[44],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],
[45],

```
    [ 45 ],
    [ 45 ],
    [ 45 ],
    [ 45 ],
    [ 45 ],
    [ 45 ],
    [ 45 ],
    [ 45 ],
    [ 45 ],
    [ 45 ],
    [ 45 ],
    [ 45 ],
    [ 45 ]
]
class Model_shift_reg:
    Pos2=  []
    Pos3=  []
    Pos4=  []
    c001_CPos1=  []
    c001_CPos2=  []
    c001_CPos3=  []
    c001_CPos4=  []
    c001_c1_O1=  []
    c001_c1_O2=  []
    c001_c1_O3=  []
    c001_c1_O4=  []
    c001_h=  []
    pos1=  []


class local_pwm_servo_4mtr_xml:
    in_Go = 0
    in_Mtr1 = 0
    in_Mtr2 = 0
    in_Mtr3 = 0
    in_Mtr4 = 0
    out_Error = 0
    param_string = ""
    res_location = ""
    comment = ""
    init = local_pwm_servo_4mtr_xml_init
    step = local_pwm_servo_4mtr_xml_step
    def __init__(self, ps, rl, c):
        self.param_string = ps
        self.res_location = rl
        self.comment = c


class Components :
    c1 = local_pwm_servo_4mtr_xml("","","")


def initShiftRegister(depth,value):
    reg =  []
    i = 0
    for i in range(depth):
        reg.append(value)
    return reg

def shiftRegister(reg,value):
    reg.pop()
    reg.insert(0, value)
```

```
def limit(value,min,max):
    if (value < min):
        return  min
    elif (value > max):
        return max
    else:
        return value


def delayPause():
    sleep(0.001)


def setup_io():
    pinMode("14;PD",1)      #Pino: input


def init():
    data.Error = bool ( 0)
    data.Pos2 = 0
    data.Pos3 = 0
    data.Pos4 = 0
    data.Start = bool ( 0)
    data.c001_CPos1 = 0
    data.c001_CPos2 = 0
    data.c001_CPos3 = 0
    data.c001_CPos4 = 0
    data.c001_Dist = 0
    data.c001_Go = 0
    data.c001_Speed = 1
    data.c001_TPos1 = 0
    data.c001_TPos2 = 0
    data.c001_TPos3 = 0
    data.c001_TPos4 = 0
    data.c001_c1_I1 = 0
    data.c001_c1_I2 = 0
    data.c001_c1_I3 = 0
    data.c001_c1_I4 = 0
    data.c001_c1_O1 = 0
    data.c001_c1_O2 = 0
    data.c001_c1_O3 = 0
    data.c001_c1_O4 = 0
    data.c001_h = 0
    data.pos1 = 0
    data.c001_Done = bool ( 0)
    data.c001_MoveTo = bool ( 0)
    data.c001_c1_Store = bool ( 0)
    marking.c001_p011 = 1
    marking.c001_p015 = 0
    marking.c001_p019 = 0
    marking.p011 = 0
    marking.p012 = 0
    marking.p013 = 0
    marking.p014 = 0
    marking.p032 = 1
    marking.p040 = 0
    marking.p042 = 0
    marking.p044 = 0
    marking.p046 = 0
    marking.p048 = 0
    marking.p050 = 0
    marking.p074 = 0
    marking.p076 = 0
    marking.p098 = 0
    marking.p100 = 0
```

```
    marking.p115 = 0
    marking.p117 = 0
    marking.p119 = 0
    marking.p121 = 0
    shift_reg.Pos2 = initShiftRegister( 1, 0)
    shift_reg.Pos3 = initShiftRegister( 1, 0)
    shift_reg.Pos4 = initShiftRegister( 1, 0)
    shift_reg.c001_CPos1 = initShiftRegister( 1, 0)
    shift_reg.c001_CPos2 = initShiftRegister( 1, 0)
    shift_reg.c001_CPos3 = initShiftRegister( 1, 0)
    shift_reg.c001_CPos4 = initShiftRegister( 1, 0)
    shift_reg.c001_c1_O1 = initShiftRegister( 1, 0)
    shift_reg.c001_c1_O2 = initShiftRegister( 1, 0)
    shift_reg.c001_c1_O3 = initShiftRegister( 1, 0)
    shift_reg.c001_c1_O4 = initShiftRegister( 1, 0)
    shift_reg.c001_h = initShiftRegister( 1, 0)
    shift_reg.pos1 = initShiftRegister( 1, 0)
    components.c1.init()
    return


def readInputs():
    data.Start = digitalread("14;PD")
    return


def writeOutputs():
    return


def executionStep():
    new_marking.c001_p011 = 0
    new_marking.c001_p015 = 0
    new_marking.c001_p019 = 0
    new_marking.p011 = 0
    new_marking.p012 = 0
    new_marking.p013 = 0
    new_marking.p014 = 0
    new_marking.p032 = 0
    new_marking.p040 = 0
    new_marking.p042 = 0
    new_marking.p044 = 0
    new_marking.p046 = 0
    new_marking.p048 = 0
    new_marking.p050 = 0
    new_marking.p074 = 0
    new_marking.p076 = 0
    new_marking.p098 = 0
    new_marking.p100 = 0
    new_marking.p115 = 0
    new_marking.p117 = 0
    new_marking.p119 = 0
    new_marking.p121 = 0
    avail_marking.c001_p011 = marking.c001_p011
    avail_marking.c001_p015 = marking.c001_p015
    avail_marking.c001_p019 = marking.c001_p019
    avail_marking.p011 = marking.p011
    avail_marking.p012 = marking.p012
    avail_marking.p013 = marking.p013
    avail_marking.p014 = marking.p014
    avail_marking.p032 = marking.p032
    avail_marking.p040 = marking.p040
    avail_marking.p042 = marking.p042
    avail_marking.p044 = marking.p044
    avail_marking.p046 = marking.p046
```

```
avail_marking.p048 = marking.p048
avail_marking.p050 = marking.p050
avail_marking.p074 = marking.p074
avail_marking.p076 = marking.p076
avail_marking.p098 = marking.p098
avail_marking.p100 = marking.p100
avail_marking.p115 = marking.p115
avail_marking.p117 = marking.p117
avail_marking.p119 = marking.p119
avail_marking.p121 = marking.p121
transition_fired.c001_t012 = bool ( 0)
transition_fired.c001_t017 = bool ( 0)
transition_fired.c001_t021 = bool ( 0)
transition_fired.t015 = bool ( 0)
transition_fired.t019 = bool ( 0)
transition_fired.t026 = bool ( 0)
transition_fired.t030 = bool ( 0)
transition_fired.t034 = bool ( 0)
transition_fired.t041 = bool ( 0)
transition_fired.t043 = bool ( 0)
transition_fired.t045 = bool ( 0)
transition_fired.t047 = bool ( 0)
transition_fired.t049 = bool ( 0)
transition_fired.t051 = bool ( 0)
transition_fired.t075 = bool ( 0)
transition_fired.t077 = bool ( 0)
transition_fired.t099 = bool ( 0)
transition_fired.t101 = bool ( 0)
transition_fired.t116 = bool ( 0)
transition_fired.t118 = bool ( 0)
transition_fired.t120 = bool ( 0)
transition_fired.t122 = bool ( 0)
data.c001_Go = limit( marking.c001_p019, 0, 255 )
data.c001_o037_out = limit( 0 if ( marking.c001_p015 ) else\
   shift_reg.c001_h [0] + 1 if ( marking.c001_p019 ) else\
   shift_reg.c001_h [0], 0, 14401 )
data.o002_out = bool ( data.c001_Go)
data.o009_K = 50
data.o014_out = limit( 1 if ( (marking.p011) ) else\
   -79 if ( (marking.p013) ) else\
   90 if ( (marking.p119) ) else\
   shift_reg.pos1 [0], -90, 90 )
data.o020_out = limit( -10 if ( (marking.p011) ) else\
   26 if ( (marking.p074) ) else\
   -19 if ( (marking.p115) ) else\
   28 if ( (marking.p048) ) else\
   58 if ( (marking.p040) ) else\
   shift_reg.Pos2 [0], -80, 80 )
data.o087_out = limit( 59 if ( (marking.p011) ) else\
   -8 if ( (marking.p074) ) else\
   61 if ( (marking.p115) ) else\
   -30 if ( (marking.p048) ) else\
   18 if ( (marking.p040) ) else\
   shift_reg.Pos3 [0], -80, 80 )
data.o098_out = limit( -77 if ( (marking.p011) ) else\
   -46 if ( (marking.p098) ) else\
   -77 if ( (marking.p044) ) else\
   shift_reg.Pos4 [0], -80, 80 )
data.Pos2 = limit( data.o020_out, -80, 80 )
data.Pos3 = limit( data.o087_out, -80, 80 )
data.Pos4 = limit( data.o098_out, -80, 80 )
data.c001_Speed = limit( data.o009_K, 1, 100 )
data.c001_h = limit( data.c001_o037_out, 0, 14401 )
data.pos1 = limit( data.o014_out, -90, 90 )
data.c001_TPos1 = limit( data.pos1, -90, 90 )
```

```
    data.c001_TPos2 = limit( data.Pos2, -90, 90 )
    data.c001_TPos3 = limit( data.Pos3, -90, 90 )
    data.c001_TPos4 = limit( data.Pos4, -90, 90 )
    data.c001_o001_out = limit( data.c001_TPos1 - shift_reg.c001_c1_O1 [0], -32768, 32767 )
    data.c001_o007_out = limit( data.c001_TPos2 - shift_reg.c001_c1_O2 [0], -32768, 32767 )
    data.c001_o010_out = limit( data.c001_TPos3 - shift_reg.c001_c1_O3 [0], -32768, 32767 )
    data.c001_o013_out = limit( data.c001_TPos4 - shift_reg.c001_c1_O4 [0], -32768, 32767 )
    data.c001_o018_out = limit( (data.c001_o001_out * data.c001_o001_out +
data.c001_o007_out * data.c001_o007_out + data.c001_o010_out * data.c001_o010_out +
data.c001_o013_out * data.c001_o013_out) / 16, 0, 8100 )
    data.c001_o028_out = limit( 4 * c001_o028_table  [int(limit(data.c001_o018_out-1,0,2047))]
[0], 0, 360 )
    data.c001_o011_out = limit( 40 * data.c001_o028_out / data.c001_Speed, 0, 14400 )
    data.c001_Dist = limit( data.c001_o011_out, 0, 14400 )
    data.c001_o002_out = bool ( data.c001_h >= data.c001_Dist)
    data.c001_o034_out = limit( shift_reg.c001_c1_O1 [0] + data.c001_h * data.c001_o001_out /
data.c001_Dist if ( (marking.c001_p019 and (data.c001_Dist != 0)) ) else\
        shift_reg.c001_CPos1 [0], -90, 90 )
    data.c001_o046_out = limit( shift_reg.c001_c1_O2 [0] + data.c001_h * data.c001_o007_out /
data.c001_Dist if ( (marking.c001_p019 and (data.c001_Dist != 0)) ) else\
        shift_reg.c001_CPos2 [0], -90, 90 )
    data.c001_o054_out = limit( shift_reg.c001_c1_O3 [0] + data.c001_h * data.c001_o010_out /
data.c001_Dist if ( (marking.c001_p019 and (data.c001_Dist != 0)) ) else\
        shift_reg.c001_CPos3 [0], -90, 90 )
    data.c001_o062_out = limit( shift_reg.c001_c1_O4 [0] + data.c001_h * data.c001_o013_out /
data.c001_Dist if ( (marking.c001_p019 and (data.c001_Dist != 0)) ) else\
        shift_reg.c001_CPos4 [0], -90, 90 )
    data.c001_CPos1 = limit( data.c001_o034_out, -90, 90 )
    data.c001_CPos2 = limit( data.c001_o046_out, -90, 90 )
    data.c001_CPos3 = limit( data.c001_o054_out, -90, 90 )
    data.c001_CPos4 = limit( data.c001_o062_out, -90, 90 )
    data.c1_Go = bool ( data.o002_out)
    data.c1_Mtr1 = limit( data.c001_CPos1, -90, 90 )
    data.c1_Mtr2 = limit( data.c001_CPos2, -90, 90 )
    data.c1_Mtr3 = limit( data.c001_CPos3, -90, 90 )
    data.c1_Mtr4 = limit( data.c001_CPos4, -90, 90 )
    components.c1.in_Go = data.c1_Go
    components.c1.in_Mtr1 = data.c1_Mtr1
    components.c1.in_Mtr2 = data.c1_Mtr2
    components.c1.in_Mtr3 = data.c1_Mtr3
    components.c1.in_Mtr4 = data.c1_Mtr4
    components.c1.step(0)
    data.c1_Error = components.c1.out_Error
    data.c001_c1_I1 = limit( data.c001_CPos1, -32768, 32767 )
    data.c001_c1_I2 = limit( data.c001_CPos2, -32768, 32767 )
    data.c001_c1_I3 = limit( data.c001_CPos3, -32768, 32767 )
    data.c001_c1_I4 = limit( data.c001_CPos4, -32768, 32767 )
    data.Error = bool ( data.c1_Error)
    if (avail_marking.c001_p015 >= 1) :
        transition_fired.c001_t017 = bool ( 1)
        avail_marking.c001_p015 = avail_marking.c001_p015 - 1
        new_marking.c001_p019 = new_marking.c001_p019 + 1

    if (avail_marking.c001_p019 >= 1 and data.c001_o002_out != 0) :
        transition_fired.c001_t021 = bool ( 1)
        avail_marking.c001_p019 = avail_marking.c001_p019 - 1
        new_marking.c001_p011 = new_marking.c001_p011 + 1

    if (avail_marking.p011 >= 1) :
        transition_fired.t015 = bool ( 1)
        avail_marking.p011 = avail_marking.p011 - 1
        new_marking.p012 = new_marking.p012 + 1

    if (avail_marking.p013 >= 1) :
        transition_fired.t026 = bool ( 1)
```

```
        avail_marking.p013 = avail_marking.p013 - 1
        new_marking.p014 = new_marking.p014 + 1

    if (avail_marking.p032 >= 1 and data.Start != 0) :
        transition_fired.t034 = bool ( 1)
        avail_marking.p032 = avail_marking.p032 - 1
        new_marking.p011 = new_marking.p011 + 1

    if (avail_marking.p040 >= 1) :
        transition_fired.t041 = bool ( 1)
        avail_marking.p040 = avail_marking.p040 - 1
        new_marking.p042 = new_marking.p042 + 1

    if (avail_marking.p044 >= 1) :
        transition_fired.t045 = bool ( 1)
        avail_marking.p044 = avail_marking.p044 - 1
        new_marking.p046 = new_marking.p046 + 1

    if (avail_marking.p048 >= 1) :
        transition_fired.t049 = bool ( 1)
        avail_marking.p048 = avail_marking.p048 - 1
        new_marking.p050 = new_marking.p050 + 1

    if (avail_marking.p074 >= 1) :
        transition_fired.t075 = bool ( 1)
        avail_marking.p074 = avail_marking.p074 - 1
        new_marking.p076 = new_marking.p076 + 1

    if (avail_marking.p098 >= 1) :
        transition_fired.t099 = bool ( 1)
        avail_marking.p098 = avail_marking.p098 - 1
        new_marking.p100 = new_marking.p100 + 1

    if (avail_marking.p115 >= 1) :
        transition_fired.t116 = bool ( 1)
        avail_marking.p115 = avail_marking.p115 - 1
        new_marking.p117 = new_marking.p117 + 1

    if (avail_marking.p119 >= 1) :
        transition_fired.t120 = bool ( 1)
        avail_marking.p119 = avail_marking.p119 - 1
        new_marking.p121 = new_marking.p121 + 1

    data.c001_Done = bool ( transition_fired.c001_t021)
    data.o066_out = bool ( transition_fired.t015 or transition_fired.t026 or transition_fired.t075 or
transition_fired.t099 or transition_fired.t116 or transition_fired.t120 or transition_fired.t049 or
transition_fired.t045 or transition_fired.t041)
    data.c001_MoveTo = bool ( data.o066_out)
    if (avail_marking.c001_p011 >= 1 and data.c001_MoveTo != 0) :
        transition_fired.c001_t012 = bool ( 1)
        avail_marking.c001_p011 = avail_marking.c001_p011 - 1
        new_marking.c001_p015 = new_marking.c001_p015 + 1

    if (avail_marking.p012 >= 1 and data.c001_Done != 0) :
        transition_fired.t019 = bool ( 1)
        avail_marking.p012 = avail_marking.p012 - 1
        new_marking.p013 = new_marking.p013 + 1

    if (avail_marking.p014 >= 1 and data.c001_Done != 0) :
        transition_fired.t030 = bool ( 1)
        avail_marking.p014 = avail_marking.p014 - 1
        new_marking.p074 = new_marking.p074 + 1

    if (avail_marking.p042 >= 1 and data.c001_Done != 0) :
        transition_fired.t043 = bool ( 1)
```

```
    avail_marking.p042 = avail_marking.p042 - 1
    new_marking.p032 = new_marking.p032 + 1

if (avail_marking.p046 >= 1 and data.c001_Done != 0) :
    transition_fired.t047 = bool ( 1)
    avail_marking.p046 = avail_marking.p046 - 1
    new_marking.p040 = new_marking.p040 + 1

if (avail_marking.p050 >= 1 and data.c001_Done != 0) :
    transition_fired.t051 = bool ( 1)
    avail_marking.p050 = avail_marking.p050 - 1
    new_marking.p044 = new_marking.p044 + 1

if (avail_marking.p076 >= 1 and data.c001_Done != 0) :
    transition_fired.t077 = bool ( 1)
    avail_marking.p076 = avail_marking.p076 - 1
    new_marking.p098 = new_marking.p098 + 1

if (avail_marking.p100 >= 1 and data.c001_Done != 0) :
    transition_fired.t101 = bool ( 1)
    avail_marking.p100 = avail_marking.p100 - 1
    new_marking.p115 = new_marking.p115 + 1

if (avail_marking.p117 >= 1 and data.c001_Done != 0) :
    transition_fired.t118 = bool ( 1)
    avail_marking.p117 = avail_marking.p117 - 1
    new_marking.p119 = new_marking.p119 + 1

if (avail_marking.p121 >= 1 and data.c001_Done != 0) :
    transition_fired.t122 = bool ( 1)
    avail_marking.p121 = avail_marking.p121 - 1
    new_marking.p048 = new_marking.p048 + 1

data.c001_c1_Store = bool ( transition_fired.c001_t012)
data.c001_c1_o010_out = limit( data.c001_c1_I1 if ( (data.c001_c1_Store) ) else\
    shift_reg.c001_c1_O1 [0], -32768, 32767 )
data.c001_c1_o015_out = limit( data.c001_c1_I2 if ( (data.c001_c1_Store) ) else\
    shift_reg.c001_c1_O2 [0], -32768, 32767 )
data.c001_c1_o020_out = limit( data.c001_c1_I3 if ( (data.c001_c1_Store) ) else\
    shift_reg.c001_c1_O3 [0], -32768, 32767 )
data.c001_c1_o025_out = limit( data.c001_c1_I4 if ( (data.c001_c1_Store) ) else\
    shift_reg.c001_c1_O4 [0], -32768, 32767 )
data.c001_c1_O1 = limit( data.c001_c1_o010_out, -32768, 32767 )
data.c001_c1_O2 = limit( data.c001_c1_o015_out, -32768, 32767 )
data.c001_c1_O3 = limit( data.c001_c1_o020_out, -32768, 32767 )
data.c001_c1_O4 = limit( data.c001_c1_o025_out, -32768, 32767 )
shiftRegister( shift_reg.Pos2, data.Pos2)
shiftRegister( shift_reg.Pos3, data.Pos3)
shiftRegister( shift_reg.Pos4, data.Pos4)
shiftRegister( shift_reg.c001_CPos1, data.c001_CPos1)
shiftRegister( shift_reg.c001_CPos2, data.c001_CPos2)
shiftRegister( shift_reg.c001_CPos3, data.c001_CPos3)
shiftRegister( shift_reg.c001_CPos4, data.c001_CPos4)
shiftRegister( shift_reg.c001_c1_O1, data.c001_c1_O1)
shiftRegister( shift_reg.c001_c1_O2, data.c001_c1_O2)
shiftRegister( shift_reg.c001_c1_O3, data.c001_c1_O3)
shiftRegister( shift_reg.c001_c1_O4, data.c001_c1_O4)
shiftRegister( shift_reg.c001_h, data.c001_h)
shiftRegister( shift_reg.pos1, data.pos1)
marking.c001_p011 = avail_marking.c001_p011 + new_marking.c001_p011
marking.c001_p015 = avail_marking.c001_p015 + new_marking.c001_p015
marking.c001_p019 = avail_marking.c001_p019 + new_marking.c001_p019
marking.p011 = avail_marking.p011 + new_marking.p011
marking.p012 = avail_marking.p012 + new_marking.p012
marking.p013 = avail_marking.p013 + new_marking.p013
```

```
    marking.p014 = avail_marking.p014 + new_marking.p014
    marking.p032 = avail_marking.p032 + new_marking.p032
    marking.p040 = avail_marking.p040 + new_marking.p040
    marking.p042 = avail_marking.p042 + new_marking.p042
    marking.p044 = avail_marking.p044 + new_marking.p044
    marking.p046 = avail_marking.p046 + new_marking.p046
    marking.p048 = avail_marking.p048 + new_marking.p048
    marking.p050 = avail_marking.p050 + new_marking.p050
    marking.p074 = avail_marking.p074 + new_marking.p074
    marking.p076 = avail_marking.p076 + new_marking.p076
    marking.p098 = avail_marking.p098 + new_marking.p098
    marking.p100 = avail_marking.p100 + new_marking.p100
    marking.p115 = avail_marking.p115 + new_marking.p115
    marking.p117 = avail_marking.p117 + new_marking.p117
    marking.p119 = avail_marking.p119 + new_marking.p119
    marking.p121 = avail_marking.p121 + new_marking.p121
    return


def main():
    init()
    setup_io()
    while( 1 ):
        readInputs()
        executionStep()
        writeOutputs()
        delayPause()


data = Model_data()
marking = Model_marking()
new_marking = Model_new_marking()
avail_marking = Model_avail_marking()
transition_fired = Model_transition_fired()
components = Components()
shift_reg = Model_shift_reg()
main()
```

Anexo VII

Documento XML com o modelo da Figura 3.

```
<?xml version="1.0"?>
<net name="estacoes_do_ano" type="iopt-flow">
  <place id="p001" x="450" y="360" init_marking="1">
    <name off_x="-10" off_y="-10" text="Inverno"/>
    <comment off_x="0" off_y="20" text="-"/>
  </place>
  <transition id="t002" x="575" y="345" priority="0">
    <name off_x="-10" off_y="-10" text="Frio"/>
    <comment off_x="0" off_y="20" text="-"/>
  </transition>
  <place id="p003" x="435" y="530" init_marking="0">
    <name off_x="-10" off_y="-10" text="Outono"/>
    <comment off_x="0" off_y="20" text="-"/>
  </place>
  <transition id="t004" x="365" y="440" priority="0">
    <name off_x="-10" off_y="-10" text="Chuva"/>
    <comment off_x="0" off_y="20" text="-"/>
  </transition>
  <place id="p005" x="705" y="515" init_marking="0">
    <name off_x="-10" off_y="-10" text="Ver&#xE3;o"/>
    <comment off_x="0" off_y="20" text="-"/>
  </place>
  <transition id="t006" x="600" y="530" priority="0">
    <name off_x="-10" off_y="-10" text="Calor"/>
    <comment off_x="0" off_y="20" text="-"/>
  </transition>
  <place id="p007" x="685" y="355" init_marking="0">
    <name off_x="-10" off_y="-10" text="Primavera"/>
    <comment off_x="0" off_y="20" text="-"/>
  </place>
  <transition id="t008" x="765" y="420" priority="0">
    <name off_x="-10" off_y="-10" text="Ameno"/>
    <comment off_x="0" off_y="20" text="-"/>
  </transition>
  <arc id="a009" type="normal" source="p001" target="t002"/>
  <arc id="a010" type="normal" source="t002" target="p007"/>
  <arc id="a011" type="normal" source="p007" target="t008"/>
  <arc id="a012" type="normal" source="t008" target="p005"/>
  <arc id="a013" type="normal" source="p005" target="t006"/>
  <arc id="a014" type="normal" source="t006" target="p003"/>
  <arc id="a015" type="normal" source="p003" target="t004"/>
  <arc id="a016" type="normal" source="t004" target="p001"/>
</net>
```