MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA

SÉRGIO GABRIEL DOS SANTOS DIAS

ENGENHARIA REVERSA: MODELAGEM DE UMA PLATAFORMA DE MANUFATURA INTEGRADA POR COMPUTADOR

RIO DE JANEIRO
2021

SÉRGIO GABRIEL DOS SANTOS DIAS

ENGENHARIA REVERSA: MODELAGEM DE UMA PLATAFORMA DE MANUFATURA INTEGRADA POR COMPUTADOR

Projeto de Final de Curso apresentado ao Curso de Graduação em Engenharia Eletrônica do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Bacharel em Engenharia Eletrônica.

Orientador(es): Antonio Eduardo Carrilho da Cunha, Dr. Eng

Rio de Janeiro

2021

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 – Praia Vermelha
Rio de Janeiro – RJ CEP: 22290-270

Santos Dias, Sérgio Gabriel dos.
Engenharia Reversa: Modelagem de uma Plataforma de Manufatura Integrada por Computador / Sérgio Gabriel dos Santos Dias. – Rio de Janeiro, 2021.
44 f.

Orientador(es): Antonio Eduardo Carrilho da Cunha.

Projeto de Final de Curso (graduação) – Instituto Militar de Engenharia, Engenharia Eletrônica, 2021.

1. Mecatrônica. 2. Engenharia de Sistemas. 3. Engenharia Reversa. 4. FS100. 5. Scorbase. i. da Cunha, Antonio Eduardo Carrilho (orient.) ii. Título

SÉRGIO GABRIEL DOS SANTOS DIAS

Engenharia Reversa: Modelagem de uma Plataforma de Manufatura Integrada por Computador

Projeto de Final de Curso apresentado ao Curso de Graduação em Engenharia Eletrônica do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Bacharel em Engenharia Eletrônica.

Orientador(es): Antonio Eduardo Carrilho da Cunha.

Aprovado em Rio de Janeiro, 21 de Outubro de 2021, pela seguinte banca examinadora:

Cel. **Antonio Eduardo Carrilho da Cunha** - Dr. Eng

Maj. **Raquel Stella da Silva de Aguiar** - Dr. ISAE

Maj. **Erick Menezes Moreira** - Dr. IME

Rio de Janeiro
2021

Ao Instituto Militar de Engenharia, alicerce da nossa formação e conhecimento.

AGRADECIMENTOS

Os agradecimentos principais são direcionados à minha família e amigos, que juntamente com todos os meus professores me deram o suporte necessário durante esse projeto e todos os meus anos de estudos no IME.

Um agradecimento especial ao orientador Cel Antonio Eduardo Carrilho da Cunha, Maj Raquel Stella da Silva de Aguiar e Maj Erick Menezes Moreira e ao aluno de doutorado Igor Calixto Cohen, por todo o auxílio e disponibilidade.

"Todos podem ver as táticas de minhas conquistas, mas ninguém consegue discernir a estratégia que gerou as vitórias."

Sun Tzu

RESUMO

Com o intuito de modernizar o ensino do Instituto Militar de Engenharia, foi adquirida uma plataforma de Manufatura Integrada por Computador, do inglês CIM - *Computer Integrated Manufacture*, que simula uma linha de produção dentro dos princípio e tecnologias associadas à indústria 4.0.

Além da instalação, para a completa operação do sistema, é necessário o conhecimento e o domínio sobre as linguagens de programação envolvidas, tanto em nivel ScorBase quanto nos Jobs da FS100, e também das relações físicas e computacionais dos componentes.

Nos manuais das linguagens utilizadas no sistema, não estão presentes instruções detalhadas o suficiente que permitam a criação de novos códigos, principalmente com seus testes. Dessa forma, com o presente trabalho, buscou-se aumentar a capacidade de utilização da planta por meio da Engenharia Reversa do código.

ABSTRACT

In order to modernize the teaching in IME, it was acquired a CIM - Computer Integrates Manufacture platform, which simulates a production line within the principles and technologies associated with industry 4.0.

In addition to the installation, a full operation of the system and knowledge of the programming languages involved are required, ranging from ScorBase level to FS100 jobs, and also the relation of the physical and computational components.

In the manuals of the languages used in the system, there are not enough detailed instructions to allow the creation of new codes, mainly with its tests. Thus, with the present work, we sought to increase the plant's utilization capacity through Reverse Engineering of the code.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

CIM	*Computer Integrated Manufacture*, ou Manufatura Integrada por Computador
bdd	*Block Definition Diagram*, ou Diagrama de Definição de Blocos
EAP	Estrutura Analítica do Projeto

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

O laboratório de mecatrônica do IME por meio da platforma MecatrIME possui a capacidade de simular a operação de uma planta industrial de pequeno porte dentro dos conceitos de Indústria 4.0, tanto para software quanto para hardware.

A Quarta Revolução Industrial, também chamada de Indústria 4.0, engloba os conceitos de inteligência artificial, robótica, internet das coisas e computação em nuvem, com o objetivo de melhorar processos e aumentar a produtividade por meio da digitalização das atividades industriais.

A plataforma MecatrIME simula uma planta industrial de produção de pequena escala, com seis estações de trabalho composto por braços robóticos e algumas funções específicas, uma esteira para fazer a movimentação física entre as estações e uma estação de gerenciamento, como visto na Figura 1.

Figura 1 – Organização da planta MecatrIME a partir da estação de gerenciamento

A organização da planta MecatrIME encontra-se detalhada na Figura 2, identificando cada uma das 6 estações de trabalho. Da figura temos P1 como a estação de armazenamento, P2 identifica a estação de torneamento, P3 a estação de usinagem, P4 a estação de soldagem, P5 a estação de metrologia, P6 a estação de montagem e inspeção visual, P7 a instrução de gravação a laser. A estação P8 corresponde ao controle da esteira de transporte e o P9 corresponde ao *Manager* que faz o gerenciamento de todo o sistema.

A complexidade do sistema exige então o conhecimento de conceitos de Engenharia de Sistemas, que representam a organização e a relação entre os componentes físicos e programas computacionais, bem como a relação entre as estações e o gerente.

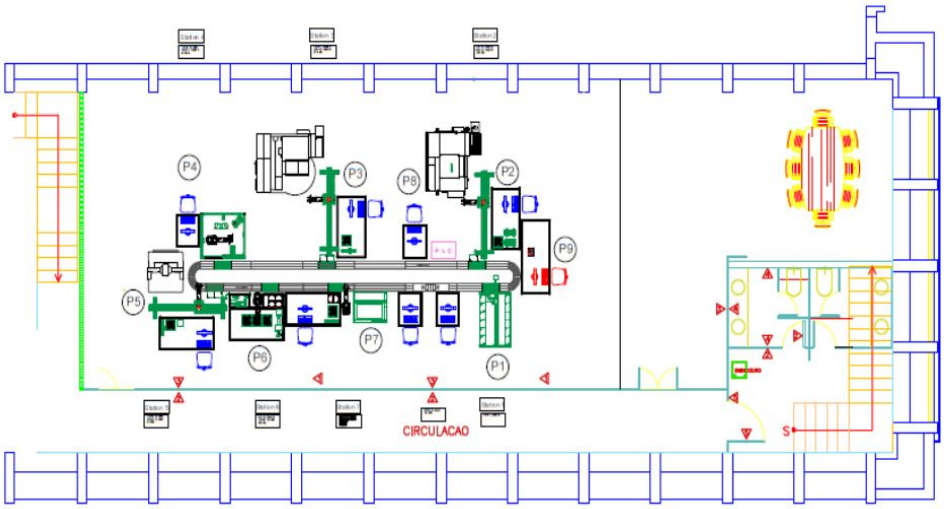

Figura 2 – Organização da planta MecatrIME identificada por estações

A relação entre os objetos físicos atuadores, como a esteira transportadora, os braços robóticos, o robô cartesiano e o torno por exemplo, e as estações virtuais com seus codigos de operação é descrita por meio de um diagrama de blocos conhecido como diagrama cyberfísico.

Dentro de cada de cada estação virtual, o funcionamento das instruções é subdvidido em subrotinas do ScorBase que utiliza jobs do FS100, em mais baixo nível de máquina, com as ordens vindo da estação *Manager*. O conhecimento total do sistema requer o conhecimento das linguagens de programação com o intuito de alterar o código para a exploração completa das funcionalidades e alteração de parâmetros, conhecimento até então não dominado devido à falta de passagem completa do conhecimento por parte da empresa fornecedora.

Por isso, decidiu-se por meio desse projeto e do conhecimento atual da sintaxe e gramática das linguagens de programação construir uma ferramenta que permitisse a análise de códigos, originais ou alterados, para julgamento de seu funcionamento, dessa forma aumentando o domínio da plataforma pelo laboratório, por meio da Engenharia Reversa do código.

1.2 Objetivo

Com base nos requisitos levantados de ferramenta de entrada e saída de autômatos, o escopo do projeto foi:

- Criação de arquivos no formato .dot do *GraphViz* dos scripts em ScorBase do torno;
- Criação dos autômatos dos programas;

- Desenvolvimento de um software que automatize a entrada de script em ScorBase e que tenha como saída os arquivos no formato .dot e figura do autômato correspondente.

1.3 Metodologia

Primeiramente foi definido o escopo do programa que seria utilizado como alvo de pesquisa do projeto, sendo escolhidas as subrotinas do ScorBase relativas ao funcionamento do torno. Dentro dos comandos relativos às execuções dessas ações foram delimitados os casos de início e término, bem como instruções que realizem isntruções diferentes e não lineares.

Após a identificação de cada uma das instruções, com suas possíveis rotas tomadas durante a execução foram traçadas de forma manual, utilizando o programa *yEd*. Esses autômatos gerado no programa serviram como controle sendo um modelo que deveria ser alcançado pela saída do programa em Python.

1.4 Guia a Leitura

O presente texto está organizado seguindo a sequência: Capítulo 1 é uma introdução com a contextualização do projeto, objetivos e metodologia; Capítulo 2 apresenta uma visão geral do problema; O capítulo 3 apresenta a base conceitual do projeto; Capítulo 4 apresenta as ferramentas principais utilizadas no projeto; Capítulo apresenta a solução de tradução de ScorBase para máquina de estados; Capítulo 6 apresenta os resultados dos ensaios com os scripts do escopo e por fim o Capítulo 7 aborda a conclusão do projeto, com as contribuições e possíveis próximos passos do projeto.

2 VISÃO GERAL DO PROBLEMA E ESTRUTURAÇÃO DO PROJETO

2.1 Visão geral do projeto

O projeto visa aumentar a capacidade operacional do laboratório de mecatrônica do IME, por meio da possibilidade de alteração do código com o conhecimento da sua linguagem.

Isso se dá a partir de um software que age como um parser da linguagem ScorBase, recebendo como entrada um script da linguagem e tendo como output um arquivo no formado dot do *GraphViz* e um autômato com as transições de estados representativa do código. O esquema do programa está representado na Figura 3.

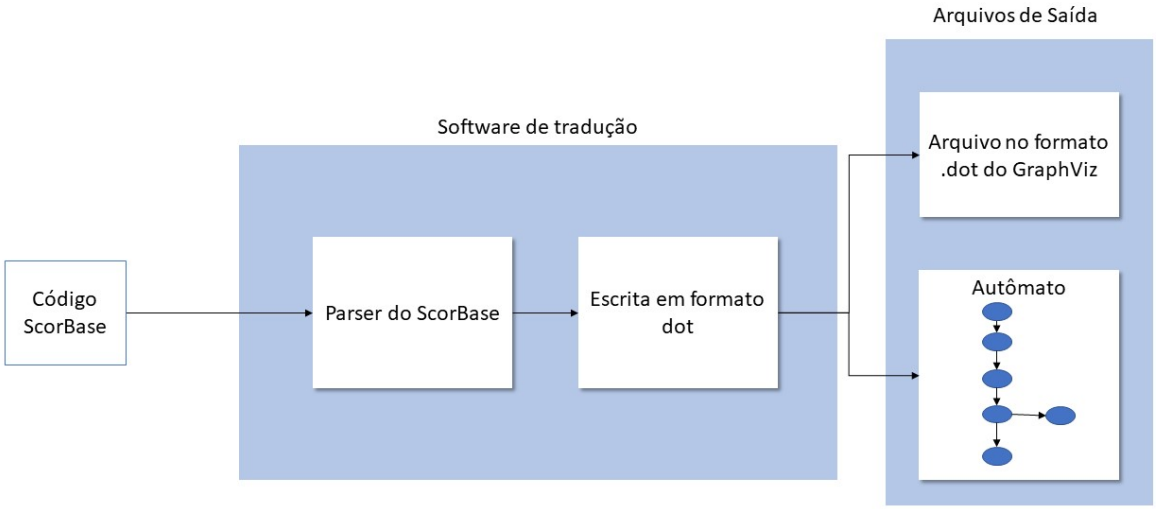

Figura 3 – Diagrama do funcionamento do código.

2.2 Estrutura do Projeto

Com base na demanda e nos prazos, o projeto foi estruturado conforme a Estrtura Analítica do Projeto (EAP) da Figura 4

Com a divisão de frentes do trabalho do projeto, foi possível organizar um cronograma conforme as atividades previstas. Na primeira fase do projeto o foco foi o enten-

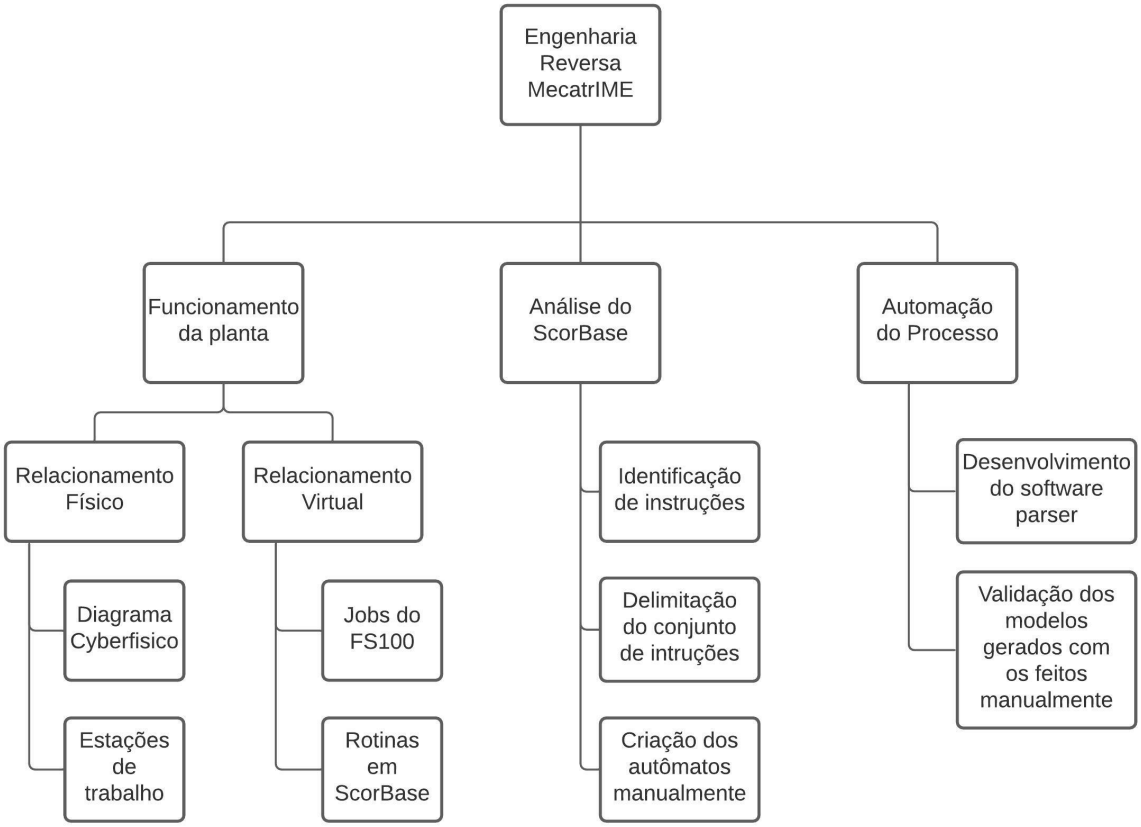

Figura 4 – Estrutura Analítica do Projeto.

dimento do funcionamento do sistema, bem como as tecnologias envolvidas. A segundo etapa foi focada na análise dos scripts e definição do escopo do código a ser trabalhado, juntamente com o desenvolvimento de autômatos de forma manual utilizando o *yEd*. Por fim, foi desenvolvida a ferramenta de tradução de código de ScorBase para dot e criação de autômatos de forma automatizada.

3 BASE CONCEITUAL

3.1 Organização da plataforma MecatrIME

A plataforma da MecatrIME, por ser um sistema complexo, é representado por meio de diagramas de blocos, típicos da Engenharia de Sistema. O sistema é divido em blocos por meio de um diagrama de blocos conhecido como bdd O diagrama cyberfísico de um sistema complexo representa a integração entre as partes computacionais, físicas e de rede. Nesses tipos de sistemas a operação dos elementos físicos influencia na resposta computacional por meio do controle do código e o contrário também é válido.

Para a plataforma MecatrIME a organização em subsistemas de forma visual é organizada conforme a Figura 5. Essa estrutura é a base para o entendimento do relacionamento entre cada uma das partes, bem como a melhor forma de representação visual de cada uma dessas partes.

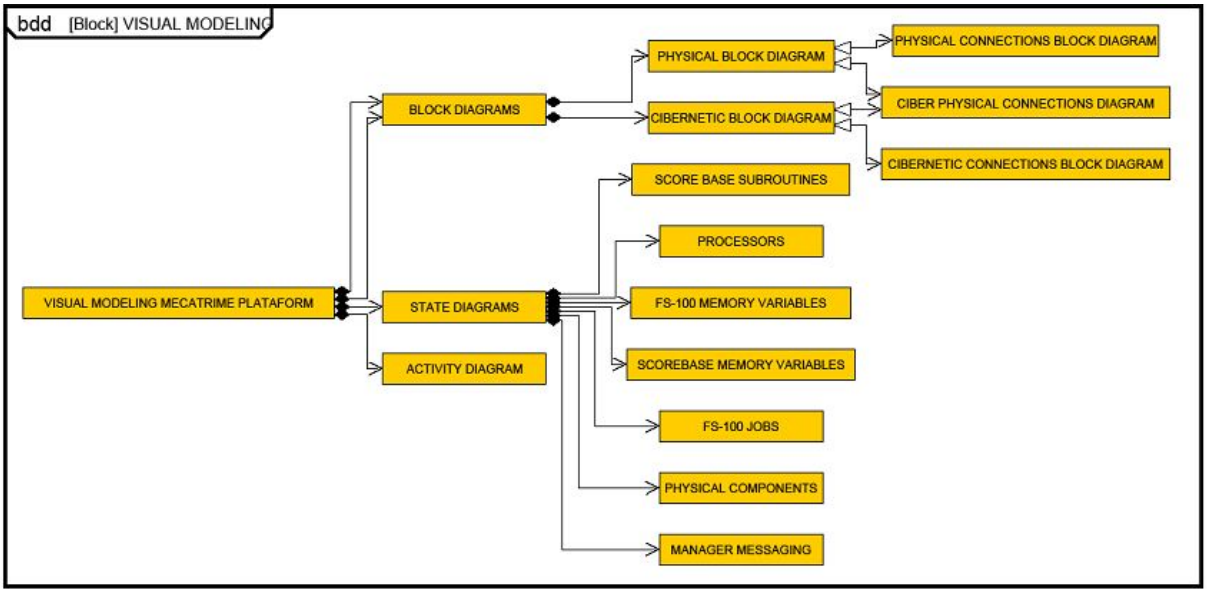

Figura 5 – Diagrama de Blocos da plataforma MecatrIME.

A organização do diagrama cyberfísico é a presente na Figura 6, no qual são identificadas quais estações de trabalho da plataforma são afetadas por rotinas da linguagem ScorBase e como essas estações interagem com o *Manager* e umas com as outras. O esquema de cores identifica as rotinas do ScorBase, em rosa, os jobs da FS100, em azul, e as partes mecânicas do sistema, em amarelo. Por meio do diagrama cyberfísico também fica evidente a relação entre o ScorBase e os jobs da FS100.

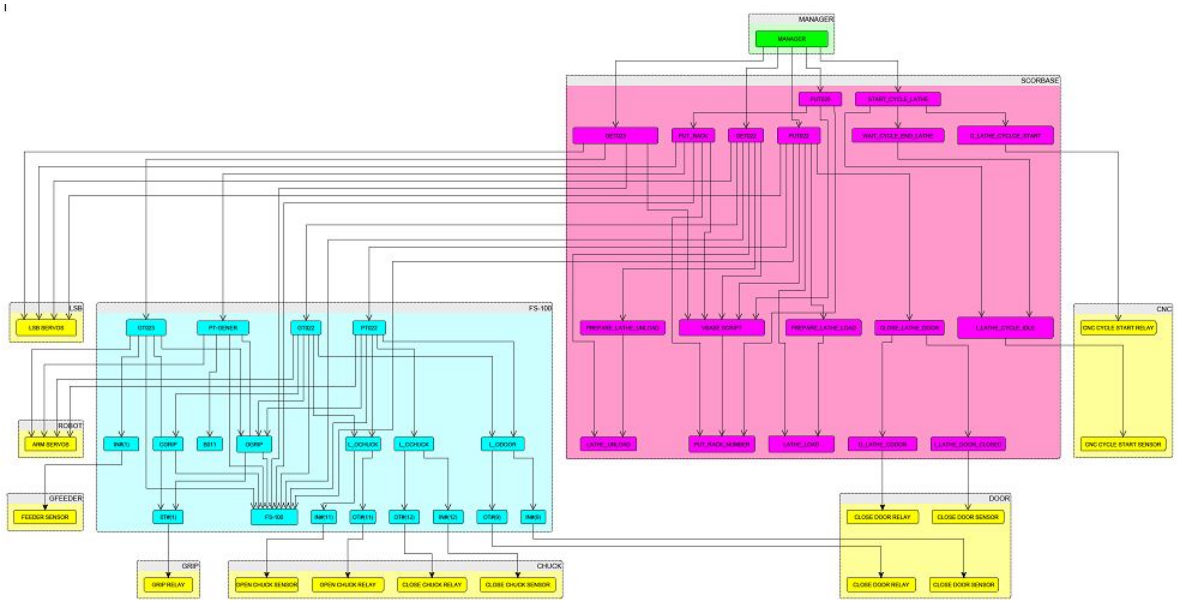

Figura 6 – Diagrama cyberfísico.

3.2 ScorBase e FS100

Dentro da parte computacional o sistema possui um controlador FS100(1) da fabricante Yaskawa Motoman, que possui dentro de suas vantagens um software aberto, permitindo novas implementações e alterações. O controlador representa os jobs em linguagem de mais baixo nível.

A linguagem utilizada em cada uma das estações, em nível mais alto, é a ScorBase da Intelitek(2). Essa linguagem é amplamente utilizada em aplicações de robótica. Juntamente com o FS100 compõem a parte computacional do sistema.

3.3 Autômatos

Na teoria da computação, autômatos são modos abstratos de máquinas que realizam ações baseadas em uma entrada seguindo uma série de estados configurados anteriormente. Em cada estado uma função do código decide qual estado seguinte deve ser tomado, para que no fim o diagrama completo de estados e transições represente a função ou código implementado.

3.4 GraphViz

Para a automatização da criação dos autômatos foi escolhido utilizar o GraphViz, uma ferramenta de construção de gráficos de codigo aberto, e utilizando a linguagem

DOT(3). Essa linguagem utiliza-se de uma gramática abstrata para criação de diagramas de acordo com sua sintaxe.

4 ENSAIOS

Na primeira fase de execução do projeto foram criadas as premissas do trabalho, optando pelo foco nas rotinas do ScorBase. Cada linha de uma rotina ScorBase foi identificada com uma possível operação, sendo dividida em casos especiais, como se segue:

- Chamada: Instruções de *Call* e Run não necessariamente resolvem sua instrução no estágio seguinte, necessitando ter um tratamento diferente;
- Comentários: Linhas do código que devem ser ignoradas na criação do autômato;
- Casos: Instrução de *If* fará com que o código tenha 2 estados possíveis dependendo da validação da condição;
- Espera: Instrução de *Wait* dos Jobs da FS100 podem resultar em *timeouts* encerrando a execução da subrotina.

As instruções foram, então, divididas em comandos de *START* e *FINISH*, e utilizadas como os labels das transições de estados. Dessa forma, o código de uma subrotina do ScorBase *GET023* abaixo:

```
Set Subroutine GET023
Print to Screen: GET FROM Gravity Feeder
Call Subroutine SCRIPT.GET_FROM_GFDR1
Print to Screen: P1,P2, PB1:  'SCRIPT.P1', 'SCRIPT.P2' , 'SCRIPT.PB1'
Print to Screen: P3,P4, P5:  'SCRIPT.P3', 'SCRIPT.P4' , 'SCRIPT.P5'
Copy FS100 Position SCRIPT.P1 to Position 1001
Copy FS100 Position SCRIPT.P2 to Position 1002
Copy FS100 Position SCRIPT.P3 to Position 1003
Copy FS100 Position SCRIPT.P4 to Position 1004
Go to Position SCRIPT.PB1 Speed 50 (%)
FS100 Start Job GT023
Wait FS100_DELAY_TIME (10ths of seconds)
FS100 Job Wait 60 (seconds)
Send Message $Start to MANAGER  ID=TASK_ID
Return from Subroutine
```

Quando traduzida de forma manual para a linguagem DOT do GraphViz, com as transições de estado e considerando os casos especiais temos:

```
digraph GET023 {

A -> B [label = "START Subroutine GET023"];
B -> C [label = "START Call Subroutine SCRIPT.GET_FROM_GFDR1"];
C -> D [label = "FINISH Call Subroutine SCRIPT.GET_FROM_GFDR1"];
D -> E [label = "START Copy FS100 Position SCRIPT.P1 to Position 1001"];
E -> F [label = "FINISH Copy FS100 Position SCRIPT.P1 to Position 1001"];
F -> G [label = "START Copy FS100 Position SCRIPT.P1 to Position 1002"];
G -> H [label = "FINISH Copy FS100 Position SCRIPT.P1 to Position 1002"];
H -> I [label = "START Copy FS100 Position SCRIPT.P1 to Position 1003"];
I -> J [label = "FINISH Copy FS100 Position SCRIPT.P1 to Position 1003"];
J -> K [label = "START Copy FS100 Position SCRIPT.P1 to Position 1004"];
K -> L [label = "FINISH Copy FS100 Position SCRIPT.P1 to Position 1004"];
L -> M [label = "START Go to Position SCRIPT.PB1 Speed 50(%)"];
M -> N [label = "FINISH Go to Position SCRIPT.PB1 Speed 50(%)"];
N -> O [label = "START FS100 Job GET023"];
O -> P [label = "START Wait FS100_DELAY_TIME (10ths of seconds)"];
P -> Q [label = "FINISH Wait FS100_DELAY_TIME (10ths of seconds)"];
Q -> R [label = "TIMEOUT Job GET023"];
Q -> S [label = "FINISH Job Get023"];
S -> T [label = "Send Message $Start to MANAGER ID = TASK_ID"];
T -> U [label = "FINISH Subroutine GET023"];
}
```

E a partir de cada um desses códigos foi gerado um autômato como o da Figura 7. Essa mesma conversão foi feita para cada uma das subrotinas do ScorBase, obtendo um autômato para cada de forma manual, presentes no Anexo A.

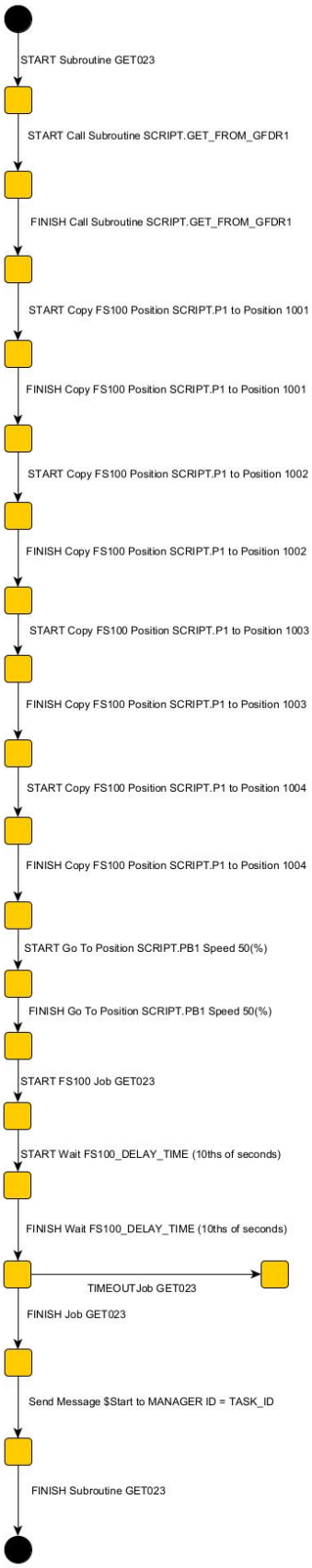

Figura 7 – Autômato da subrotina GET023 do ScorBase.

Outro caso importante presente no código foi a identificação das variáveis de memória do ScorBase, que apresentam uma lógica de autômatos diferente da lógica de uma subrotina. Para as variáveis devem ser previstos quais estados ela pode ser inicializada e quais as alterações podem ser executadas em seu valor. Um exemplo de código DOT do *GraphViz* está abaixo, seguido pelo seu autômato na Figura 8

```
digraph LATHE_LOAD{

A -> B
B -> C
B -> D [label = "READ LATHE_LOAD"];
D -> B [label = "LATHE_LOAD = 0"];
B -> E [label = "SET VARIABLE LATHE_LOAD = 1"];
E -> B [label = "SET VARIABLE LATHE_LOAD = 0"];
E -> F [label = "READ LATHE_LOAD"];
F -> E [label = "LATHE_LOAD = 1"];
E -> G
H -> E
}
```

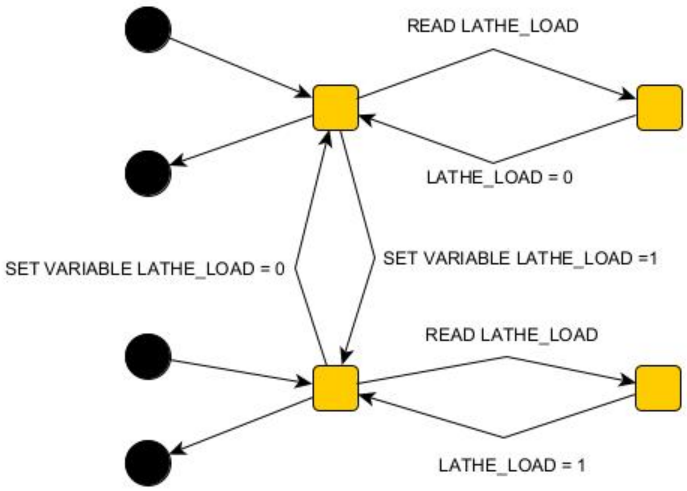

Figura 8 – Autômato da variável de memória LATHE_LOAD.

O próximo passo do projeto foi a construção da ferramenta de automação da criação de códigos DOT e os autômatos em png a partir do script de ScorBase. A linguagem escolhida para isso foi o Python, presente no Anexo B e também no endereço do *GitHub* <https://github.com/S-Santos17069/PFC-2021.git>, resultando em códigos como o que segum, novamente da subrotina GET023:

```
digraph GET023 {
0 -> 1 [label = "START Set Subroutine GET023"];
1 -> 2 [label = "START Print to Screen: GET FROM Gravity Feeder"];
2 -> 3 [label = "FINISH Print to Screen: GET FROM Gravity Feeder"];
3 -> 4 [label = "START Call Subroutine SCRIPT.GET_FROM_GFDR1"];
4 -> 5 [label = "FINISH Call Subroutine SCRIPT.GET_FROM_GFDR1"];
5 -> 6 [label = "START Print to Screen: P1,P2, PB1:  'SCRIPT.P1',
'SCRIPT.P2' , 'SCRIPT.PB1'"];
6 -> 7 [label = "FINISH Print to Screen: P1,P2, PB1:  'SCRIPT.P1',
'SCRIPT.P2' , 'SCRIPT.PB1'"];
7 -> 8 [label = "START Print to Screen: P3,P4, P5:  'SCRIPT.P3',
'SCRIPT.P4' , 'SCRIPT.P5'"];
8 -> 9 [label = "FINISH Print to Screen: P3,P4, P5:  'SCRIPT.P3',
'SCRIPT.P4' , 'SCRIPT.P5'"];
9 -> 10 [label = "START Copy FS100 Position SCRIPT.P1 to Position 1001"];
10 -> 11 [label = "FINISH Copy FS100 Position SCRIPT.P1 to Position 1001"];
11 -> 12 [label = "START Copy FS100 Position SCRIPT.P2 to Position 1002"];
12 -> 13 [label = "FINISH Copy FS100 Position SCRIPT.P2 to Position 1002"];
13 -> 14 [label = "START Copy FS100 Position SCRIPT.P3 to Position 1003"];
14 -> 15 [label = "FINISH Copy FS100 Position SCRIPT.P3 to Position 1003"];
15 -> 16 [label = "START Copy FS100 Position SCRIPT.P4 to Position 1004"];
16 -> 17 [label = "FINISH Copy FS100 Position SCRIPT.P4 to Position 1004"];
17 -> 18 [label = "START Go to Position SCRIPT.PB1 Speed 50 (%)"];
18 -> 19 [label = "FINISH Go to Position SCRIPT.PB1 Speed 50 (%)"];
19 -> 20 [label = "START FS100 Start Job GT023"];
20 -> 21 [label = "START Wait FS100_DELAY_TIME (10ths of seconds)"];
21 -> 22 [label = "FINISH Wait FS100_DELAY_TIME (10ths of seconds)"];
22 -> TIMEOUT [label = "TIMEOUT Job GT023"];
22 -> 23 [label = "FINISH Job GT023"];
23 -> 24 [label = "Send Message $Start to MANAGER  ID=TASK_ID"];
24 -> 25 [label = "FINISH Subroutine GET023
"];
}
```

E o autômato gerado foi o da Figura 9. Todos os códigos DOT e autômatos gerados pelo software estão presentes no Anexo C.

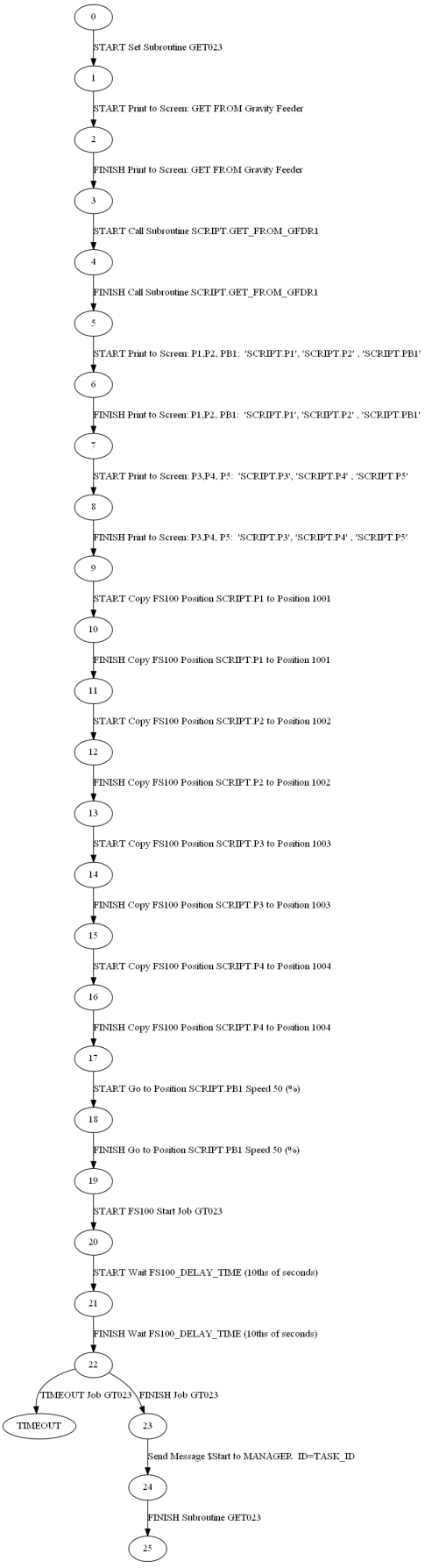

Figura 9 – Autômato da subrotina GET023 do ScorBase gerado pelo software.

Como é perceptível, os autômatos e o código gerado pelo programa automatizado tiveram uma grande confiabilidade quando comparados com os resultados de controle, gerados manualmente. Por isso a ferramenta pode ser considerada confiável dentro do escopo definido, fornecendo a capacidade de testes do código com alterações, devido a velocidade de testes e geração dos autômatos com a utilização da ferramenta, com uma análise posterior para validação dos autômatos gerados como plausíveis e para encontrar possíveis estados mortos, onde o autômato estaria encerrado, terminando o código.

A necessidade de conseguir avaliar códigos alterados antes de executar na plataforma é a possibilidade de encontrar os casos de erros ou de estados mortos, que poderiam danificar o maquinário causando um dano e um custo elevado para manutenção e conserto.

5 CONCLUSÃO

A ferramenta apresentou um grande ganho para o laboratório por permitir o teste de códigos alterados sem a necessidade de criar os autômatos manualmente para cada caso modificado. Essa mudança permite um ganho em tempo pois o código se torna mais eficaz, necessitando somente de uma checagem dos autômatos finais, para o caso de erros ou estados indesejados ou mortos, que encerrariam o programa.

Além disso o uso da ferramenta possibilita um outro ganho que seria a possibilidade de testes de códigos alterando funções e parâmetros das estações de trabalho, sem a necessidade de operação física. Desse modo pode-se reduzir os custos de manutenção bem como diminuir os riscos de falhas e danos, pois os códigos que forem testar nas máquinas já teriam sido previamente válidados e examinados.

5.1 Pontos não abordados

Alguns pontos observados como continuidade do projeto e que permitiriam uma abordagem completa da planta incluem:

- Inclusão de toda a gramática ScorBase;
- Inclusão das variáveis de memória;
- Expansão da ferramenta para operar também com os Jobs da FS100;
- Inclusão dos diagramas cyberfísicos da planta.

REFERÊNCIAS

1 FS100 Controller. 2021. Site oficial da Motoman. Disponível em: <https://www.motoman.com/en-us/products/controllers/fs100>. Acesso em: 13 setembro 2021.

2 SCORBASE. 2021. Site oficial da Intelitek. Disponível em: <https://downloads.intelitek.com/Manuals/Robotics/ER-4u/Scorbase_USB_I.pdf>. Acesso em: 13 setembro 2021.

3 DOCUMENTATION for DOT in GraphViz. 2021. Site oficial do GraphViz. Disponível em: <https://graphviz.org/doc/info/lang.html>. Acesso em: 13 setembro 2021.

ANEXO A – AUTÔMATOS E CÓDIGOS DOT

```
    digraph GET022 {

A -> B [label = "START Subroutine GET022"];
B -> C [label = "START Subroutine PREPARE_LATHE_LOAD"];
C -> D [label = "FINISH Subroutine PREPARE_LATHE_LOAD"];
D -> E [label = "START Subroutine SCRIPT.PUT_TO_LATHE1"];
E -> F [label = "FINISH Subroutine SCRIPT.PUT_TO_LATHE1"];
F -> G [label = "START Go To Position SCRIPT.PB1 Speed 50%"];
G -> H [label = "FINISH Go To Position SCRIPT.PB1 Speed 50%"];
H -> I [label = "START Wait 1(10ths of seconds)"];
I -> J [label = "FINISH Wait 1(10ths of seconds)"];
J -> H [label = "LATHE_LOAD == 0"];
J -> K [label = "NOT LATHE_LOAD == 0"];
K -> L [label = "START Copy FS100 Position SCRIPT.P1 To Position 1001"];
L -> M [label = "FINISH Copy FS100 Position SCRIPT.P1 To Position 1001"];
M -> N [label = "START FS100 Job GET022"];
N -> O [label = "START Wait 10 (10ths of seconds)"];
O -> P [label = "FINISH Wait 10 (10ths of seconds)"];
P -> R [label = "TIMEOUT Job GET022"];
P -> Q [label = "FINISH Wait 10 (10ths of seconds)"];
Q -> S [label = "Send Message $Start to MANAGER_ID = TASK_ID"];
S -> T [label = "FINISH Subroutine GET022"];
}
```

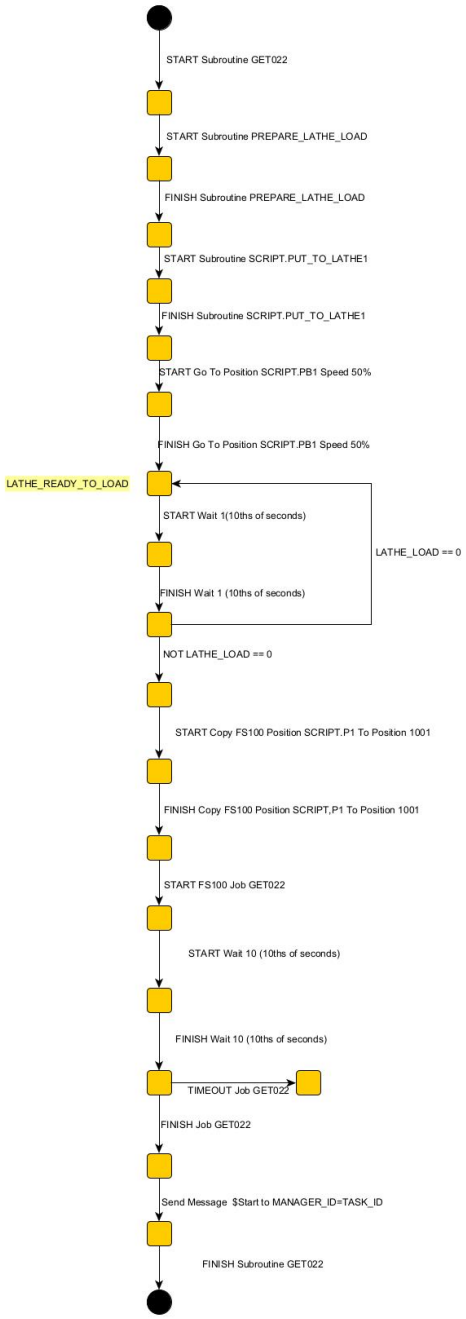

Figura 10 – Autômato da subrotina GET022

```
 digraph PREPARE_LATHE_LOAD {

A -> B [label = "START Subroutine PREPARE_LATHE_LOAD"];
B -> C [label = "SET Variable LATHE_LOAD = 1"];
C -> D [label = "FINISH Subroutine PREPARE_LATHE_LOAD"];

}
```

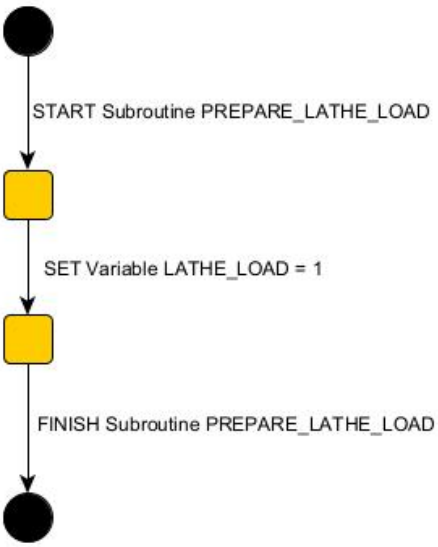

Figura 11 – Autômato da subrotina PREPARE_LATHE_LOAD

```
digraph PREPARE_LATHE_UNLOAD {
A -> B [label = "START Subroutine PREPARE_LATHE_UNLOAD"];
B -> C [label = "SET Variable LATHE_UNLOAD = 1"];
C -> D [label = "FINISH Subroutine PREPARE_LATHE_UNLOAD"];
}
```

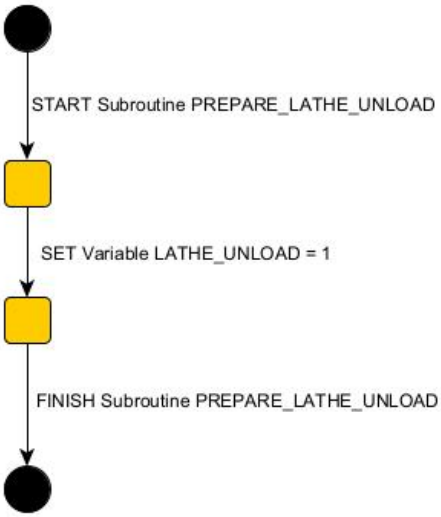

Figura 12 – Autômato da subrotina PREPARE_LATHE_UNLOAD

```
digraph LATHE_LOAD{

A -> B
B -> C
B -> D [label = "READ LATHE_LOAD"];
D -> B [label = "LATHE_LOAD = 0"];
B -> E [label = "SET VARIABLE LATHE_LOAD = 1"];
```

```
E -> B [label = "SET VARIABLE LATHE_LOAD = 0"];
E -> F [label = "READ LATHE_LOAD"];
F -> E [label = "LATHE_LOAD = 1"];
E -> G
H -> E
}
```

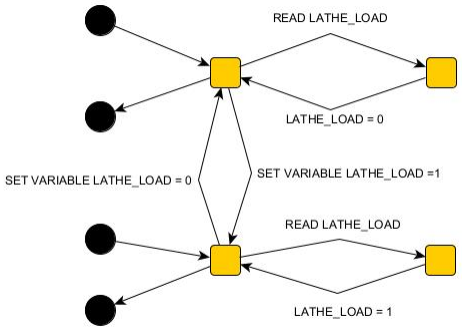

Figura 13 – Autômato da subrotina LATHE_LOAD

```
digraph PUT_RACK {
A -> B [label = "START Subroutine PUT_RACK"];
B -> C [label = "START Subroutine SCRIPT.PUT_TO_RACK(PUT_RACK_NUMBER)"];
C -> D [label = "FINISH Subroutine SCRIPT.PUT_TO_RACK(PUT_RACK_NUMBER)"];
D -> E [label = "START Go to Position SCRIPT.PB1 Speed 50(%)"];
E -> F [label = "FINISH Go to Position SCRIPT.PB1 Speed 50(%)"];
F -> G [label = "START Copy FS100 Position SCRIPT.P1 to Position 1001"];
G -> H [label = "FINISH Copy FS100 Position SCRIPT.P1 to Postion 1001"];
H -> I [label = "START FS100 Job GT022"];
I -> J [label = "START Wait 10 (10ths of seconds)"];
J -> K [label = "FINISH Wait 10 (10ths of seconds)"];
K -> L [label = "FINISH Job GT022"];
K -> M [label = "TIMEOUT Job GT022"];
L -> N [label = "Send MEssage $Start to MANAGER_ID = TASL_ID"];
N -> O [label = "FINISH Subroutine PUT_RACK"];
}
```

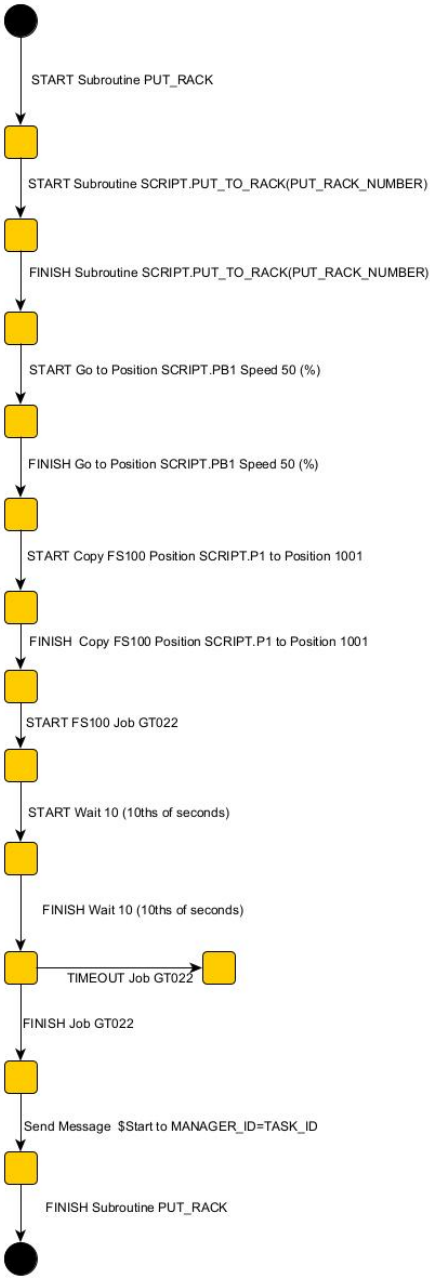

Figura 14 – Autômato da subrotina PUT_RACK

```
digraph PUT025 {
A -> B [label = "START Subroutine PUT025"];
B -> C [label = "START Subroutine SCRIPT.PUT_TO_RACK(1)"];
C -> D [label = "FINISH Subroutine SCRIPT.PUT_TO_RACK(1)"];
D -> E [label = "SET Variable PUT_RACK_NUMBER=1"];
E -> F [label = "START Subroutine PUT_RACK"];
F -> G [label = "FINISH Subroutine PUT_RACK"];
G -> H [label = "FINISH Subroutine PUT025"];
}
```

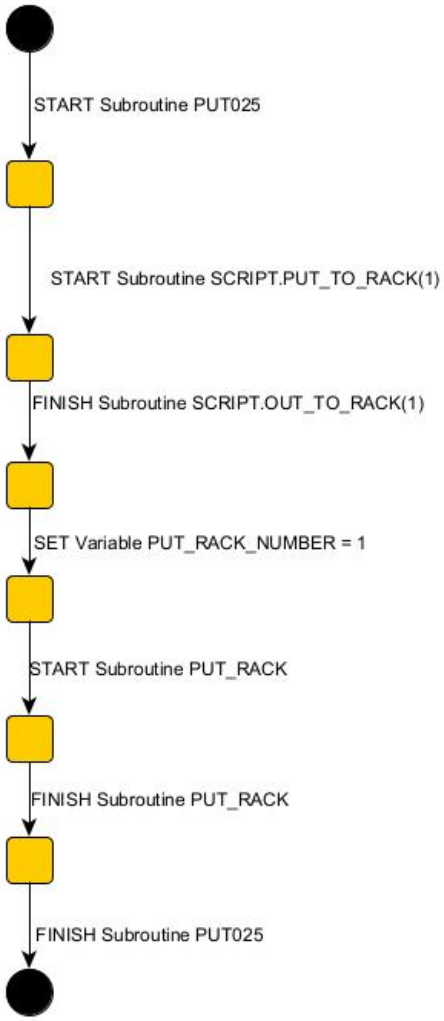

Figura 15 – Autômato da subrotina PUT025

```
   digraph PUT025 {
 0 -> 1 [label = "START Subroutine PUT025"];
 1 -> 2 [label = "START Subroutine SCRIPT.PUT_TO_RACK(1)"];
 2 -> 3 [label = "FINISH Subroutine SCRIPT.PUT_TO_RACK(1)"];
 3 -> 4 [label = "SET Variable PUT_RACK_NUMBER=1"];
 4 -> 5 [label = "START Subroutine PUT_RACK"];
 5 -> 6 [label = "FINISH Subroutine PUT_RACK"];
 6 -> 7 [label = "FINISH Subroutine PUT025"];
}
```

ANEXO B – CÓDIGO PYTHON DA FERRAMENTA

```
    import os

i = 0
x = 1
aux = ""
linha = ""
linha_aux = " "
linha_aux_2 = " "
retorno = 0

nome_do_arquivo = input("nome do arquivo ")

f = open(nome_do_arquivo + ".txt","r")
g = open(nome_do_arquivo + ".dot","w")

g.write("digraph " + nome_do_arquivo +" { \n")

for line in f:
    linha = line
    if linha.find('Run') != -1:
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "START " + linha[:-1] + '"' + "];\n")
        x += 1
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "FINISH " + linha[:-1] + '"' + "];\n")
        x += 1
    elif linha.find('Print') != -1:
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "START " + linha[:-1] + '"' + "];\n")
        x += 1
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "FINISH " + linha[:-1] + '"' + "];\n")
        x += 1
    elif linha.find('Call') != -1:
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "START " + linha[:-1] + '"' + "];\n")
        x += 1
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "FINISH " + linha[:-1] + '"' + "];\n")
        x += 1
    elif linha.find('Copy') != -1:
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "START " + linha[:-1] + '"' + "];\n")
        x += 1
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "FINISH " + linha[:-1] + '"' + "];\n")
        x += 1
    elif linha.find('Go') != -1:
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "START " + linha[:-1] + '"' + "];\n")
        x += 1
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "FINISH " + linha[:-1] + '"' + "];\n")
        x += 1
    elif linha.find('FS100 Start Job') != -1:
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "START " + linha[:-1] + '"' + "];\n")
        x += 1
        linha_aux = line
    elif linha[0:4] == 'Wait':
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "START " + linha[:-1] + '"' + "];\n")
        x += 1
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "FINISH " + linha[:-1] + '"' + "];\n")
        x += 1
    elif linha.find('FS100 Job') != -1:
```

```
        g.write(str(x) + " -> TIMEOUT" + " [label = " + '"' + "TIMEOUT" + linha_aux[11:-1] + '"' + "];\n" )
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "FINISH" + linha_aux[11:-1] + '"' + "];\n")
        x += 1
    elif linha.find('Send') != -1:
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + linha[:-1] + '"' + "];\n")
        x += 1
    elif linha.find('Set') != -1:
        g.write("0 -> 1"  + " [label = " + '"' + "START " + linha[:-1] + '"' + "];\n")
        linha_aux_2 = line
    elif linha.find('Return') != -1:
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"' + "FINISH " + linha_aux_2[4:] + '"' + "];\n")
    elif linha[0:2] == "If":
        g.write(str(x) + " -> " + str(retorno) + " [label = " + '"' + linha[3:-1] + '"' + "];\n")
        g.write(str(x) + " -> " + str(x+1) + " [label = " + '"NOT ' + linha[3:-1] + '"' + "];\n")
        x += 1
    else:
        retorno = x


g.write("}")
f.close()
g.close()

comando = "dot -Tpng " + nome_do_arquivo +".dot -o " + nome_do_arquivo + ".png"
os.system("cmd /k " + comando)
```

ANEXO C – AUTÔMATOS E CÓDIGOS DOT DA FERRAMENTA

```
    digraph GET022 {
0 -> 1 [label = "START Set Subroutine GET022"];
1 -> 2 [label = "START Print to Screen: GET FROM LATHE (CONCEPT 250)"];
2 -> 3 [label = "FINISH Print to Screen: GET FROM LATHE (CONCEPT 250)"];
3 -> 4 [label = "START Run Subroutine PREPARE_LATHE_UNLOAD"];
4 -> 5 [label = "FINISH Run Subroutine PREPARE_LATHE_UNLOAD"];
5 -> 6 [label = "START Call Subroutine SCRIPT.GET_FROM_LATHE1"];
6 -> 7 [label = "FINISH Call Subroutine SCRIPT.GET_FROM_LATHE1"];
7 -> 8 [label = "START Print to Screen: P1,P2,P3, P4:  'SCRIPT.P1', 'SCRIPT.P2' , 'SCRIPT.P3' ,'SCRIPT.P4'"];
8 -> 9 [label = "FINISH Print to Screen: P1,P2,P3, P4:  'SCRIPT.P1', 'SCRIPT.P2' , 'SCRIPT.P3' ,'SCRIPT.P4'"];
9 -> 10 [label = "START Print to Screen: PB1,PB2:  'SCRIPT.PB1', 'SCRIPT.PB2'"];
10 -> 11 [label = "FINISH Print to Screen: PB1,PB2:  'SCRIPT.PB1', 'SCRIPT.PB2'"];
11 -> 12 [label = "START Go to Position SCRIPT.PB1 Speed 50 (%)"];
12 -> 13 [label = "FINISH Go to Position SCRIPT.PB1 Speed 50 (%)"];
13 -> 14 [label = "START Wait 1 (10ths of seconds)"];
14 -> 15 [label = "FINISH Wait 1 (10ths of seconds)"];
15 -> 13 [label = "LATHE_UNLOAD ==0 Jump to LATHE_READY_TO_UNLOAD"];
15 -> 16 [label = "NOT LATHE_UNLOAD ==0 Jump to LATHE_READY_TO_UNLOAD"];
16 -> 17 [label = "START Copy FS100 Position SCRIPT.P1 to Position 1001"];
17 -> 18 [label = "FINISH Copy FS100 Position SCRIPT.P1 to Position 1001"];
18 -> 19 [label = "START Copy FS100 Position SCRIPT.P2 to Position 1002"];
19 -> 20 [label = "FINISH Copy FS100 Position SCRIPT.P2 to Position 1002"];
20 -> 21 [label = "START Copy FS100 Position SCRIPT.P3 to Position 1003"];
21 -> 22 [label = "FINISH Copy FS100 Position SCRIPT.P3 to Position 1003"];
22 -> 23 [label = "START Copy FS100 Position SCRIPT.P4 to Position 1004"];
23 -> 24 [label = "FINISH Copy FS100 Position SCRIPT.P4 to Position 1004"];
24 -> 25 [label = "START FS100 Start Job GT022"];
25 -> 26 [label = "START Wait 10 (10ths of seconds)"];
26 -> 27 [label = "FINISH Wait 10 (10ths of seconds)"];
27 -> TIMEOUT [label = "TIMEOUT Job GT022"];
27 -> 28 [label = "FINISH Job GT022"];
28 -> 29 [label = "Send Message $Start to MANAGER  ID=TASK_ID"];
29 -> 30 [label = "FINISH Subroutine GET022
"];
}
```

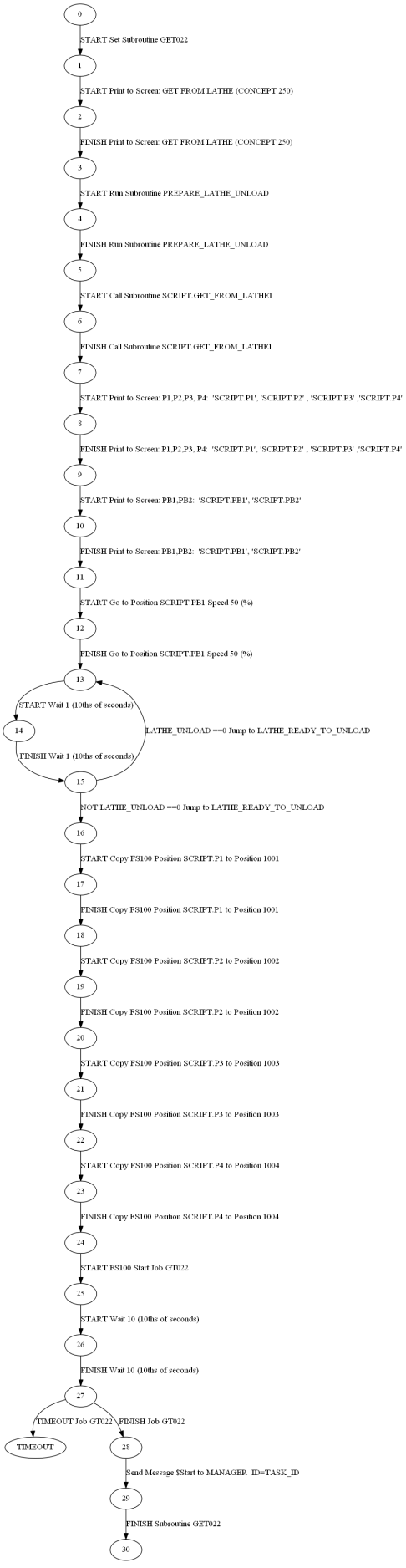

Figura 16 – Autômato da subrotina GET022

```
digraph PUT022 {
0 -> 1 [label = "START Set Subroutine PUT022"];
1 -> 2 [label = "START Print to Screen: Place on Lathe (CONCEPT 250)"];
2 -> 3 [label = "FINISH Print to Screen: Place on Lathe (CONCEPT 250)"];
3 -> 4 [label = "START Run Subroutine PREPARE_LATHE_LOAD"];
4 -> 5 [label = "FINISH Run Subroutine PREPARE_LATHE_LOAD"];
5 -> 6 [label = "START Call Subroutine SCRIPT.PUT_TO_LATHE1"];
6 -> 7 [label = "FINISH Call Subroutine SCRIPT.PUT_TO_LATHE1"];
7 -> 8 [label = "START Print to Screen: P1,P2,P3, P4:  'SCRIPT.P1', 'SCRIPT.P2' , 'SCRIPT.P3' ,'SCRIPT.P4'"];
8 -> 9 [label = "FINISH Print to Screen: P1,P2,P3, P4:  'SCRIPT.P1', 'SCRIPT.P2' , 'SCRIPT.P3' ,'SCRIPT.P4'"];
9 -> 10 [label = "START Print to Screen: PB1,PB2:  'SCRIPT.PB1', 'SCRIPT.PB2'"];
10 -> 11 [label = "FINISH Print to Screen: PB1,PB2:  'SCRIPT.PB1', 'SCRIPT.PB2'"];
11 -> 12 [label = "START Go to Position SCRIPT.PB1 Speed 50 (%)"];
12 -> 13 [label = "FINISH Go to Position SCRIPT.PB1 Speed 50 (%)"];
13 -> 14 [label = "START Wait 1 (10ths of seconds)"];
14 -> 15 [label = "FINISH Wait 1 (10ths of seconds)"];
15 -> 13 [label = "LATHE_LOAD==0 Jump to LATHE_READY_TO_LOAD"];
15 -> 16 [label = "NOT LATHE_LOAD==0 Jump to LATHE_READY_TO_LOAD"];
16 -> 17 [label = "START Copy FS100 Position SCRIPT.P1 to Position 1001"];
17 -> 18 [label = "FINISH Copy FS100 Position SCRIPT.P1 to Position 1001"];
18 -> 19 [label = "START Copy FS100 Position SCRIPT.P2 to Position 1002"];
19 -> 20 [label = "FINISH Copy FS100 Position SCRIPT.P2 to Position 1002"];
20 -> 21 [label = "START Copy FS100 Position SCRIPT.P3 to Position 1003"];
21 -> 22 [label = "FINISH Copy FS100 Position SCRIPT.P3 to Position 1003"];
22 -> 23 [label = "START Copy FS100 Position SCRIPT.P4 to Position 1004"];
23 -> 24 [label = "FINISH Copy FS100 Position SCRIPT.P4 to Position 1004"];
24 -> 25 [label = "START FS100 Start Job PT022"];
25 -> 26 [label = "START Wait 10 (10ths of seconds)"];
26 -> 27 [label = "FINISH Wait 10 (10ths of seconds)"];
27 -> TIMEOUT [label = "TIMEOUT Job PT022"];
27 -> 28 [label = "FINISH Job PT022"];
28 -> 29 [label = "START Call Subroutine CLOSE_LATHE_DOOR"];
29 -> 30 [label = "FINISH Call Subroutine CLOSE_LATHE_DOOR"];
30 -> 31 [label = "Send Message $Finish to MANAGER  ID=TASK_ID"];
31 -> 32 [label = "START Go to Position SCRIPT.PB2 Speed 50 (%)"];
32 -> 33 [label = "FINISH Go to Position SCRIPT.PB2 Speed 50 (%)"];
33 -> 34 [label = "Send Message $End to MANAGER  ID=TASK_ID"];
34 -> 35 [label = "FINISH Subroutine PUT022
"];
}
```

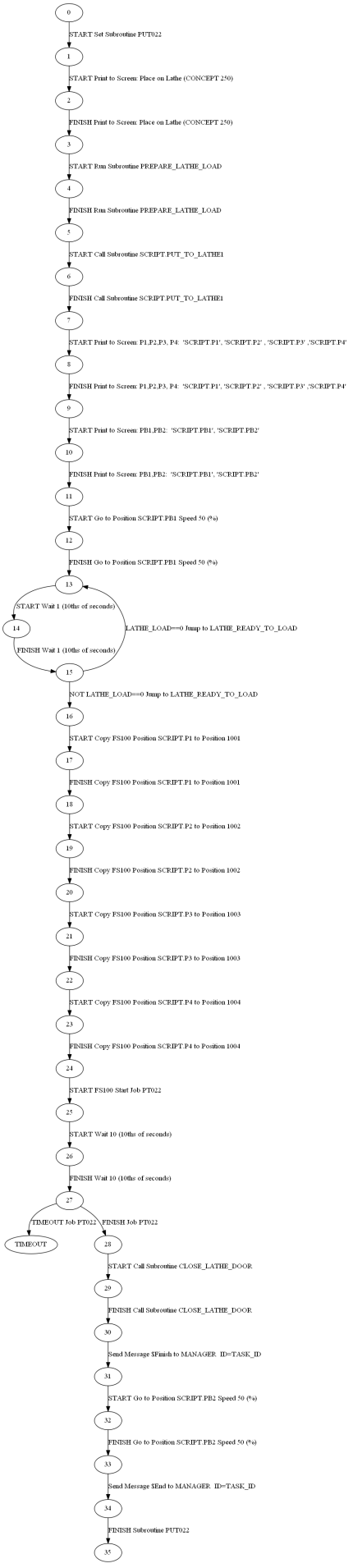

Figura 17 – Autômato da subrotina PUT022

```
digraph START_LATHE_CYCLE {
0 -> 1 [label = "START Set Subroutine START_CYCLE_LATHE"];
1 -> 2 [label = "START Wait 15 (10ths of seconds)"];
2 -> 3 [label = "FINISH Wait 15 (10ths of seconds)"];
3 -> 4 [label = "START Print to Screen: WAIT_CYCLE_STARTED_LATHE"];
4 -> 5 [label = "FINISH Print to Screen: WAIT_CYCLE_STARTED_LATHE"];
5 -> 6 [label = "START Wait Until Digital Input 15 is OFF ****** I_LATHE_CYCLE_IDLE   "];
6 -> 7 [label = "FINISH Wait Until Digital Input 15 is OFF ****** I_LATHE_CYCLE_IDLE   "];
7 -> 8 [label = "START Run Subroutine WAIT_CYCLE_END_LATHE"];
8 -> 9 [label = "FINISH Run Subroutine WAIT_CYCLE_END_LATHE"];
9 -> 10 [label = "FINISH Subroutine START_CYCLE_LATHE
"];
}
```

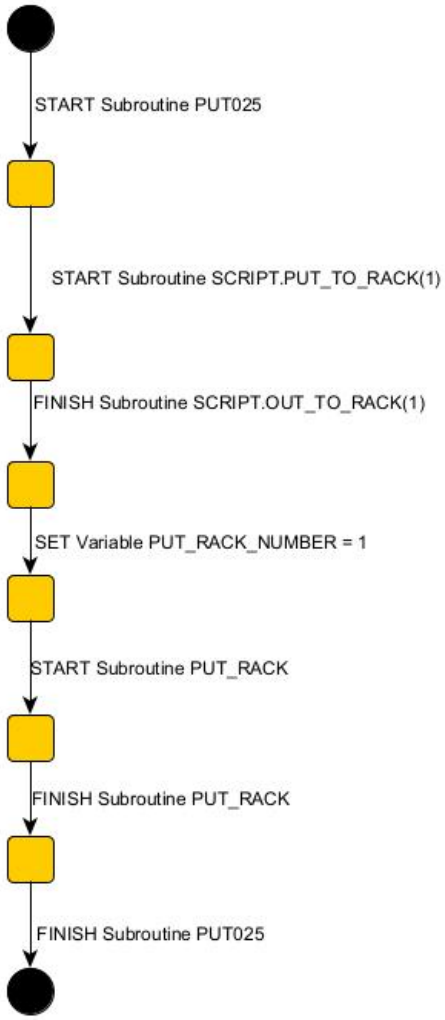

Figura 18 – Autômato da subrotina PUT025

```
    digraph PUT025 {
0 -> 1 [label = "START Subroutine PUT025"];
1 -> 2 [label = "START Subroutine SCRIPT.PUT_TO_RACK(1)"];
2 -> 3 [label = "FINISH Subroutine SCRIPT.PUT_TO_RACK(1)"];
3 -> 4 [label = "SET Variable PUT_RACK_NUMBER=1"];
4 -> 5 [label = "START Subroutine PUT_RACK"];
5 -> 6 [label = "FINISH Subroutine PUT_RACK"];
```

```
6 -> 7 [label = "FINISH Subroutine PUT025"];
}
```

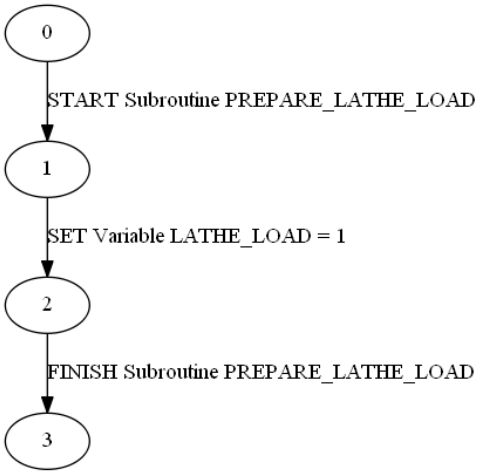

Figura 19 – Autômato da subrotina PREPARE_LATHE_LOAD

```
digraph PREPARE_LATHE_LOAD {

0 -> 1 [label = "START Subroutine PREPARE_LATHE_LOAD"];
1 -> 2 [label = "SET Variable LATHE_LOAD = 1"];
2 -> 3 [label = "FINISH Subroutine PREPARE_LATHE_LOAD"];
}
```

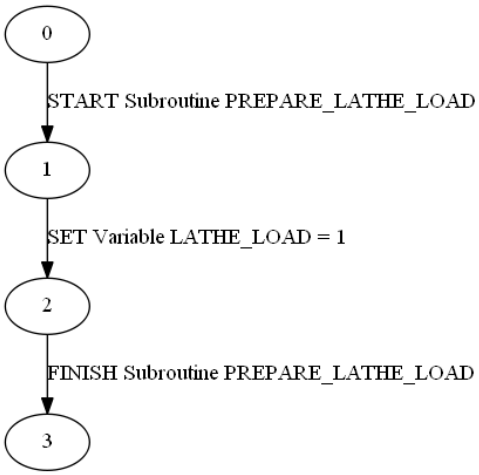

Figura 20 – Autômato da subrotina PREPARE_LATHE_UNLOAD

```
digraph PREPARE_LATHE_UNLOAD {

0 -> 1 [label = "START Subroutine PREPARE_LATHE_UNLOAD"];
1 -> 2 [label = "SET Variable LATHE_UNLOAD = 1"];
2 -> 3 [label = "FINISH Subroutine PREPARE_LATHE_UNLOAD"];
}
```

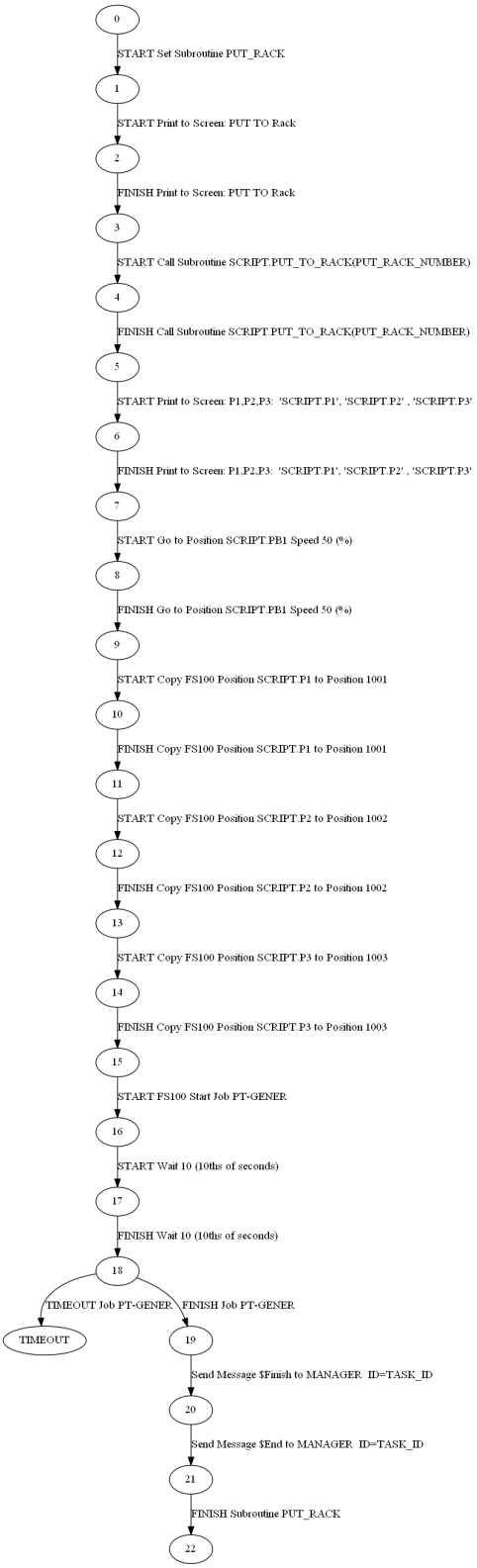

Figura 21 – Autômato da subrotina PUT_RACK

```
 digraph PUT_RACK {
0 -> 1 [label = "START Set Subroutine PUT_RACK"];
1 -> 2 [label = "START Print to Screen: PUT TO Rack"];
2 -> 3 [label = "FINISH Print to Screen: PUT TO Rack"];
3 -> 4 [label = "START Call Subroutine SCRIPT.PUT_TO_RACK(PUT_RACK_NUMBER)"];
```

```
4 -> 5 [label = "FINISH Call Subroutine SCRIPT.PUT_TO_RACK(PUT_RACK_NUMBER)"];
5 -> 6 [label = "START Print to Screen: P1,P2,P3:  'SCRIPT.P1', 'SCRIPT.P2' , 'SCRIPT.P3' "];
6 -> 7 [label = "FINISH Print to Screen: P1,P2,P3:  'SCRIPT.P1', 'SCRIPT.P2' , 'SCRIPT.P3' "];
7 -> 8 [label = "START Go to Position SCRIPT.PB1 Speed 50 (%)"];
8 -> 9 [label = "FINISH Go to Position SCRIPT.PB1 Speed 50 (%)"];
9 -> 10 [label = "START Copy FS100 Position SCRIPT.P1 to Position 1001"];
10 -> 11 [label = "FINISH Copy FS100 Position SCRIPT.P1 to Position 1001"];
11 -> 12 [label = "START Copy FS100 Position SCRIPT.P2 to Position 1002"];
12 -> 13 [label = "FINISH Copy FS100 Position SCRIPT.P2 to Position 1002"];
13 -> 14 [label = "START Copy FS100 Position SCRIPT.P3 to Position 1003"];
14 -> 15 [label = "FINISH Copy FS100 Position SCRIPT.P3 to Position 1003"];
15 -> 16 [label = "START FS100 Start Job PT-GENER"];
16 -> 17 [label = "START Wait 10 (10ths of seconds)"];
17 -> 18 [label = "FINISH Wait 10 (10ths of seconds)"];
18 -> TIMEOUT [label = "TIMEOUT Job PT-GENER"];
18 -> 19 [label = "FINISH Job PT-GENER"];
19 -> 20 [label = "Send Message $Finish to MANAGER  ID=TASK_ID"];
20 -> 21 [label = "Send Message $End to MANAGER  ID=TASK_ID"];
21 -> 22 [label = "FINISH Subroutine PUT_RACK
"];
}
```