UNIVERSIDADE FEDERAL DO PARANÁ

FRANCISCO RICARDO TABORDA AGUIAR

INTERPRETAÇÃO DOS DIALETOS RS274-D E EXTRAÇÃO DOS DADOS TEMPORAIS DA USINAGEM EM MÁQUINAS CNC

CURITIBA

2023

FRANCISCO RICARDO TABORDA AGUIAR

INTERPRETAÇÃO DOS DIALETOS RS274-D E EXTRAÇÃO DOS DADOS TEMPORAIS DA USINAGEM EM MÁQUINAS CNC

Dissertação apresentada como requisito parcial à obtenção do título de Mestre em Engenharia de Manufatura, Programa de Pós-Graduação em Engenharia de Manufatura, Setor de Tecnologia, Universidade Federal do Paraná .

Orientador: Prof. Dr. Dalberto Dias da Costa

CURITIBA

2023

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
UNIVERSIDADE FEDERAL DO PARANÁ
SISTEMA DE BIBLIOTECAS – BIBLIOTECA DE CIÊNCIA E TECNOLOGIA

Aguiar, Francisco Ricardo Taborda
Interpretação dos dialetos RS274-D e extração dos dados temporais da usinagem em máquinas CNC / Francisco Ricardo Taborda Aguiar. – Curitiba, 2023.
1 recurso on-line : PDF.

Dissertação (Mestrado) - Universidade Federal do Paraná, Setor de Tecnologia, Programa de Pós-Graduação em Engenharia de Manufatura.

Orientador: Dalberto Dias da Costa

1. Máquinas-ferramenta - Controle numérico - Programação. 2. Programação (Computadores). 3. Usinagem. 4. Máquinas - Indústria. I. Universidade Federal do Paraná. II. Programa de Pós-Graduação em Engenharia de Manufatura. III. Costa, Dalberto Dias da. IV. Título.

Bibliotecário: Elias Barbosa da Silva CRB-9/1894

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação ENGENHARIA DE MANUFATURA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **FRANCISCO RICARDO TABORDA AGUIAR** intitulada: **INTERPRETAÇÃO DOS DIALETOS RS274-D E EXTRAÇÃO DOS DADOS TEMPORAIS DA USINAGEM EM MÁQUINAS CNC**, sob orientação do Prof. Dr. DALBERTO DIAS DA COSTA, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.
A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 15 de Setembro de 2023.

Assinatura Eletrônica
03/10/2023 15:54:41.0
DALBERTO DIAS DA COSTA
Presidente da Banca Examinadora

Assinatura Eletrônica
03/10/2023 15:17:35.0
ADRIANO FAGALI DE SOUZA
Avaliador Externo (UNIVERSIDADE FEDERAL DE SANTA CATARINA)

Assinatura Eletrônica
03/10/2023 16:32:30.0
CLAUDIMIR JOSÉ REBEYKA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

AGRADECIMENTOS

Em primeiro lugar, a Deus, por ter permitido que eu tivesse saúde e determinação para não desanimar durante a realização deste trabalho. À minha esposa e aos meus filhos, por compreenderem as várias horas em que eu estive ausente por conta do envolvimento com os estudos. Ao meu orientador, Dr. Dalberto Dias da Costa, pela sua inestimável supervisão, apoio técnico e conselhos durante todo o meu mestrado. Minha gratidão se estende à Universidade Federal do Paraná, pela oportunidade para a realização do curso. Aos docentes, diretores, coordenadores e administração do curso que proporcionaram o ambiente adequado para o desenvolvimento dos estudos.

Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12, 2)

RESUMO

Os programas *CN* (Controle Numérico) fornecem as instruções para os movimentos de corte, para os movimentos de avanço, para as trocas de ferramentas e para outras ações realizadas na máquina-ferramenta durante a usinagem de peças. Portanto, os programas podem fornecer dados importantes para o entendimento dos eventos que ocorrem durante a usinagem. A geração de programas para os equipamentos *CN* é baseada em um padrão conhecido como *RS274-D*. Porém, os fabricantes de máquinas têm estendido esse padrão, incluindo novos comandos ou modificando a sintaxe de comandos existentes para adaptá-los para funções customizadas. As diferenças de sintaxe levaram ao surgimento de diversos dialetos do padrão *RS274-D*. Esses dialetos dificultam o uso do programa *CN* como uma fonte de dados para auxiliar na simulação do processo de usinagem. Além de resultar na falta de interoperabilidade dos programas entre diferentes sistemas. Embora exista o interesse acadêmico para a utilização de novos formatos que sejam capazes de resolver o problema gerado pelos dialetos, como o *STEP-NC*, os programas *RS274-D* ainda predominam na indústria. Diante desse contexto, o presente trabalho propõe uma metodologia para interpretar os dialetos e extrair uma estrutura de dados que forneça um detalhamento a respeito dos eventos definidos no programa e a marcação de tempo em que cada um desses eventos deverá ocorrer na máquina-ferramenta. A estrutura de dados possibilitou a criação de um modelo virtual da usinagem da peça na máquina-ferramenta, que pode ser utilizado para análises e simulações do processo de usinagem. O algoritmo para a interpretação dos dialetos foi desenvolvido a partir de técnicas de Análise Léxica e Análise Sintática. Os dados extraídos são pós-processados por programas que implementam as funções customizadas do dialeto. As regras de produção da Análise Sintática e as funções customizadas podem ser adaptadas para diferentes dialetos, o que garante flexibilidade ao sistema proposto. Foi apresentada uma arquitetura escalável para a implementação do software, com uma abordagem baseada em microserviços. O projeto foi modelado por meio de diagramas UML (Unified Modeling language). Como estudo de caso, foi implementado um protótipo para processar programas CN escritos no dialeto *Mach-9* do fornecedor *Romi*. O algoritmo foi capaz de reconhecer os códigos *G* e gerar a estrutura de dados. A partir dos dados foi possível simular as marcações de tempo relacionados com as mudanças das velocidades de rotação, das trocas de ferramentas e das velocidades de avanço ao longo da usinagem. O modelo de dados desenvolvido possibilitou a transpilação para as Funções Canônicas de Usinagem, que podem formar uma camada abstrata para a integração com outros sistemas, visto que apresentam uma sintaxe padronizada e livre de dialetos.

Palavras-chave: transpilação; funções canônicas de usinagem; programação CN.

ABSTRACT

The NC (Numerical Control) programs provide instructions for the cutting movements, feed movements, tool changes, and other actions performed into the machine tool while machining parts. Therefore, the programs can provide relevant data about the events which occur during machining. The *NC* programming is based on a standard called *RS274-D*. However, the CNC manufacturers have extended this NC language by including new commands or changing the syntax of some existing commands to adapt them with their customized functions. The differences in the syntax led to the emergence of several dialects of the *RS274-D* standard. These dialects make it hard to use the *NC* program as a data source to compose the machining process simulation. In addition to resulting in the lack of interoperability of programs between different systems. Despite the academic interest in the new formats capable of solving the problem generated by dialects, such as *STEP-NC*, *RS274-D* programs still predominate in the industry. In this context, the present work proposes a methodology to interpret the dialects and extract a data structure that provides detail about the events defined in the program and the timestamp in which each of these events should occur on the machine tool. The data structure enabled the creation of a virtual model of machining the workpiece on the machine tool, which can be used for analysis and simulations of the machining process. The algorithm for the interpretation of the dialects was developed from Lexical Analysis and Syntactic Analysis techniques. The extracted data is post-processed by programs that implement the dialect custom functions. The Parsing Production Rules and the Custom Functions can be adapted to different dialects, which guarantees flexibility to the proposed system. The software architecture is scalable and is based on a microservices approach. The project was modeled using UML (Unified Modeling Language) diagrams. As a case study, a prototype was implemented to parse *NC* programs written in the Mach-9 dialect from the supplier Romi. The algorithm was able to recognize the *G codes* and generate the data structure. From the data, it was possible to simulate the timestamps related to changes in rotational speeds, tool changes, and feed rates during machining. The data model made possible the transpilation to the Canonical Machining Functions, which can act as an abstract layer for integration with other systems since they present a standardized, dialect-free syntax.

Key-words: transpilation; canonical machining function; nc programming.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A usinagem é uma atividade relevante na indústria de manufatura e responsável por uma parcela significativa do total de energia consumida neste setor. A usinagem está presente em diversas indústrias. Na indústria metalúrgica, a transformação dos materiais metálicos ocorre, principalmente, através do uso de máquinas CNC (Comando Numérico Computadorizado). Empresas de outros ramos, também, se utilizam de equipamentos CNC no seu processo produtivo. Por exemplo, a fabricação de placas de circuitos impressos demanda o uso de máquinas CNC para furação e fresamento.

Camargo et al. (2015) pontuaram que conhecer o tempo real de usinagem e relacioná-lo com os demais parâmetros significativos do processo é fundamental para o planejamento da produção e para a definição dos custos das operações. Porém, obter o tempo real de usinagem corresponde a uma grande dificuldade no processo de fabricação. Os autores apontam que as estimativas feitas pelos sistemas CAM podem ser muito inferiores ao tempo real.

A comunicação, o compartilhamento e a troca de dados é um fator importante para o processo de digitalização da linha de produção na indústria. Pois os engenheiros e especialistas demandam um acesso rápido e estruturado dos dados coletados ao longo do processo de manufatura. Botkina et al. (2018) notaram que a digitalização do planejamento e o desenvolvimento de um processo de produção moderno utilizam o conceito de *Gêmeos Digitais* [1] ao invés de protótipos físicos da linha de produção. Porém, o processo de construção de um *Gêmeo Digital* demanda um modelo de informação padronizado. Além disso, a extração de dados das máquinas demanda o uso de tecnologias específicas. Dizdarević et al. (2019) indicaram que os protocolos para a troca de mensagens entre dispositivos incluem MQTT [2], AMQP [3], XMPP [4], DDS [5], HTTP [6], e CoAP [7], entre outros. Na camada de rede, protocolos como o LoRaWan [8], Bluetooth e ZigBee são exemplos comumente utilizados. Máquinas antigas não possuem estes recursos. E mesmo em equipamentos modernos, fornecedores diferentes podem utilizar protocolos de comunicação diferentes. Trabalhos têm sido desenvolvidos para o compartilhamento de dados a nível de aplicação. Por exemplo, o MTConnect, como demonstrarm os autores Liu et al. (2018). Mas estes demandam a

[1] Do Inglês, Digital Twin
[2] Message Queuing Telemetry Transport
[3] Advanced Message Queuing Protocol
[4] Extensible Messaging and Presence Protocol
[5] Data Distribution Service
[6] Hypertext Transfer Protocol
[7] Constrained Application Protocol
[8] Long Range Wide Area Network

implantação de adaptadores ou de sistemas invasivos para que ocorra a integração dos dados.

A diversidade dos formatos, também, pode ser verificada na programação *CN*. Os programas *CN* definem os comandos para a execução dos movimentos de corte, avanço, posicionamentos e outras ações necessárias à usinagem. Portanto, os programas podem fornecer dados importantes sobre os eventos que ocorrem durante o processo de usinagem, como a aceleração dos eixos, as trocas de ferramentas e as velocidades de avanço, por exemplo. Mas para que seja possível a extração de dados a partir dos arquivos *CN*, é necessário que os programas sejam feitos a partir de um formato de arquivo padronizado. Embora existam padrões formais para a geração dos programas, observa-se que os fornecedores de máquinas adaptam a sintaxe dos programas para atender às suas necessidades específicas. Dharmawardhana et al. (2018) pontuaram que as adaptações na sintaxe dos programas criaram incompatibilidades entre os diferentes sistemas *CN*. A mesma peça necessita de programas diferentes para ser usinada em máquinas de fornecedores diferentes. Essa condição resulta na falta de interoperabilidade no fluxo de informações após a geração e o processamento dos programas, conforme foi verificado por Zhang et al. (2015), que comentaram, também, sobre o uso dos Pós-processadores [9] nos sistemas *CAM* [10] como um esforço para gerar os programas com a sintaxe correta para os vários fornecedores de máquinas. Porém, os autores observam que há um número elevado tanto de sistemas *CAM* quanto de fornecedores de máquinas *CNC*, o que faz com que a demanda por Pós-Processadores tenha que ser muito alta.

Integração e interoperabilidade são conceitos chave nos sistemas de manufatura moderna. Adamczyk et al. (2020) notaram que a nova indústria de fábricas inteligentes deve empregar um amplo conjunto de ferramentas de software para dar suporte aos processos de manufatura, o que impõe um desafio em termos de interoperabilidade e integração.

Chen et al. (2008) observaram que o conceito de interoperabilidade difere do conceito de integração. Para eles, a integração significa que todos os componentes estão *fortemente acoplados*, o que indica que os componentes são interdependentes e não podem ser separados. Por outro lado, interoperabilidade significa componentes *frouxamente acoplados* nos quais eles podem trocar serviços enquanto continuam localmente sua lógica de operação. Com base nesse conceito de *acoplamento*, os autores concluem que dois sistemas integrados são, inevitavelmente, interoperáveis. Mas dois sistemas interoperáveis não são, necessariamente, integrados. Integração e

[9] Pós-Processadores são softwares ou sub-rotinas que convertem um arquivo gerado em um sistema *CAM* em um programa que possa ser identificado em um determinado sistema *CN* (MAGAMBO; YING, 2013).

[10] Computer Aided Manufacturing

interoperabilidade são conceitos fundamentais em sistemas de manufatura inteligente.

A falta de interoperabilidade afeta o fluxo de informações de fabricação de duas maneiras. Em primeiro lugar, mudança de código que ocorrer no chão de fábrica não pode ser propagada para outros sistemas, como CAPP [11], CAM e CAD [12]. Em segundo lugar, os sistemas localizados no chão de fábrica, como o QIF [13] e o MES [14] exigirão interfaces especializadas para capturar as mudanças que afetam, por exemplo, dimensões ou o tempo de processamento.

Entretanto, o fluxo da manufatura digital relacionado com o ciclo de vida de um produto ocorre de maneira unidirecional, do projeto para a fabricação. A realimentação do chão de fábrica para o projeto ainda não é suportado.

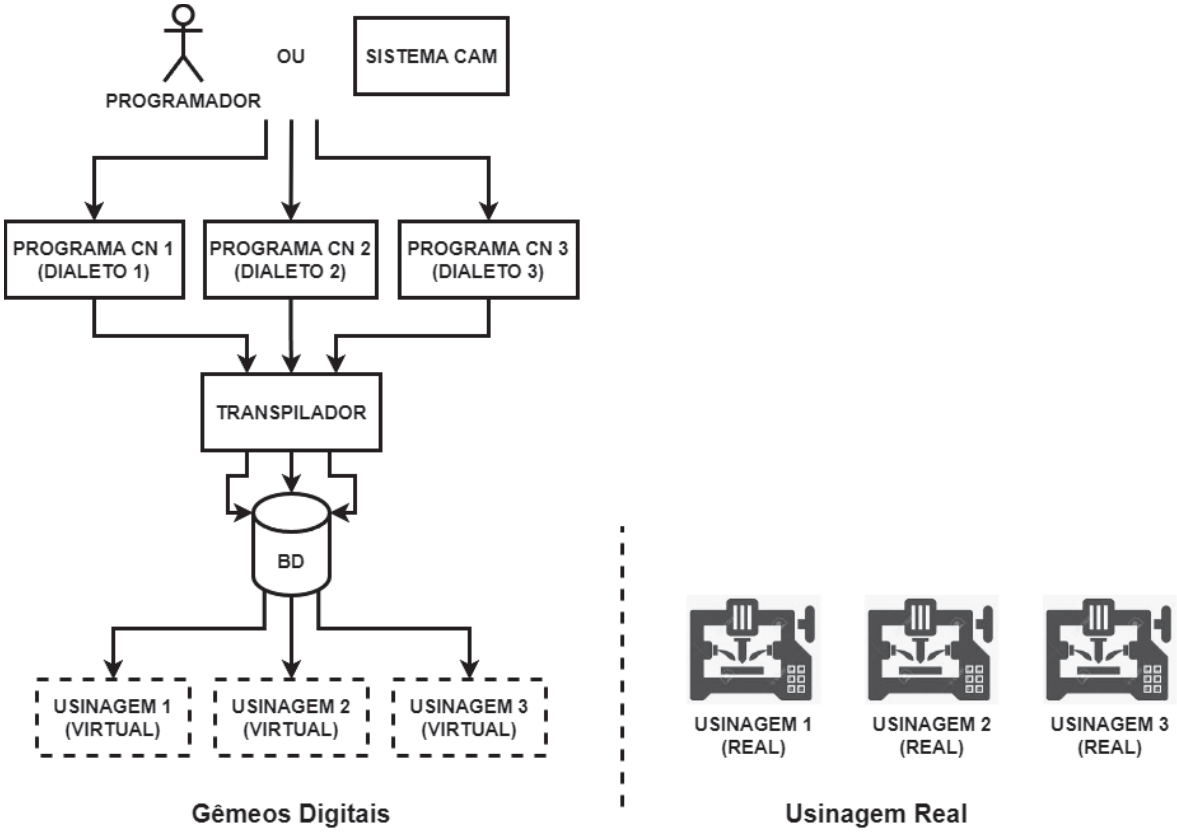
Nesse contexto, Liu e Xu (2017) propuseram uma nova geração de máquinas-ferramentas baseadas em sistemas ciber-físicos, que englobam um *Gêmeo Digital* da máquina-ferramenta. O *Gêmeo Digital* pode compartilhar dados entre a produção e os vários sistemas de Tecnologia da Informação nos diferentes níveis. Pode envolver a extração de dados a partir de sistemas CAD, CAM, CAE [15] e dados de gerenciamento a partir de sistemas ERP [16], MES, PDM [17], entre outros. Além disso, podem explorar dados operacionais de equipamentos na manufatura, tais como dados do produto, dados da qualidade e dados da manutenção. Os componentes no nível físico permitem carregar dados em tempo real para a sincronização do *Gêmeo Digital* com o seu correpondente gêmeo físico, para detectar anomalias, predição e otimização. Para a criação de um *Gêmeo Digital*, porém, é necessário coletar e analisar dados de várias fontes, como dimensões físicas, informações sobre a manufatura, dados operacionais e fluxos de informação. Essas informações são combinadas em um modelo virtual. Mas essa construção requer um modelo de informação padronizado.

Apesar do interesse em torno de novos formatos de dados que proporcionem a padronização e que sejam capazes de tratar do problema da falta de interoperabilidade, como o *STEP-NC* (NASSEHI et al., 2008), e da inovação tecnológica que vem acontecendo nos equipamentos CNC, o formato *RS274-D* ainda predomina. Os controladores de CNC não conseguem traduzir diretamente os códigos do formato *STEP-NC*. Controladores e softwares CAM baseados em *STEP-NC* parecem ser a exceção e não a regra no momento em que a pesquisa foi realizada. Neste contexto, uma camada intermediária entre controladores *CN* atuais e um padrão de alto nível precisa ser desenvolvido.

[11] Computer Aided Process Planning
[12] Computer Aided Design
[13] Quality Information Framework
[14] Manufacturing Execution System
[15] Computer Aided Engineering
[16] Enterprise Resource Planning
[17] Product Data Management

O presente trabalho apresenta uma abordagem para interpretar um programa *RS274-D* e convertê-lo para um modelo de dados padronizado. O modelo de dados é capaz de fornecer um detalhamento dos eventos definidos no programa e a marcação de tempo em que cada um desses eventos deve ocorrer na máquina-ferramenta. A interpretação contempla a interpretação das funções customizadas do programa. Por exemplo, cada comando de ciclo de furação é convertido para uma sequência de blocos de dados, aonde cada bloco representa um único movimento na máquina, o que facilita a extração de dados (por exemplo, cálculos de deslocamento e marcação de tempo). A estrutura de dados fornece um modelo virtual da usinagem da peça no equipamento, que pode ser utilizado para análises e simulações do processo de usinagem. O desenho da FIGURA 1 apresenta uma visão conceitual dessa proposta. O sistema proposto pode ser utilizado tanto em equipamentos modernos (com tecnologia embarcada) quanto em equipamentos antigos (desprovidos de tecnologias modernas).

FIGURA 1 – PROPOSTA PARA O MODELO VIRTUAL DA USINAGEM

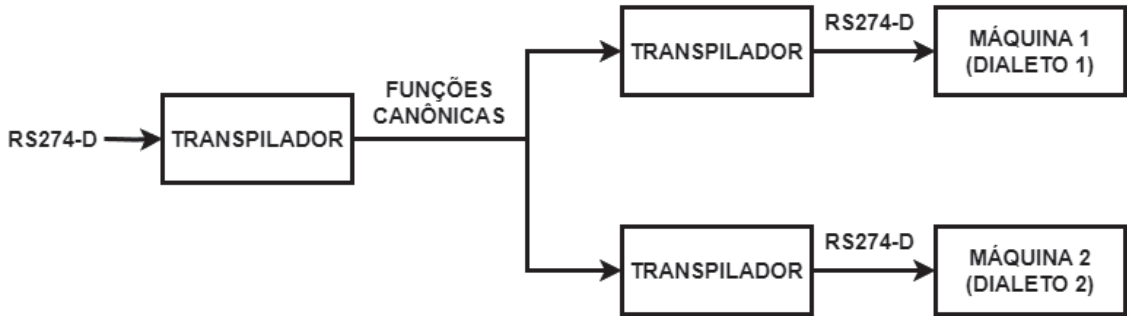


FONTE: Autor (2023)

Além de fornecer um modelo virtual, os dados podem ainda ser exportados para um arquivo de Funções Canônicas de Usinagem. As Funções Canônicas de Usinagem podem fornecer uma camada intermediária de abstração, pois são uma representação neutra (ZHANG et al., 2015) e apresentam uma sintaxe padronizada,

o que não ocorre com os diversos dialetos [18] *RS274-D* (DHARMAWARDHANA et al., 2018). Essa camada de abstração facilita a interoperabilidade e a integração com outros sistemas. A FIGURA 2 apresenta uma possível aplicação da camada de abstração. O arquivo de entrada consiste em um dialeto do padrão *RS274-D* que é traduzido para Funções Canônicas de Usinagem. Em uma segunda etapa, as Funções Canônicas de Usinagem são convertidas para outros dialetos *RS274-D*. Assim, é possível fazer com que o mesmo programa *CN* de entrada possa ser traduzido para ser utilizado em diferentes sistemas.

FIGURA 2 – PROPOSTA PARA CAMADA DE ABSTRAÇÃO



FONTE: Autor (2023)

Outra possível aplicação do sistema proposto está relacionada com a integração de sistemas, conforme demonstrado na figura FIGURA 3. O modelo de dados extraído do programa *CN* pode ser importado por outros sistemas (como um *ERP*, por exemplo) para o levantamento de informações, tais como o tempo de usinagem e a lista de ferramentas.

FIGURA 3 – PROPOSTA PARA INTEGRAÇÃO DE SISTEMAS

FONTE: Autor (2023)

As seções 1.1 e 1.2 apresentam os objetivos do projeto. Já o capítulo 2 expõe a literatura utilizada como referência no trabalho. O capítulo 3 faz o detalhamento da metologia proposta no projeto. O capítulo 4, por sua vez, apresenta um estudo de caso e a implementação. O capítulo 5 demonstra os resultados obtidos com a implementação. Finalmente, o capítulo 6 faz a conclusão do projeto e propõe os trabalhos futuros.

[18] Dialeto: termo emprestado da linguagem natural para ilustrar as diferenças de sintaxe encontradas nos programas *CN* gerados para diferentes sistemas, conforme verificado por Zhang et al. (2015).

1.1 OBJETIVO GERAL

Desenvolver uma metodologia para interpretar os diferentes dialetos do formato *RS274-D* e gerar um modelo de dados, no qual cada movimento da máquina seja registrado com uma marcação de tempo.

1.2 OBJETIVOS ESPECÍFICOS

São definidos os seguinte objetivos específicos.

- Projetar uma estrutura de dados que contemple dois objetivos:
  - Possibilitar o registro temporal dos eventos decorrentes do processamento do programa de controle numérico.
  - Formar uma camada abstrata de dados que possa ser utilizada para tratar das questões da interoperabilidade entre os dialetos.
- Representar, por meio de diagramas abstratos, a metodologia proposta na dissertação.
- Implementar um estudo de caso a partir de um dialeto específico.

2 REVISÃO BIBLIOGRÁFICA

As próximas seções tratam da consulta bibliográfica utilizada como referência para o desenvolvimento desta dissertação. A seção 2.1 apresenta conceitos referentes ao paradigma conhecido como *Gêmeo Digital*. A seção 2.2 trata do padrão *RS274-D*. O padrão *STEP-NC* é brevemente apresentado na seção 2.3. As Funções Canônicas de Usinagem são apresentadas na seção 2.4. A seção 2.5 trata da *Transpilação*, dos conceitos que envolvem essa técnica e apresenta alguns exemplos. Já a seção 2.6 faz uma apresentação sucinta sobre *Tradução Automática Neural*.

2.1 GÊMEO DIGITAL

Rabelo et al. (2020) traduziram o conceito de *Gêmeo Digital* como uma simulação integrada, multi-física, multi-escala e probabilística de um sistema, que usa modelos físicos e dados para espelhá-lo de forma fidedigna e em tempo real, cuidando de forma preditiva da sua própria permanência no ambiente frente a problemas não previstos no sistema real ou virtual.

Segundo Semeraro et al. (2021) o *Gêmeo Digital* consiste em um novo paradigma para simulações e representa a evolução de outros campos de pesquisa, tais como:

- Virtual Manufacturing (VM): sistema cujo objetivo é gerar uma representação virtual de um sistema físico sem o uso de elementos reais.
- Model-based Predictive Control (MPC): usa um modelo do sistema para fazer predições sobre o comportamento futuro do sistema.
- Building Information Modelling (BIM): é um processo para criar e gerenciar um modelo contendo informação digital sobre um determinado ativo, com o objetivo de melhorar a eficiência do seu projeto e construção.

Lu et al. (2020) fizeram uma revisão sobre o desenvolvimento de tecnologias em sistemas de manufatura e processo, no contexto da Indústria 4.0. Os autores citam cenários de utilização de *Gêmeos Digitais* em processos de manufatura e apresentam uma visão geral de aplicações existentes. No artigo, os autores demonstraram um modelo de referência e apresentaram tecnologias, ferramentas e padrões para o desenvolvimento de *Gêmeos Digitais* para Fábricas Inteligentes. Os autores acreditam que o processo de construção de um *Gêmeo Digital* necessita de um modelo de informação

padronizado. Além disso, as pesquisas em torno de protocolos de comunicação, padrões, processamento de dados temporais e confiabilidade necessitam ser priorizadas para o próximo estágio de desenvolvimento dos *Gêmeos Digitais*.

Semeraro et al. (2021) pontuaram que os *Gêmeos Digitais* podem ser usados para auxiliar no projeto de novos produtos, testar protótipos em situações de tempo real, prever como o usuário final ira utilizar o produto e como o projeto irá complementar o ambiente do produto. Dados em tempo real para o modelo digital para simular e testar idéias antes mesmo do processo de manufatura ser iniciado. Os *Gêmeos Digitais* podem ser utilizados, também, para planejar e reconfigurar o produto em resposta às mudanças externas. Com relação à manufatura, um *Gêmeo Digital* para controle e otimização pode analisar dados coletados a partir de dispositivos físicos e procurar soluções otimizadas para o chão de fábrica. Por exemplo, Lu et al. (2020) propuseram um método de modelagem para máquinas CNC.

Semeraro et al. (2021) pontuaram que o *Gêmeo Digital* é um elemento chave para o alinhamento das empresas de manufatura com a revolução da tecnologia da informação. O *Gêmeo Digital* embute uma imagem virtual da realidade, sendo constantemente sincronizada com o cenário real para que possa fornecer um modelo virtual de alta fidelidade para simular os estados e comportamentos com a capacidade de avaliar, otimizar e fazer predições. Diferentes estratégias relacionadas com a próxima revolução industrial apresentam como objetivo comum a integração do espaço físico com o espaço virtual. Isso visa obter maior flexibilidade e escalabilidade dos sistemas de manufatura através das tecnologias de informação. Os autores em Semeraro et al. (2021) citaram que o conceito para o uso de *Gêmeos* foi verificado no programa Apollo da NASA, aonde dois veículos espaciais idênticos foram construídos para permitir o espelhamento das condições do espaço durante a missão.

Botkina et al. (2018) propuseram uma réplica digital de uma ferramenta física, apresentando o formato de dados e a estrutura para armazenar o fluxo de informações e o gerenciamento de dados para que possa ser utilizado em aplicações de análise de dados e produtividade. A proposta apresentada pelos autores utiliza o formato *ISO 13399*, que é um padrão internacional para a representação de dados de ferramentas de corte (LI et al., 2014). É apresentado, pelos autores, um modelo que coleta dados automaticamente, por meio da arquitetura *LISA* (THEORIN et al., 2017) e mensagens padronizadas, que os autores chamaram de *Tweets*. O modelo possibilita ajustes contínuos do modelo digital. O ajuste é feito através do mecanismo que os autores chamaram de *Tweeting Machine*, que alimenta a linha de produção com dados. Esse mecanismo possibilita o aperfeiçoamento do modelo digital da ferramenta, a partir do momento em que foi instanciada conforme o projeto, passando pelo processo de montagem e os dados referentes ao seu uso na linha de produção, por meio de contextos

no padrão *STEP-NC*. Assim, o modelo da ferramenta pode representar, de forma mais precisa, as propriedades da ferramenta de corte. Isso possibilita simulações, análises e controle para promover a melhoria contínua do processo. Os autores apresentaram um caso de uso na manufatura de produtos de veículos pesados. No esquema proposto, o projeto é gerado no sistema *CAD*. A descrição do processo de manufatura é concebida pelo sistema *CAM*. O modelo digital da ferramenta, baseado no formato ISO 13399, é obtido pelo *ToolMaker* e informado pela *Tweeting Machine*. Assim, a usinagem pode ser planejada, simulada e verificada. As ferramentas são ordenadas e entregues. A montagem da ferramenta ocorre com base na informação obtida pelo *ToolMaker*. As dimensões da montagem da ferramenta são medidas e informadas. Assim, uma instância (ou ferramenta física) do modelo digital é criada. Uma referência do modelo da ferramenta física é informada para o banco de dados. Durante a manufatura, os dados da ferramenta (como medidas, compensações das feramentas e dados de corte) são informados em contextos do formato *STEP-NC*. A análise dos dados coletados pode ser feita por um software de aprendizado de máquina. Todas as informações relacionadas com a utilização da ferramenta são visualizadas pelo operador. Operações de gargalo, fim inesperado da vida útil da ferramenta e ajustes no corte para manter tolerância, são exemplos de mensagens que podem ser enviadas para os Especialistas. Assim, o processo pode ser verificado, analisado e colocado em produção. A réplica pode ser utilizada para o planejamento do processo em uma solução de usinagem otimizada. Isto garante o processo de melhoria contínua do processo e da ferramenta. Além de uma melhor predição do comportamento, simulações e cálculos da performance.

Uma vez que o programa *CN* define as instruções para serem executadas na máquina-ferramenta, a criação de um modelo virtual da usinagem a partir dos eventos definidos no programa pode levar à construção de um *Gêmeo Digital* para o processo de usinagem. A próxima seção vai apresentar o Padrão *RS274-D*, que corresponde ao padrão predominante para a programação *CN*.

2.2 PADRÃO RS274-D

Nos anos de 1960, a *EIA* [1] desenvolveu um padrão conhecido como *RS274* para programar máquinas CNC [2]. A última versão do padrão é chamada *RS274-D* e data de 1979. Em 1982 este padrão foi adotado pela ISO sendo chamado de *ISO6983*. A norma *ISO6983-1:2009* (ISO, 2009) traz especificações e recomendações para o formato de dados para o posicionamento, a movimentação e o controle de trajetórias para máquinas de controle numérico. Com isto, o objetivo é reduzir a variedade de programas e promover a uniformidade de técnicas de programação para favorecer

[1] Electronic Industry Association
[2] Computer Numeric Control

a intercambiabilidade de programas entre diferentes máquinas. Outra proposta do padrão, é que máquinas simples possam ser programadas com um formato simples de programa, o qual possa ser extensível para máquinas mais complexas. Os programas padrão *RS274-D* são organizados em linhas de código, também chamados de blocos, os quais contém conjuntos de comandos para o sistema de controle do equipamento. Um bloco consiste de um número de linha opcional no início do bloco, seguido por uma ou mais palavras, cada qual com uma instrução específica. Uma palavra consiste de uma letra, sucedida por um sinal algébrico, se aplicável, seguido por um número ou por uma expressão que pode ser avaliada para um número (KRAMER; PROCTOR, 1998). Cada palavra pode definir um comando ou fornecer um argumento para um comando. Por exemplo, *"G01X15"* é uma linha de código com duas palavras:

- *G01*: Mover em linha com com a velocidade de avanço programada.
- *X15*: Fornece o valor 15, eixo X, para o comando.

As palavras devem ser apresentadas na seguinte sequência dentro do bloco (ISO, 2009):

- As palavras de preparação *G*;
- As palavras de dimensão, que deveriam ser arranjadas na seguinte sequência: *X*, *Y*, *Z*, *U*, *V*, *WP*, *Q*, *R*, *A*, *B*, *C*.
- As palavras de interpolação *I*, *J* e *K*, que são aplicáveis somente para determinados grupos de eixos.
- A palavra de função de avanço (*F*).
- A palavra de função de velocidade de corte (*S*).
- A palavra de função de ferramenta (*T*).
- A palavra de função de miscelânea (*M*).

Muitos comandos permanecem ativos até que sejam modificados, seja explicitamente ou implicitamente, por outro comando. Tais comandos são chamados *modais*. Os comandos *não modais*, por sua vez, têm efeito somente nas linhas em que eles ocorrem.

A TABELA 1 lista os comandos definidos para a linguagem RS274/VGER, que é uma extensão do formato *RS274-D*, descrita em Kramer e Proctor (1998).

TABELA 1 – COMANDOS MODAIS PARA O FORMATO RS274/VGER

Comandos	**Propósito**
G0, G1, G2, G3, G38, G80, G81, G82, G83, G84, G85, G86, G87, G88, G89	Movimento
G17, G18, G19	Seleção do plano
G90, G91	Modo das distâncias
G93, G94	Modo da velocidade de corte
G20, G21	Unidade
G40, G41, G42	Compensação do diâmetro de corte
G43, G49	Distância da ferramenta
G98, G99	Modo de retorno em ciclos fixos
G54, G55, G56, G57, G58, G59, G59.1, G59.2, G59.3	Seleção do sistema de coordenadas
M0, M1, M2, M30, M60	Parada
M6	Troca de ferramenta
M3, M4, M5	Sentido do giro do eixo
M7, M8, M9	Refrigeração
M48, M49	Interruptor do avanço e do corte

FONTE: **Adaptado de** Kramer e Proctor (1998))

Alguns países da Europa adotaram extensões para o padrão. Por exemplo, o padrão DIN66025, que é usado na Alemanha. Guo et al. (2012) observam que os padrões são apenas nominais hoje em dia, pois a tecnologia dos sistemas CNC avançou muito desde que as normas foram publicadas. Muitas funções, não suportadas pelas normas, foram adicionadas aos sistemas CNC. Além disto, os fornecedores de máquinas estenderam os padrões de programação CNC com sintaxes e comandos adicionais para adaptá-los às suas funções customizadas. Atualmente, segundo Zhang et al. (2015), as diferenças na sintaxe levam aos chamados dialetos *RS274-D* ao invés de uma linguagem NC padrão. Esses dialetos impõem uma barreira para a interoperabilidade e para a integração dos programas *CN* entre diferentes sistemas. Assim, diferentes programas devem ser gerados para usinar a mesma peça em diferentes máquinas. Para demonstrar estas diferenças de sintaxe, a TABELA 2 apresenta alguns exemplos de comandos e como os mesmos devem ser escritos em dois diferentes fornecedores de máquina.

TABELA 2 – COMPARAÇÕES ENTRE DIALETOS

Comando	**Fanuc**	**Siemens**
Interpolação Linear	G01 X...Y...	G1 X...Y...
Interpolação Circular (Horário)	G02 X...Y...I...J...	G2 X...Y...I...J...
Ciclo Fixo de Furação	G83...	CYCLE G83...
Definicação da Unidade	G20 ou G21	G70 ou G71

FONTE: Adaptado de Zhang et al. (2015)

Neste cenário de diversidade de dialetos *RS274-D*, Dharmawardhana et al. (2018) comentam sobre o surgimento de um padrão chamado *Standard for the Ex-*

change of Product Model Data (*STEP*), *ISO 10303*, para a troca de dados entre *CAD / CAPP / CAM*. Esse padrão é tratado, de forma sucinta, na próxima seção.

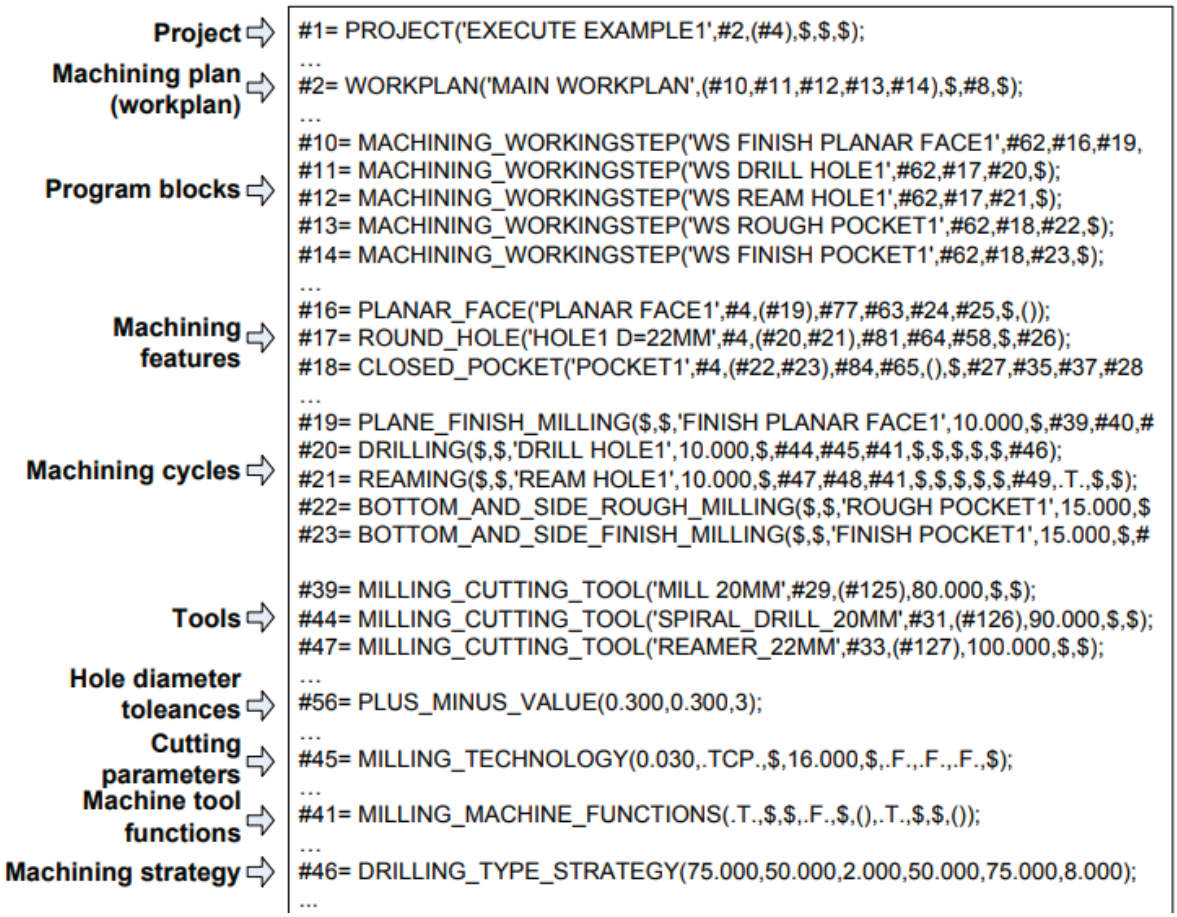
2.3 PADRÃO STEP-NC

Um grande esforço para enfrentar o problema da interoperabilidade e da integração tem sido realizado pelo Comitê Técnico da ISO 184 - Sistemas de Automação Industrial e Integração - e foi formalizado na norma ISO 10303 *Sistemas de Automação Industrial e Integração - Representação e troca de dados do produto* também conhecido como *STEP* (KASSIM et al., 2016). A transferência de dados entre sistemas CAD/CAM foi contemplada pelo Comitê Técnico ISO 184 em duas abordagens: como o modelo de referência da aplicação (ARM) da ISO14649 e como o modelo interpretado da aplicação (AIM), que está documentada na ISO10303 *Application Protocol 238* (KASSIM et al., 2016). Ambas as abordagens são conhecidas como STEP-NC (KUMAR et al., 2007).

Conforme apontado por Liu et al. (2018) e Navas et al. (2021) o conjunto de normas acima mencionado é uma condição necessária para resolver a falta de interoperabilidade e integração no atual processo de usinagem em máquinas CNC e promover a implementação de fabricação inteligente com base em máquinas-ferramenta ciber-físicas.

A FIGURA 4 apresenta um trecho de um programa em *STEP-NC* para ilustrar o modelo de dados desse padrão.

FIGURA 4 – EXEMPLO DE ARQUIVO *STEP-NC* PADRÃO ISO14649



FONTE: Adaptado de Pobozniak e Sobieski (2017)

Muitos trabalhos têm sido escritos visando o uso do padrão STEP-NC para integrar o fluxo de informações no processo de usinagem. Jaider et al. (2015), por exemplo, propuseram a construção de um sistema de decisão para a seleção de ferramentas de corte. A seleção ocorre automaticamente a partir das características da usinagem que são extraídas do STEP-NC através de um programa escrito em Python. Os autores Álvares et al. (2020) desenvolveram seis arquiteturas STEP-NC diferentes para aplicações de usinagem robótica. Danjou et al. (2017) propuseram uma abordagem de circuito de fabricação fechado (closed loop) para obter o feedback de informações das máquinas CNC para os sistemas CAM, baseados em STEP-NC. Dharmawardhana et al. (2018) fizeram uma revisão sobre a pesquisa e o desenvolvimento de controladores baseados em STEP-NC na última década e os seus recursos para implementar a interoperabilidade em um circuito de fabricação fechado.

A falta de capacidade de monitoramento e rastreabilidade no STEP-NC foi abordada por Campos e Miguez (2011). Dois esforços complementares relatados na literatura tratam da aparente falta de capacidade de monitoramento observada no STEP-NC. O primeiro é o MTConnect Standard. MTConnect é um padrão para a troca

de dados e informações que é baseado em um dicionário de dados de termos que descrevem informações associadas com as operações da fabricação (MTCONNECT-INSTITUTE, 2021). O segundo é o OPC-UA (Open Platform Communications - Unified Architecture). O OPC-UA 40501 especifica um modelo de informação para a representação de uma máquina-ferramenta, fornecendo condições para permitir a troca de informações entre uma máquina-ferramenta e sistemas de software como MES, SCADA, ERP, ou sistemas de análise de dados (OPC-FOUDATION, 2021).

Apesar do interesse em torno do formato *STEP-NC* e da inovação tecnológica que vem acontecendo nos equipamentos de CNC, o formato *RS274-D* ainda é dominante. Os processos de usinagem ainda dependem de máquinas antigas controladas por sistemas proprietários que, nas palavras de Zhang et al. (2015), são considerados dialetos das normas *EIA 274D* ou *ISO 6983*.

Os autores em Odendahl et al. (2008) propuseram o desenvolvimento de uma camada adicional de software que utiliza sub-programas ou macros para tratar da portabilidade de programas entre diferentes fornecedores de máquinas CNC. Esta camada de software foi chamada de Máquina Virtual CNC e o objetivo é permitir que um programa em *STEP-NC* seja traduzido para o padrão *RS274-D* e seja portável entre diferentes máquinas CNC. A motivação inicial para o projeto da Máquina Virtual CNC foi para testar a portabilidade do *STEP-NC* em programas na Boeing e no NIST. Na abordagem proposta, um sistema *CAM* gera o programa em *AP238* para o tradutor da Máquina Virtual que utiliza chamadas genéricas de Macros ao invés de código específico em *RS274-D*. A idéia é que cada fabricante de equipamento forneça um conjunto próprio de rotinas de macros. As Macros ou Sub-programas enriquecem a programação dos equipamentos. Os blocos dos programas podem ser escritos através de variáveis, que podem ser mudadas ao longo do processamento do programa. Além disso, as Macros podem fornecer variáveis que dão acesso ao estado da máquina-ferramenta. Por exemplo, a programação de Macros permite a leitura de variáveis para a posição, velocidade de avanço, velocidade de rotação, limites da máquina, entre outros atributos. Os autores esperam que a Máquina Virtual possa oferecer uma abstração que seja capaz de padronizar a diversidade dos dialetos *RS274-D*.

2.4 FUNÇÕES CANÔNICAS DE USINAGEM

Há algum tempo, a *ISD* [3] do *NIST* [4] trabalhou no projeto *EMC* [5] (KRAMER; PROCTOR, 1998). O objetivo do projeto foi construir um padrão de interface de programação para controladores de máquinas com arquitetura aberta e demonstrar as

[3] Intelligente Systems Division
[4] National Institute of Standards and Technology
[5] Enhanced Machine Controller

implementações da arquitetura do *NGC* [6]. A arquitetura *NGC* possui partes independentes, uma das quais é a especificação para a linguagem *RS274/NGC*, uma extensão do padrão *RS274-D*. O projeto *EMC* foi desenvolvido com parceiros industriais em um controlador de máquina-ferramenta de arquitetura aberta conhecido como *VGER*. O trabalho de Kramer e Proctor (1998) descreveu o interpretador *RS274/VGER*, que consiste em um sistema que lê códigos no dialeto *RS274/NGC* e produz chamadas para um conjunto de Funções Canônicas de Usinagem. O interpretador tem dois modos: um que é integrado com o controlador *EMC* e outro que é autônomo. O sistema de controle *EMC* interpreta códigos *CN* de arquivos ou comandos individuais inseridos através da capacidade de entrada manual de dados do sistema de controle. No modo autônomo, o interpretador lê códigos a partir do teclado ou de um arquivo *CN* e imprime as Funções Canônicas de Usinagem no terminal do computador, mas pode ser redirecionado para um arquivo. Os autores definiram as regras de produção e os símbolos *tokens* para modelar a árvore de análise. O interpretador é escrito em linguagem *C++* e é focado em um dialeto específico, o *RS274/NGC*.

As Funções Canônicas de Usinagem foram definidas pelo *NIST*. Os autores em Proctor et al. (1997) explicaram que as Funções Canônicas de Usinagem foram projetadas com três objetivos em mente.

- Todas as funcionalidades dos centros de usinagem de 3 a 6 eixos devem ser cobertas pelos comandos.
- Os comandos canônicos de movimento devem ser portáveis para serem reconhecidos em placas de controle comerciais disponíveis no mercado.
- Deve ser possível interpretar comandos RS274 em Funções Canônicas de Usinagem.

As Funções Canônicas de Usinagem consistem em comandos atômicos, pois cada comando produz um único movimento de ferramenta ou um único movimento lógico. Os comandos *RS274-D*, por sua vez, incluem dois tipos. Aqueles para os quais um único comando *RS274-D* corresponde exatamente a uma Função Canônica de Usinagem. E aqueles para os quais um único comando *RS274-D* será traduzido em várias Funções Canônicas de Usinagem. A TABELA 3 contém a listagem das principais funções canônicas utilizadas no projeto. A listagem completa das Funções Canônicas de Usinagem pode ser verificada em Proctor et al. (1997).

[6] Next Generation Controller

TABELA 3 – FUNÇÕES CANÔNICAS DE USINAGEM

Função	**Descrição básica**
SELECT_PLANE (CANON_PLANE *plane*)	Utiliza o plano definido em *plane* como o plano selecionado.
USE_LENGTH_UNITS (CANON_UNITS *units*)	Define *units* como sendo a unidade para o comprimento. Os valores aceitáveis para *units* são: CANON_UNITS_INCHES (polegadas), CANON_UNITS_MM (milímetros) e CANON_UNITS_CM (centímetros).
SET_ORIGIN_OFFSETS (double *x*, double *y*, double *z*, double *a*, double *b*, double *c*)	Define a origem do programa com as coordenadas absolutas, *x*, *y*, *z*, *a* e *c*. A unidade para *x*, *y* e *z* é a mesma que está sendo utilizada no momento em que o comando foi executado. A unidade para *a*, *b* e *c* é graus.
SET_TRAVERSE_RATE (double *rate*)	Define o valor superior da velocidade do movimento de aproximação que será utilizado durante os movimentos rápidos que ocorrem, normalmente, quando o equipamento não está fazendo operações de corte.
STRAIGHT_TRAVERSE (double *x*, double *y*, double *z*, double *a*, double *b*, double *c*)	Faz um movimento de posicionamento linear a partir da posição atual até o ponto definido por *x*, *y*, *z*, *a*, *b* e *c*. É esperado que não ocorra operação de corte durante este movimento.
SET_FEED_RATE (double *rate*)	Define a velocidade de avanço.
STRAIGHT_FEED (double *x*, double *y*, double *z*, double *a*, double *b*, double *c*)	Move em linha reta, na velocidade de avanço previamente definida, a partir da posição atual até a posição dada por *x*, *y* e *z*.
SET_SPINDLE_SPEED (double *speed*)	Define a velocidade de rotação.
START_SPINDLE_CLOCKWISE ()	Gira o fuso no sentido horário na velocidade previamente definida.
START_SPINDLE_COUNTERCLOCKWISE ()	Gira o fuso no sentido anti-horário na velocidade previamente definida.
STOP_SPINDLE_TURNING ()	Comando de parada do fuso.
SELECT_TOOL (int *slot*)	Seleciona a ferramenta definida por *slot* (posição no magazine).
CHANGE_TOOL (int *slot*)	Devolve a ferramenta que está no fuso e substitui pela ferramenta definida por *slot*.
USE_TOOL_LENGTH_OFFSET (double *length*)	Define o ajuste no comprimento da ferramenta para o valor dado por *length*.
DWELL (double *seconds*)	Os eixos não se movem pelo tempo especificado no argumento *seconds*, que deve ser positivo.
COMMENT (char * *text*)	Não tem efeito físico, apenas define um comentário.

FONTE: **Autor (2022)**

Assim como ocorre com o *STEP-NC*, o formato baseado nas Funções Canônicas de Usinagem não tem sido amplamente adotado entre os controladores de CNC comerciais. Porém, a atomicidade das Funções Canônicas de Usinagem representa

uma característica interessante para o registro temporal de cada evento que ocorre na máquina-ferramenta durante o processamento do programa *CN*. Além disto, a simplicidade da sintaxe e a padronização das chamadas de funções são pontos positivos para a construção de uma camada de abstração a partir das Funções Canônicas de Usinagem.

2.5 TRANSPILAÇÃO

Segundo Kulkarni et al. (2015), um Compilador pode ser definido como um software que lê um programa escrito em uma determinada linguagem (linguagem fonte) e o traduz para um programa equivalente em outra linguagem (linguagem alvo). Guo et al. (2012) notaram que os Compiladores são utilizados para traduzir de uma linguagem de alto nível (como *C++*, por exemplo) para uma linguagem de nível inferior (como *Assembly* ou instruções de máquina). Um Transpilador, por sua vez, é um tipo de Compilador no qual tanto a linguagem fonte quanto a linguagem alvo são de alto nível Kulkarni et al. (2015).

A transpilação tem sido usada em muitas aplicações industriais. Emscripten (ZAKAI, 2011) permite a tradução de código *C/C++* para *Javascript* para a execução de aplicativos em navegadores modernos. Nunnari e Heloir (2018) apresentaram um método para implementar os controladores de movimentos de humanos virtuais em uma linguagem de implementação de referência e transpilá-lo para múltiplos mecanismos de jogo (por exemplo, *C#*, *Python* e assim por diante). Maheshwari e Reddy (2015) propuseram uma abordagem para transpilar código *ActionScript3* legado para *Javascript*. Bouraqadi e Mason (2016) usaram transpilação para converter código *SmallTalk* em código *Javascript*. Bysiek et al. (2016) usaram transpilação como uma abordagem para converter código *Fortran* em código *Python* moderno.

Guo et al. (2012) propuseram o desenvolvimento de um processador universal para programas *RS274-D*. A metodologia proposta pelos autores utiliza a transpilação para traduzir os arquivos NC. O trabalho apresenta o conceito do NCPP (Processador de Programas de Controle Numérico), que é o módulo que faz a conversão dos códigos G em instruções a serem executadas no Controlador da CNC. O NCPP é projetado para ser capaz de tratar de uma variedade de programas *CN*. As diferentes especificações de programas são armazenadas em dicionários, que são chamados de NCSD (Dicionário de Especificações de Controle Numérico). Os dicionários são gerados pelo gerador de dicionários. O gerador aceita especificações de programas CN e converte-as em dicionários no NCSD, sendo salvos em uma biblioteca de dicionários. Um mecanismo de interpretação separado foi projetado para aceitar programas *CN* e verificar a sintaxe através do NCSD. O mecanismo funciona de maneira similar aos compiladores de computação e possui três módulos: análise léxica, análise sintática e

gerador de Funções Canônicas. A sintaxe do programa NC é representada por uma variante do *BNF*. Os autores implementaram um interpretador, em linguagem *TCL* [7], para cada comando. O mecanismo converte o arquivo NC em Funções de Usinagem Canônicas se não for detectado erros de sintaxe durante a interpretação do arquivo. O principal benefício do projeto é a flexibilidade para o processamento de diferentes tipos de programas, sendo necessário apenas ajustes no módulo NCSD para traduzir novos arquivos. O NCSD pode mudar conforme o programa *CN*. Mas o mecanismo de interpretação permanece o mesmo. Porém, o ponto negativo é que os autores preferiram criar o analisador de sintaxe manualmente, usando tecnologias como *TCL*, o que aumenta a complexidade para a implementação.

Levine (2009) pontuou que um dos aspectos-chave na compilação é dividir a tarefa em duas partes:

- Análise Léxica (ou *Scanning*).
- Análise Sintática (ou *Parsing*).

A Análise Léxica divide o texto em símbolos (conhecidos como *tokens*). A Análise Sintática, por sua vez, verifica como os símbolos devem ser arranjados para formarem sentenças válidas, de acordo com a sintaxe da linguagem.

As próximas seções irão acrescentar mais conceitos e exemplos sobre as técnicas utilizadas para a contrução do Transpilador. A seção 2.5.1 está relacionada com Análise Léxica, enquanto que a seção 2.5.2 traz conceitos relacionados com a Análise Sintática.

2.5.1 ANÁLISE LÉXICA

A Análise Léxica (ou *Scanning*) corresponde a um mecanismo de reconhecimento de padrões através de um modelo matemático conhecido como Autômato de Estado Finito (PAI T; AITHAL, 2020). O mecanismo consiste em fazer a leitura de uma cadeia de caracteres de entrada e dividi-la em símbolos, os quais possuem um significado identificado e constituem a unidade básica de uma determinada linguagem. Comparativamente, seria como dividir uma determinada sentença em um conjunto de palavras.

Uma ferramenta de software comumente utilizada para a Análise Léxica é o *Lex*, que foi projetado para o processamento léxico de um conjunto de caracteres (LESK; SCHMIDT, 1975). O *Lex* usa Expressões Regulares para definir os padrões para processar as cadeias de caracteres da entrada e dividi-las em símbolos. As Expressões

[7] Tool Command Language

Regulares são especificadas em um arquivo fonte para o Lex. As Expressões Regulares são traduzidas pelo Lex para um programa de computador em linguagem C que funciona como um Autômato de Estado Finito.

A próxima seção traz conceitos relacionados com as Expressões Regulares e apresenta a utilização dessa técnica em um trabalho desenvolvido para a intrepretação de programas *CN*.

2.5.1.1 EXPRESSÕES REGULARES

Uma Expressão Regular especifica um conjunto de caracteres em um padrão de pesquisa (LESK; SCHMIDT, 1975). Segundo Pai T e Aithal (2020), uma Expressão Regular é uma notação algébrica para descrever conjuntos de cadeias de caracteres e é usada para construir um reconhecedor para uma determinada linguagem. O conceito surgiu nos anos de 1950, quando o matemático Stephen Cole Kleene desenvolveu uma descrição formal de uma linguagem regular (KLEENE, 1951). O conceito é amplamente utilizado em programas de processamento de texto do sistema *Unix*, como o *Awk* (AHO et al., 1979).

Segundo Niemann (2018), uma Expressão Regular pode ser expressa como Autômato de Estado Finito, representado por transições entre estados. O autor aponta que há um estado inicial e um ou mais estados finais ou de aceitação. Considerar a Expressão Regular *"[0-9]+"*, que aceita cadeias de caracteres compostas apenas por dígitos. Na figura 5 o estado 0 é o estado inicial e o estado 2 é o estado de aceitação.

FIGURA 5 – AUTÔMATO DE ESTADO FINITO

FONTE: Adaptado de Niemann (2018)

O Autômato de Estado Finito transita de um estado para outro enquanto os caracteres são lidos da entrada. Enquanto o primeiro dígito é lido o Autômato de Estado Finito passa para o estado 1. Ele permanece no estado 1 enquanto mais dígitos são lidos. Quando lê um caractere diferente de um dígito ele se move para o estado 2 (o estado de aceitação). Um Autômato de Estado Finito pode ser expresso como um programa de computador (NIEMANN, 2018).

A maioria das linguagens de programação possuem suporte para Expressões Regulares. Por exemplo Python, Java e Javascript, dentre outras. Uma determinada Expressão Regular, também chamada de padrão, consiste em um conjunto de caracteres para um determinado propósito. Cada caractere da Expressão Regular pode ser um símbolo com um significado especial (metacaractere) ou um caractere lite-

ral. Por exemplo, a expressão regular *"[X][-+]?[0-9]+[.]?[0-9]*"* consiste nos seguintes componentes:

- *[X]*: um caractere *X*.
- *[-+]*: um caractere *+* ou -.
- *?*: o caractere anterior (no caso, *+* ou -) deve aparecer 0 ou 1 vez.
- *[0-9]*: um dígito (de 0 até 9).
- *+*: o caractere anterior (no caso, um dígito) deve aparecer 1 ou mais vezes.
- *[.]*: um ponto final (.).
- *?*: o caractere anterior (no caso, um ponto final) deve aparecer 0 ou 1 vez.
- *[0-9]*: um dígito (de 0 até 9).
- ***: o caractere anterior (no caso, um dígito) deve aparecer 0 ou muitas vezes.

Schroeder e Hoffmann (2006) apresentaram uma abordagem para auxiliar no processo de conversão de programas *CN* de um formato para outro. Os autores implementaram um software baseado em Expressões Regulares. As Expressões Regulares são armazenadas em um arquivo externo, em formato XML. Cada linha do arquivo *CN* é analisada pelo software para encontrar os símbolos definidos no arquivo de Expressões Regulares. Dentro da abordagem proposta pelos autores, quando uma Expressão Regular é correspondida com um símbolo, ele é removido da linha processada para evitar laços de repetições. Os símbolos não correspondentes são mostrados ao usuário. Se necessário, o usuário pode ajustar as Expressões Regulares para melhorar os resultados. O trabalho apresentou uma forma simples e flexível para converter programas *CN* de um formato para outro. Mas o algoritmo é baseado apenas em Expressões Regulares e não pode analisar comandos de várias linhas, como ocorre nos comandos de ciclo.

2.5.2 ANÁLISE SINTÁTICA

Uma linguagem significa um conjunto de cadeias de caracteres em um conjunto finito de símbolos, chamado de vocabulário da linguagem (CHOMSKY; SCHÜTZENBERGER, 1959). A Análise Sintática consiste em uma técnica para o entendimento do significado lógico ou da semântica das sentenças em uma determinada linguagem. Esse tema ganhou atenção no meio acadêmico nos anos de 1960, sendo amplamente utilizado no desenvolvimento de compiladores de linguagens de programação. Nos

anos de 1970, a Análise Sintática se tornou um campo de pesquisa bem consolidado e é uma técnica importante para o Processamento de Linguagem Natural.

O *Yacc* é uma ferramenta de software utilizada para para construir Analisadores de Sintaxe (JOHNSON; SETHI, 1990). Ele tem sido usado para implementar compiladores de linguagens de programação desde os anos de 1970. O *Yacc* gera analisadores para uma determinada especificação de linguagem que descreve a sintaxe desejada. Uma especificação para o *Yacc* consiste em uma coleção de regras gramaticais que descrevem a sintaxe de uma linguagem (JOHNSON; SETHI, 1990). A gramática para o *Yacc* é especificada usando uma variante de *BNF*. O *Yacc* transforma a especificação em um Analisador LR (JOHNSON, 1980) escrito em linguagem C.

A sub-seção 2.5.2.1 traz explicações sobre as Gramáticas Livres de Contexto. A sub-seção 2.5.2.2, por sua vez, explica o algoritmo implementado pelos Analisadores LR.

2.5.2.1 GRAMÁTICA LIVRE DE CONTEXTO

Uma Gramática Livre de Contexto (*GLC*) [8] consiste em um conjunto finito de regras (também conhecidas como produções) que descrevem como categorizar palavras ou símbolos para definir uma linguagem formal (CHOMSKY, 1959). O nome Gramática Livre de Contexto decorre do fato de que linguagens formais são definidas estritamente pelas suas regras e as suas sentenças não são influenciadas pelo contexto. Uma Gramática *BNF* [9] pode ser utilizada para expressar Gramáticas Livres de Contexto. *BNF* (Backus Naur Form) é uma meta-linguagem, que descreve sintaticamente uma linguagem de programação (MCCRACKEN; REILLY, 2003). Os autores em Naur et al. (1997) utilizaram esta técnica para descrever a linguagem *ALGOL60*. Earley (1970) observa que as Gramáticas Livres de Contexto têm sido extensivamente utilizadas para a descrição da sintaxe de linguagens de programação e linguagens naturais.

Aho e Johnson (1974) definem que, em uma Gramática Livre de Contexto, dois conjuntos de símbolos são especificados. O primeiro é o conjunto de símbolos terminais (também denominados como *tokens*). São os caracteres que formam a sentença final. As cadeias de caracteres [10] geradas por uma gramática conterão apenas símbolos terminais (AHO; JOHNSON, 1974). O segundo conjunto consiste nos símbolos não terminais. São, também, chamados de variáveis, pois se comportam como *espaços reservados* [11] para símbolos terminais.

Rodrigues e Lopes (2007) em seu trabalho a respeito da aplicação da Progra-

[8] do Inglês, *Context-free Grammar*
[9] do Inglês, *BNF Grammar*
[10] do Inglês, *strings*
[11] do Inglês, *placeholder*

mação Genética (KOZA, 1994) para a inferência de Gramáticas Livres de Contexto, apontam que uma *GLC* é definida pela quádrupla $G = (N, \sum, P, S)$, onde N é o conjunto de símbolos não terminais, $\sum$ é o conjunto de símbolos terminais (alfabeto) e P é uma lista de produções na forma $A \implies \alpha$, sendo $A \in N$ e $\alpha \in (N \cup \sum)*$.

Aho e Johnson (1974) definem uma Gramática como sendo um sistema de reescrita. Se $\alpha\mathbf{A}\gamma$ é uma sequência de símbolos gramaticais e $\mathbf{A}\rightarrow\beta$ é uma produção, então podemos escrever que $\alpha\mathbf{A}\gamma$ deriva diretamente β. Ou seja, $\alpha\mathbf{A}\gamma \Rightarrow \alpha\beta\gamma$. As produções em uma *GLC* são expressas como $\mathbf{LHS}\rightarrow\mathbf{RHS}$, onde $\mathbf{LHS}$ [12] é um único símbolo não terminal e $\mathbf{RHS}$ [13] é uma sequência de zero ou mais símbolos gramaticais.

2.5.2.2 ANALISADORES LR

Sperber e Thiemann (1995) apontam que um analisador para uma determinada *GLC* consiste em uma função que contém uma sequência de símbolos terminais ϵ como entrada e procura por uma derivação a partir de um símbolo inicial.

Earley (1970) observa que numerosos algoritmos para a análise de Gramáticas Livres de Contexto têm sido desenvolvidos. Tomita (1985) divide os algoritmos em dois grupos:

- Algoritmos para linguagens de programação.
- Algoritmos para *GLCs* arbitrárias.

Tomita (1985) indica que o segundo grupo consiste de algoritmos projetados para tratar de complexidades que não aparecem em linguagens de programação, como ambiguidade e ciclos. Esse grupo inclui o *Algoritmo Earley* (EARLEY, 1970) e o *Algoritmo CYK* (YOUNGER, 1967), dentre outros. Como exemplo de aplicação, Zulkufli et al. (2018) utilizaram uma versão modificada do *Algoritmo CYK* para a computação de cadeias de *DNA*.

O primeiro grupo de algoritmos citado por Tomita (1985) consiste de algoritmos intencionados para manipular um sub-conjunto de *GLCs* que seja suficiente para tratar de linguagens de programação. Esse grupo inclui o *Algoritmo LL* (PARR; FISHER, 2011) e o *Algoritmo LR* (KNUTH, 1965), dentre outros. Como exemplo de aplicação, Sasano e Choi (2023) utilizaram o algoritmo *LR* na implementação de um mecanismo de sugestões para autocompletar textos, conforme a sintaxe de linguagens de programação, em um software *IDE* [14].

[12] Do Inglês, Left Hand Side, Lado Esquerdo
[13] Do Inglês, Right Hand Side, Lado Direito
[14] Do Inglês, Integrated Development Environment

Aho e Johnson (1974) apontam o uso do *Algoritmo LR* para o processamento eficiente de *GLCs*. Knuth (1965) comenta que este tipo de linguagem é traduzível da esquerda para a direita. Aho e Johnson (1974) comentam que, por razões históricas, a letra *L* significa que a varredura da entrada ocorre da esquerda para a direita. A letra *R*, por sua vez, significa que a construção da derivação ocorre mais à direita. Tomita (1987) pontua que o *Algoritmo LR* corresponde a um dos algoritmos mais eficientes para a análise de gramáticas *LR*. O autor apresenta uma variação do algoritmo *LR* para a aplicação no processamento de linguagem natural.

O *Algoritmo LR* é bem descrito por Aho et al. (2007) e por Knuth (1968). Porém, Sasano e Choi (2023) fazem uma breve introdução ao funcionamento do algoritmo. Os autores apontam que a análise *LR* é do tipo *Deslocamento-Redução* [15] na qual os símbolos da *Gramática* são mantidos em uma estrutura de dados do tipo pilha e o restante dos caracteres a serem analisados permanece em uma memória *buffer* de entrada. Inicialmente, a pilha está vazia e os caracteres de entrada estão no *buffer*. Durante a análise da esquerda para a direita dos caracteres da entrada, o analisador desloca zero ou mais símbolos para a pilha até que esteja pronto para reduzir os caracteres β dos símbolos do topo da pilha. Então, o algoritmo reduz β para a produção adequada. Os autores explicam que o analisador *LR* repete esse ciclo até que a pilha contenha o símbolo inicial e a entrada esteja vazia.

Como exemplo de funcionamento do algoritmo *LR*, considerar o seguinte bloco de comando *RS274-D*: *X-15.0 Y-65.0 Z3.0*. Para que o algoritmo possa analisar uma Gramática para este exemplo, a cadeia de caracteres deve ser decomposta em símbolos (*terminais*). Isto pode ser obtido através do processamento das seguintes expressões regulares:

X = "[X][−+]?[0−9]+[.]?[0−9]∗ "
Y = "[Y][−+]?[0−9]+[.]?[0−9]∗ "
Z = "[Z][−+]?[0−9]+[.]?[0−9]∗ "

As produções para o exemplo proposto estão especificadas na TABELA 4.

[15] do Inglês, *Shift-Reduce*

TABELA 4 – PRODUÇÕES PARA O EXEMPLO

#	Production
1	E → X Y Z E
2	E → X Y E
3	E → X Z E
4	E → X E
5	E → Y Z E
6	E → Y E
7	E → Z E
8	E → \n

FONTE: **Autor (2022)**

Os termos que aparecem no *LHS* de uma produção (como o *E*, por exemplo) são os não-terminais. Os termos *X*, *Y* e *Z* são os terminais (símbolos retornados pelo algoritmo) e somente aparecem no *RHS* de uma produção. Em cada estágio o algoritmo expande um termo, substituindo o *LHS* de uma produção com o correspondente *RHS*. A TABELA 5 faz uma demonstração a respeito da análise da cadeia de caracteres com o algoritmo *Deslocamento-Redução*.

TABELA 5 – EXEMPLO DO ALGORITMO *DESLOCAMENTO-REDUÇÃO*

#	Leitura da entrada	Ação
1	⇒ X-15.0 Y-65.0 Z3.0 \n	Deslocamento
2	X-15.0 ⇒ Y-65.0 Z3.0 \n	Redução (produção 4)
3	E ⇒ Y-65.0 Z3.0 \n	Deslocamento
4	Y-65.0 ⇒ Z3.0 \n	Redução (produção 6)
5	E ⇒ Z3.0 \n	Deslocamento
6	Z3.0 ⇒ \n	Redução (produção 7)
7	E ⇒ \n	Deslocamento
8	\n ⇒	Redução (produção 8)
9	E ⇒	Accept

FONTE: **Autor (2022)**

Os termos à esquerda da seta estão na pilha, enquanto que o restante dos dados da entrada está à direita da seta. O algoritmo inicia deslocando símbolos para a pilha. Quando o elemento do topo da pilha puder ser correspondido com o *RHS* de uma produção, os símbolos do topo da pilha são substituídos pelo respectivo *LHS* da produção. Conceitualmente, os símbolos correspondentes ao *RHS* são retirados da pilha e o *LHS* da produção é colocado na pilha. Os símbolos que fazem correspondência com o *RHS* de uma determinada produção são conhecidos como manipuladores. Sendo assim, o algoritmo reduz o manipulador para o *LHS* da produção. O processo é repetido até que todos os dados de entrada tenham sido deslocados para a pilha e o não terminal inicial permaneça na pilha. No passo 1 da TABELA 5, o algoritmo desloca o símbolo X para a pilha. No passo 2, coloca a produção 3 na pilha e muda X para E. Ações ou funções podem ser executadas quando uma produção é reduzida. Esse processo

continua até que toda a entrada tenha sido analisada e a pilha contenha somente o símbolo inicial ou algum erro tenha sido encontrado.

2.6 TRADUÇÃO AUTOMÁTICA NEURAL

Até há alguns anos o desenvolvimento de sistemas de tradução automática (Machine Translation, ou MT) era, em sua maioria, implementado através de técnicas estatísticas, conhecidas como Tradutor de Máquinas Estatísticas (Statistical Machine Translation, ou SMT). Estas técnicas tornaram capaz a extração de informações implícitas a partir de *copora bilíngues*, (BROWN et al., 1993). Porém, em Maruf et al. (2021), verifica-se que as características do *SMT* eram intrínsecas à tecnologia e isto tornava estas técnicas inflexíveis. Vários trabalhos têm impulsionado o emprego de redes neurais no Processamento de Linguagem Natural (NLP), conforme pode ser verificado em Goldberg (2016). Porém, a maioria das abordagens consistiam em utilizar redes neurais como sendo componentes em sistemas *STM* tradicionais, apenas substuindo algumas partes da arquitetura, como pode ser notado em Stahlberg (2020).

O processo de *MT* tem avançado para o uso de sistemas conhecidos como Tradução Automática Neural (Neural Machine Translation, ou NMT). Estes sistemas são baseados em redes neurais que fazem a tradução das sentenças. *NMT* vem se tornando o paradigma dominante para a pesquisa e o desenvolvimento de sistemas de *MT* e tem sido utilizado em sistemas de produção, como na Google (WU et al., 2016), por exemplo.

O uso de Redes Neurais Profundas (Deep Neural Network, ou DNN) no processo de *MT* tem se destacado através dos modelos Sequência a Sequência, que fazem o uso de Redes Neurais Recorrentes (Recurrent Neural Networks, ou RNNs). A arquitetura básica destes modelos consiste em um codificador RNN que percorre cada token da sentença de origem e gera um vetor de estado com tamanho fixo. Na sequência, um decodificador RNN gera a sentença de destino, um token de cada vez, a partir do vetor de estado, como pode ser visto em Sutskever et al. (2014). Kalchbrenner et al. (2016) utilizaram uma Rede Neural Convolucional (Convolutional Neural Network, ou CNN). O codificador e o decodificador são conectados e as sequências temporais são preservadas. Os autores introduziram um mecanismo para tratar das diferenças entre o comprimento da origem e o comprimento do destino.

Outra arquitetura utilizada em sistemas NMT é o modelo Transformador, introduzido por Vaswani et al. (2017). Este modelo dispensa o uso de recorrência e convolução e utiliza um mecanismo conhecido como Atenção. Este mecanismo baseia-se em uma função que mapeia perguntas e um conjunto de pares do tipo *chave-valor* para uma saída. As perguntas, as chaves, os valores e a saída correspondem a vetores. A saída é calculada através de uma soma ponderada dos valores, aonde o peso associado

para cada valor é calculado através de uma função de compatibilidade entre a pergunta e a chave correspondente.

A Tradução Automática Neural poderia ser aplicada para a conversão dos dialetos RS274-D para as Funções Canônicas de Usinagem. Porém, vale observar que isto demandaria a obtenção de um acervo de programas escritos nos diversos dialetos para vialibilizar os processos de treinamento e teste da rede neural.

3 METODOLOGIA

O uso de Tradução Automática Neural para a interpretação dos arquivos *CN* facilitaria a extensão do sistema para novos dialetos. Pois, a tradução de novos formatos de arquivos demandaria somente o treinamento da rede neural. Porém, esta abordagem implicaria em um esforço significativo para a obtenção do conjunto de dados e os processos de preparação dos dados, treinamento, ajustes e teste da rede neural. Isto iria impor uma dificuldade adicional para o desenvolvimento do projeto, que não seria facilmente superada.

Por outro lado, a implementação de um transpilador com base em Análise Léxica e Análise Sintática dos dialetos não teria esta barreira. Porém, demandaria o ajuste do *Parser* para cada dialeto desejado. Para o presente trabalho, optou-se por desenvolver a metodologia com base na Análise Léxica e na Analise Sintática dos dialetos. Aguiar e Costa (2022) apresentaram uma metodologia semelhante para a transpilação do programa *CN* para as Funções Canônicas de Usinagem.

A abordagem propõe um algoritmo para interpretar o arquivo *CN* e convertê-lo para um formato de dados orientado a objetos, a partir do qual é possível gerar as seguintes saídas:

- Funções Canônicas de Usinagem, que consiste em um formato de arquivo simples e portável.
- Dados temporais, aonde cada movimento da máquina é registrado com uma marcação de tempo (*timestamp*).

A FIGURA 6 é um diagrama *IDEF0* [1] [2] (PRESLEY; LILES, 1995) que demonstra a visão geral do sistema projetado. [3] Os componentes indicados no diagrama serão explicados de forma mais detalhada nas próximas seções.

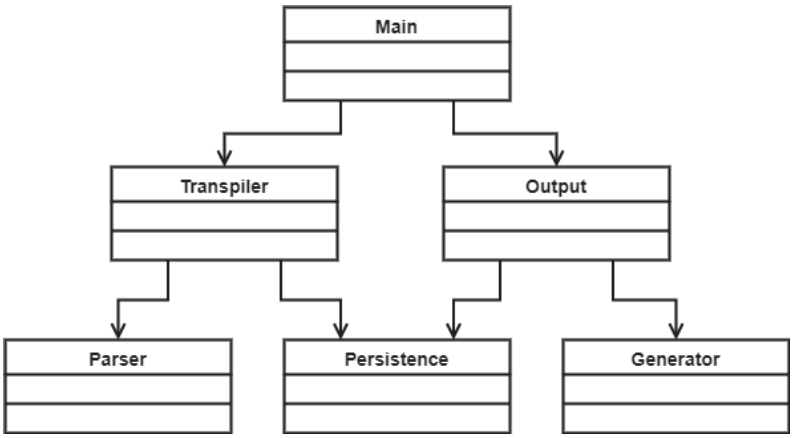
[1] IDEF0: ICAM Definition for Function Modeling
[2] ICAM: Integrated Computer Aided Manufacturing
[3] Os nomes dos componentes e das classes do sistema são apresentados em Inglês para facilitar futuras publicações do trabalho em revistas internacionais.

FIGURA 6 – VISÃO GERAL DO SISTEMA

FONTE: Autor (2023)

O projeto do software foi baseado no paradigma da Programação Orientada a Objetos (WEGNER, 1990). O diagrama da FIGURA 7 apresenta uma visão conceitual do projeto.

FIGURA 7 – DIAGRAMA DE CLASSES CONCEITUAL



FONTE: Autor (2023)

O sistema é composto por quatro componentes principais: *Main*,*Transpiler*, *Output* e *Persistence*.

O projeto pode ser implementado em uma arquitetura monolítica, na qual cada componente é implementado como uma classe dentro da mesma estrutura. Porém, a abordagem escolhida para o projeto é uma arquitetura baseada em Microserviços (DRAGONI et al., 2018), na qual cada componente é operacionalmente independente dos demais. Isto torna flexível a implantação dos componentes, conforme seja necessário oferecer escalabilidade horizontal para o sistema. Por exemplo, pode-se definir diferentes estratégias de Balanceamento de Carga (GHOMI et al., 2017) e *Cache* (LE SCOUARNEC et al., 2014) para os servidores. Os servidores podem ser implantados em um ambiente local ou na Nuvem (DILLON et al., 2010). Além disto, diferentes abordagens para replicação e redundância para o banco de dados podem ser utilizadas (COSTA et al., 2015). A comunicação entre os diferentes componentes ocorre através de interfaces de *API* (Application Programming Interface), padrão *REST* (SCHREIER,

2011), utilizando os métodos e os códigos de status definidos pelo protocolo *HTTP* (FIELDING et al., 1997).

As próximas seções irão detalhar cada um dos componentes citados. [4]

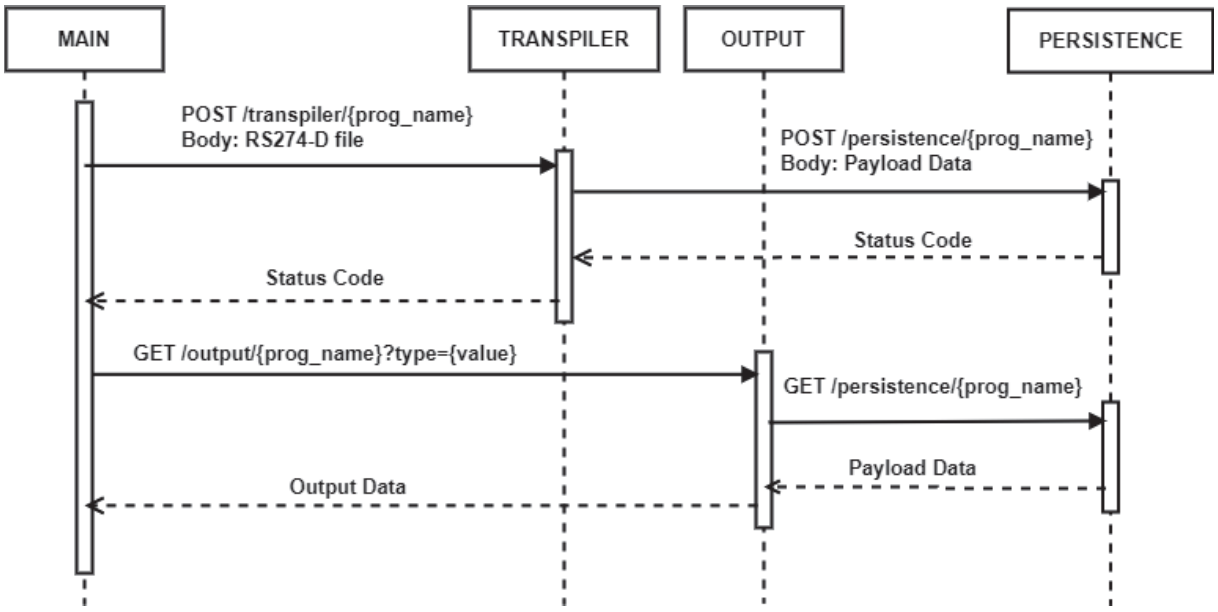
3.1 PROCESSO PRINCIPAL

O processo *MAIN* corresponde ao ponto de entrada do software. Pode ser implementado a partir de um aplicativo web ou uma interface gráfica do tipo *desktop*.

O processo *MAIN* é o processo principal do software. A transpilação do arquivo RS274-D, o processamento das funções canônicas, a extração dos dados temporais e a geração do arquivo de saída são ações disparadas a partir do processo *MAIN*.

O Diagrama de Sequência da FIGURA 8 demonstra a sequência ordenada das mensagens trocadas entre os processos.

FIGURA 8 – DIAGRAMA DE SEQUÊNCIA DO MAIN



FONTE: Autor (2023)

As duas principais funções para o usuário no processo *MAIN* são as seguintes:

- Interpretar um novo programa RS274-D e salvá-lo no banco de dados.
- Gerar um arquivo de saída (*CSV* ou *Canonical Machining Function*).

[4] Por convenção, os nomes dos módulos, processos ou instâncias serão apresentados com todas as letras maiúsculas. Os nomes das classes, por sua vez, serão apresentados somente com a primeira letra maiúscula.

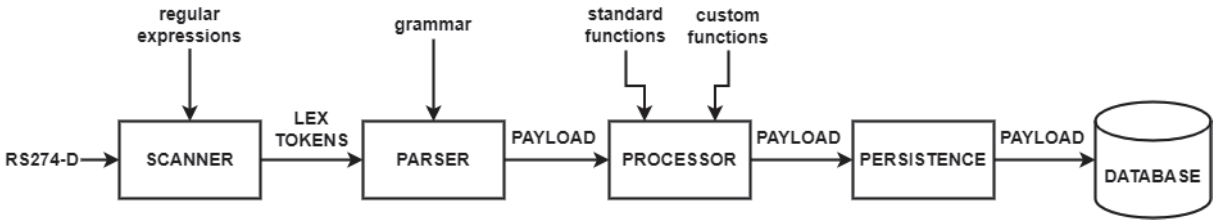
As funções do usuário podem ser implementadas através de botões na interface do usuário, que fazem requisições para as *APIs*. O diagrama da FIGURA 8 representa estas requisições através dos métodos *GET* e *POST*.

A próxima seção descreve os detalhes sobre o *TRANSPILER*.

3.2 TRANSPILADOR

O processo *TRANSPILER* é responsável por interpretar o arquivo escrito em um dialeto *RS274-D* e convertê-lo em um arquivo de dados orientado a objetos. A FIGURA 9 corresponde a um diagrama *IDEF0* que demonstra o fluxo de dados desde a leitura do arquivo *RS274-D*, passando pelos processos de transpilação até o resultado que é gravado em um banco de dados.

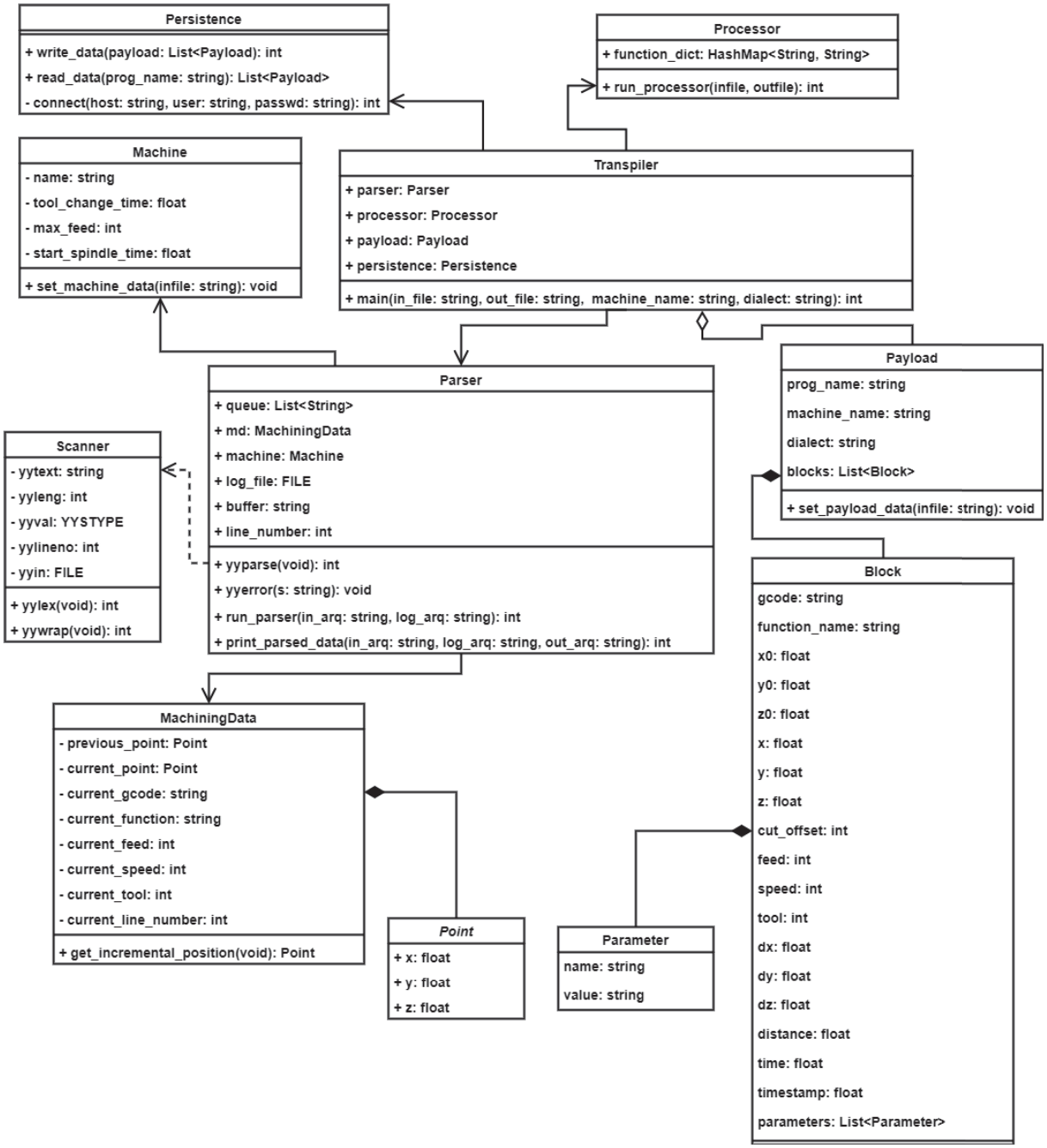
FIGURA 9 – DIAGRAMA IDEF0



FONTE: Autor (2023)

A FIGURA 10 apresenta o diagrama de classes do *TRANSPILER*.

FIGURA 10 – DIAGRAMA DE CLASSES



FONTE: Autor (2023)

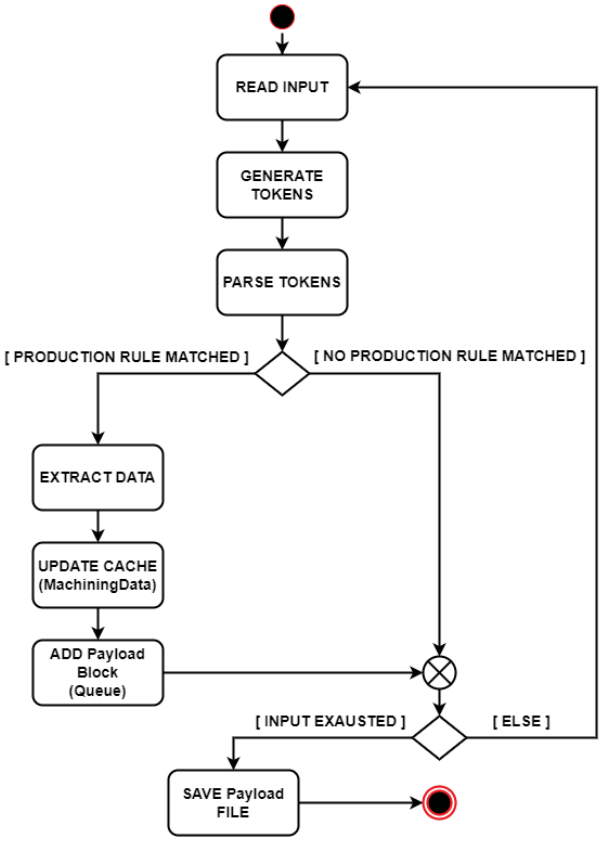
A classe *Transpiler* define referências para objetos do tipo *Parser*, *Processor* e *Payload* [5] e *Peristence* para ser capaz de desencadear o processo de interpretação do arquivo *CN* especificado no parâmetro *in_file* e gravar o resultado da interpretação em um banco de dados. A classe *Payload* modela a estrutura de dados utilizada para a

[5] *Payload*: Termo emprestado da linguagem natural, mais especificamente do setor de Transportes, que é utilizado por programadores para identificar as informações essenciais que são transportadas em integração de sistemas, conforme pode ser verificado nos trabalhos de Wittern et al. (2017) e Godefroid et al. (2020).

comunicação entre os processos do sistema e para a persistência dos dados.

O processo implementado pelo *Transpiler* é capaz de converter um arquivo *RS274-D* para um objeto do tipo *Payload*. Foi utilizada uma abordagem que emprega o uso de Análise Léxica e Análise Sintática para o desenvolvimento. O algoritmo projetado funciona como um compilador de linguagem de programação. Na prática ocorre uma transpilação, pois o arquivo escrito em um dialeto *RS274-D* é convertido para uma lista de objetos do tipo *Payload*. A FIGURA 11 apresenta uma visão geral sobre o algoritmo.

FIGURA 11 – DIAGRAMA DE ATIVIDADES DO PARSER



FONTE: Autor (2023)

O mecanismo de transpilação é baseado em um fluxo de duas etapas: o *Scanner* e o *Parser*. O primeiro processo lê a entrada (processso *READ INPUT*) e converte as cadeias de caracteres em tokens (*GENERATE TOKENS*). No diagrama da FIGURA 10, este processo é representado por uma classe chamada *Scanner*. O *Scanner* utiliza Expressões Regulares para definir padrões que são identificados na cadeia de caracteres da entrada e converte as cadeias de caracteres em símbolos.

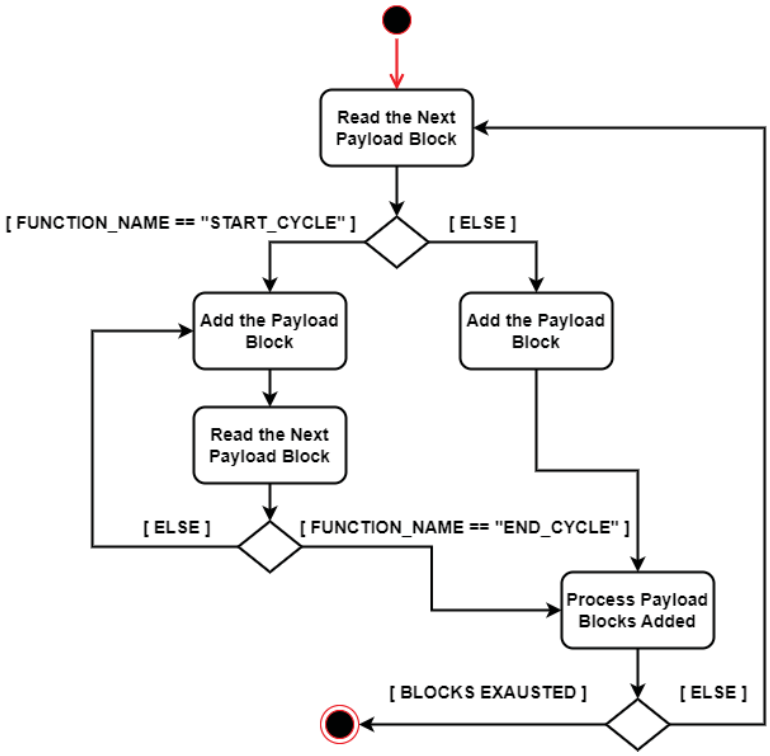
O segundo processo, por sua vez, faz a leitura dos símbolos gerados pelo *Scanner* e processa os dados de acordo com a sintaxe do dialeto *RS274-D* (*PARSE TOKENS*). No diagrama da FIGURA 10, este processo é representado pela classe chamada *Parser*. A classe *Parser* define a especificação gramatical para o dialeto,

além dos tokens que devem ser reconhecidos e as regras de produção para a tradução do dialeto *RS274-D* em objetos *Payload*. A especificação gramátical é feita em uma variação do padrão *BNF*. O *Parser* implementa um Analisador LR que processa os tokens e reduz as expressões para símbolos não-terminais. Isso é feito de acordo com a gramática especificada no *Parser*. Quando uma expressão é reduzida (*PRODUCTION RULE MATCHED*), a respectiva regra de produção é executada. Esta condição faz com que os dados sejam extraídos dos símbolos (*EXTRACT DATA*). Os dados que devem fazer parte do arquivo de saída, e que dizem respeito ao estado atual do processo de usinagem, são gravados em uma memória cache ao longo do processamento (*UPDATE CACHE*). Esta memória é representada pela classe *MachiningData* no diagrama da FIGURA 10. A classe *Machine*, por sua vez, contém os atributos da máquina-ferramenta que são importantes para o cálculo dos eventos que podem ocorrer durante a usinagem. Os dados são processados e gravados em uma estrutura de dados dinâmica, que pode ser um *Array Dinâmico* ou uma *Fila* (*ADD Payload BLOCK*). Esta estrutura de dados é representada pelo atributo *queue* da classe *Parser* no diagrama de classes da figura FIGURA 10.

Os processos implementados pelas classes *Scanner* e *Parser* atuam de forma colaborativa. O *Scanner* retorna uma cadeia de tokens que são manipulados pelo *Parser*. Cada vez que necessita de um token o *Parser* faz uma chamada para um método do *Scanner* que lê uma porção da entrada e retorna o próximo token detectado. Se os padrões não produzem um token para o *Parser*, o *Scanner* continua lendo os caracteres da entrada até que um token tenha sido detectado. Quando necessita de outro token, o *Parser* chama o método do *Scanner* novamente até que toda a entrada tenha sido processada (*INPUT EXAUSTED*) ou algum erro tenha sido detectado. O *Scanner* funciona como uma *Coroutine* (LEWIS, 2003). Cada vez em que retorna, o *Scanner* lembra do ponto em que estava para poder fazer a chamada (LEVINE, 2009).

Os movimentos e os demais eventos que ocorrerão na máquina-ferramenta durante a usinagem serão definidos pelos dados que foram extraídos pelo processo *PARSER*. O *PROCESSOR* faz o processamento desses dados, calculando os deslocamentos e as marcações de tempo para cada evento. A lista de objetos do tipo *Payload* gerada pelo *PARSER* é transmitida para o *PROCESSOR*, que é representado pela classe *Processor* no diagrama de classes da figura FIGURA 10. A saída do *PROCESSOR* é uma lista de objetos *Payload*, que definem, os eventos da usinagem, de forma granular, e as marcações de tempo em que esses eventos ocorrem. O processamento dos objetos ocorre conforme descrito na figura FIGURA 12.

FIGURA 12 – DIAGRAMA DE ATIVIDADES DO PROCESSOR



FONTE: Autor (2023)

O método *run_processor* da classe *Processor* lê cada objeto *Block* que compõem *Payload*. Neste contexto, o atributo *function_name* da classe *Payload* tem um papel importante, pois define o nome da função de usinagem conforme definido nas Funções Canônicas de Usinagem. A listagem completa das Funções Canônicas de Usinagem pode ser verificada em Proctor et al. (1997). As funções que estiverem contidas na listagem das Funções Canônicas de Usinagem são consideradas como funções padrão, sendo comuns a todos os dialetos. Por outro lado, as funcões que não estiverem contidas na listagem serão consideradas como funções customizadas, sendo definidas de forma específica pelo dialeto. Os comandos de ciclos fixos, os comandos de programação em macro e os comandos de programação parametrizada são exemplos de funções customizadas que são definidas pelo dialeto. Outro compomente importante para o processamento dos objetos *Block* é o atributo *function_dict* da classe *Processor*. O atributo corresponde a uma estrutura de dados do tipo *Hash Table* (GREEN, 2021), que consiste em pares de chave e valor, aonde cada chave corresponde ao nome de uma função de usinagem e o valor corresponde ao nome do respectivo programa que dever ser utilizado para o processamento da função. Esta abordagem permite um processo evolutivo para a interpretação dos dialetos. Pois os programas que implementam as funções customizadas podem ser desenvolvidos na medida em que sejam necessários.

Se o atributo *function_name* tiver o valor *"START_CYCLE"* significa que o

bloco corresponde ao início de um ciclo fixo ou de algum outro comando do dialeto composto por múltiplas linhas. Nestes casos, o processo vai acumulando os objetos *Block* subsequentes em uma estrutura de dados dinâmica até encontrar um objeto cujo atributo *function_name* tenha o valor *"END_CYCLE"*, o que indica que o comando foi encerrado. Os objetos *Block* que foram acumulados são enviados para serem processados pelo programa definido no atributo *function_dict*. Por outro lado, quando o atributo *function_name* tiver o valor diferente de *"START_CYCLE"*, significa que não é um comando de múltiplas linhas. Neste caso, o objeto *Block* é enviado diretamente para se processado pelo programa indicado no atributo *function_dict*.

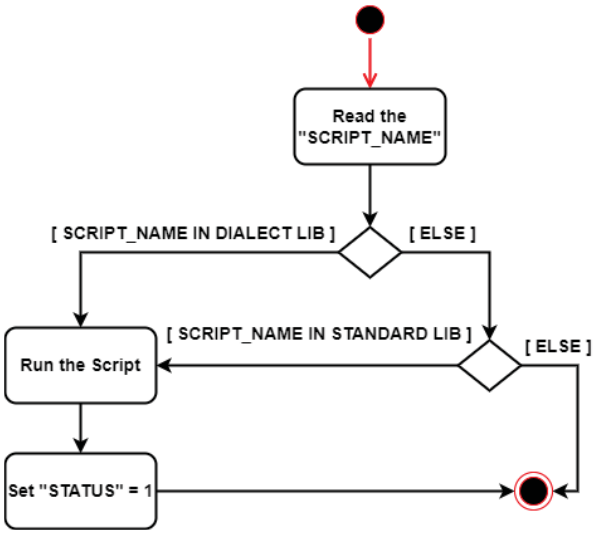
Os programas para o processamentos dos objetos *Block* são organizados em duas bibliotecas, conforme pode ser verificado no diagrama da FIGURA 13:

- *Dialect Lib*: define os programas para processamento das funções customizadas do dialeto.
- *Standard Lib*: contém os programas para o processamento das funções padrão.

Os programas para o processamento das funções customizadas são implementadas de forma independente e separada dos demais componentes do *TRANSPILER*. Isto facilita a customização e a evolução do sistema. Porém, todos os programas devem definir dois parâmetros:

- O parâmetro de entrada, que corresponde ao endereço do arquivo que contém um ou mais objetos do tipo *Block* extraídos pelo *PARSER* e que deverá ser processado pelo programa.
- O parâmetro de saída, que corresponde ao endereço do arquivo a ser gravado pelo programa, contendo o(s) objeto(s) do tipo *Block* processado(s).

FIGURA 13 – DIAGRAMA DE ATIVIDADES DO PROCESSOR



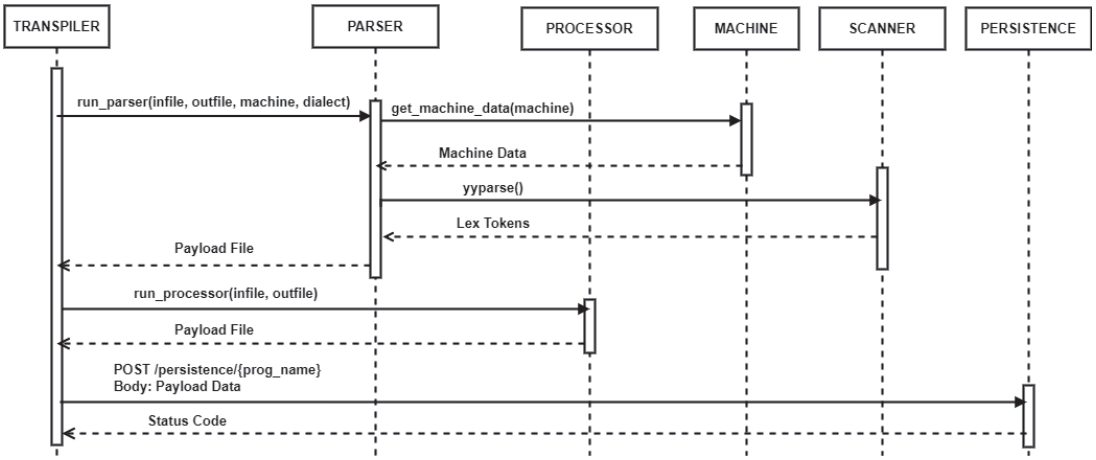
FONTE: Autor (2023)

O *PARSER* e os programas para o processamento das funções customizadas são definidos para processar um dialeto específico do padrão *RS274-D*. Desta forma, estes dois componentes coordenam o processo de transpilação, controlando a leitura e o processamento do arquivo *CN* de entrada.

Após os processos *PARSER* e *PROCESSOR* o objeto *Payload* resultante estará definido de forma completa, com todos os movimentos e os demais eventos que ocorrerão durante a usinagem e as respectivas marcações de tempo de cada evento. O passo final do *TRANSPILER* é gravar o objeto *Payload* em um banco de dados para acessos futuros. A manipulação do banco de dados é obtida através da *API PERSISTENCE*, que será descrita na próxima seção.

O diagrama da FIGURA 14 apresenta a sequência ordenada das ações executadas pelo *TRANSPILER*.

FIGURA 14 – DIAGRAMA DE SEQUÊNCIA



FONTE: Autor (2023)

O mecanismo foi projetado como uma *API* padrão *REST*, que define interfaces públicas para o processo de transpilação de programas. A TABELA 6 apresenta os principais *end-points* da *API*.

TABELA 6 – API TRANSPILER

URL	Método	Descrição
/transpiler/{prog_name}?dialect={d}&machine={m}	*POST*	*Cria novo objeto*
/transpiler/{prog_name}	*GET*	*Retorna objeto*
/transpiler/{prog_name}?dialect={d}&machine={m}	*PUT*	*Atualiza objeto*
/transpiler/{prog_name}	*DELETE*	*Apaga objeto*

FONTE: **Autor (2023)**

A próxima seção vai descrever o processo de persistência dos objetos do tipo *Payload*.

3.3 PERSISTÊNCIA DE DADOS

O *PERSISTENCE* é o processo responsável por armazenar e restaurar os dados do tipo *Payload*. Por não apresentar uma natureza relacional, a arquitetura mais adequada para o banco de dados é a *NoSQL* (DAVOUDIAN et al., 2018), mais especificamente, a arquitetura orientada a documentos.

No diagrama da FIGURA 7 o processo *PERSISTENCE* é representado pela classe *Persistence*. Assim é apresentada uma abstração do processo, que é utilizado tanto pelo *TRANSPILER* quanto pelo *OUTPUT*. Porém, o mecanismo foi concebido como uma *API* padrão *REST*, que define interfaces públicas para as operações no banco de dados. A TABELA 7 apresenta os principais *end-points* da *API*.

TABELA 7 – API PERSISTENCE

URL	Método	Descrição
/persistence/{prog_name}	*POST*	*Cria novo objeto*
/persistence/{prog_name}	*GET*	*Retorna objeto*
/persistence/{prog_name}	*PUT*	*Atualiza objeto*
/persistence/{prog_name}	*DELETE*	*Apaga objeto*

FONTE: **Autor (2023)**

A próxima seção apresenta os detalhes a respeito do processo *OUTPUT*.

3.4 GERAÇÃO DOS ARQUIVOS DE SAÍDA

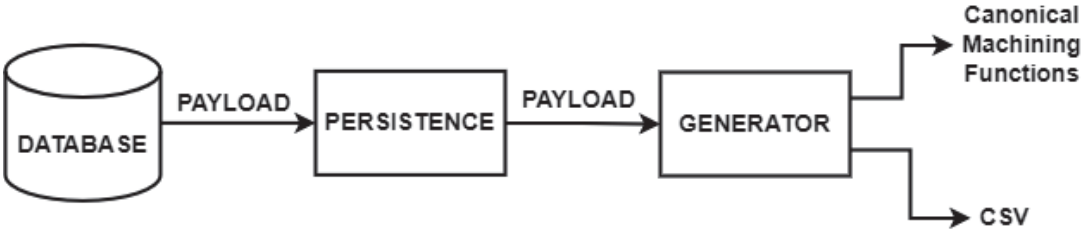
O *OUTPUT* é o processo capaz de interpretar a lista de objetos *Payload* gerada pelo *PROCESSOR* e gerar dois tipos de saída:

- Um arquivo com os dados temporais da usinagem, em formato *CSV*, para facilitar a importação do arquivo em softwares de análise de dados.

- Um arquivo em formato de Funções Canônicas de Usinagem.

A FIGURA 15 corresponde a um diagrama *IDEF0* que demonstra o fluxo desde a leitura do banco de dados (via *PERSISTENCE*) passando pelo processo *GENERATOR* até a geração do arquivo de saída.

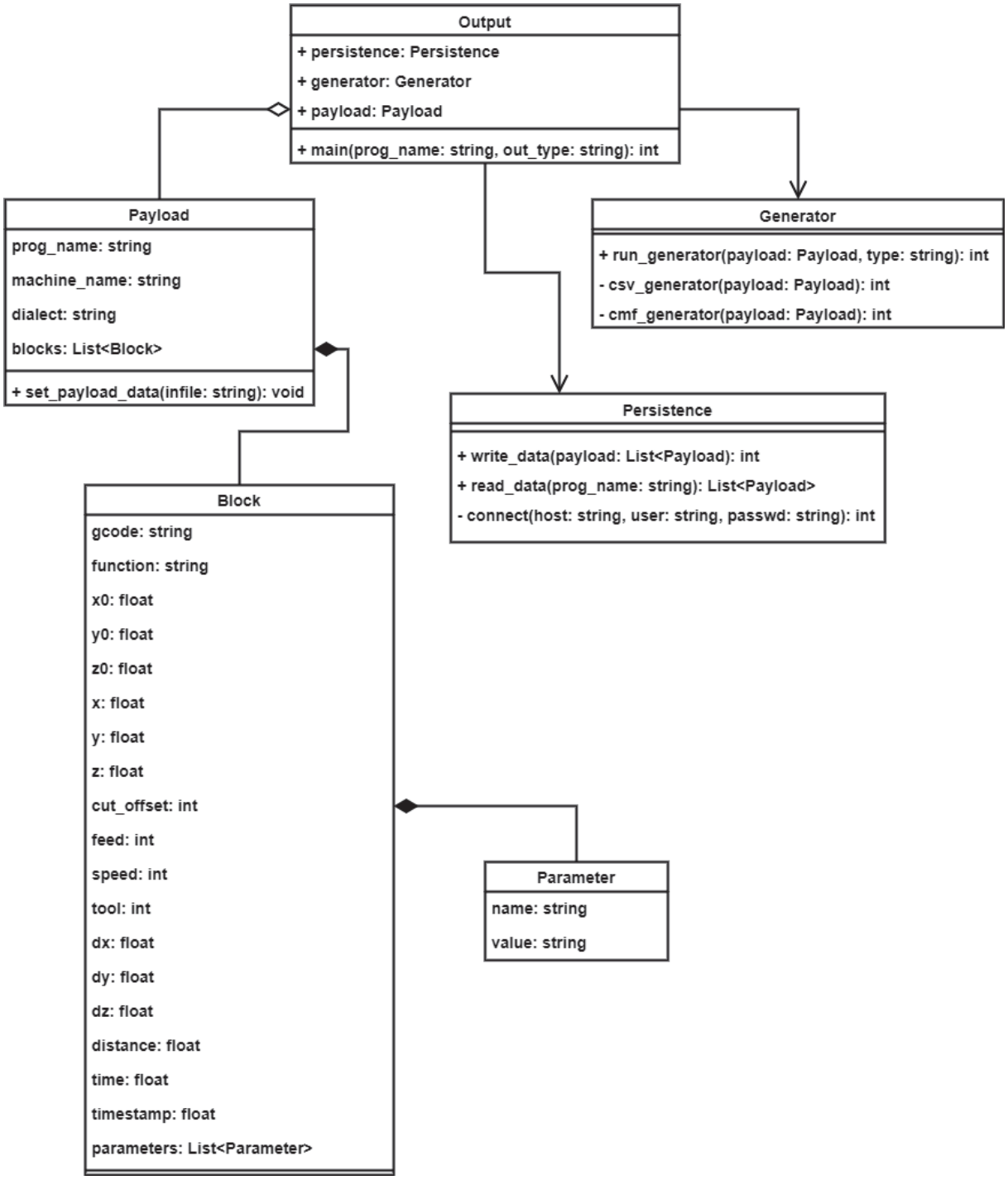
FIGURA 15 – DIAGRAMA IDEF0



FONTE: Autor (2023)

A FIGURA 16 apresenta o diagrama de classes. O processo *OUTPUT* é representado pela classe *Output*. A classe *Generator* implementa os métodos para a geração do arquivo de saída, que pode ser um arquivo de Funções Canônicas de Usinagem ou um arquivo CSV.

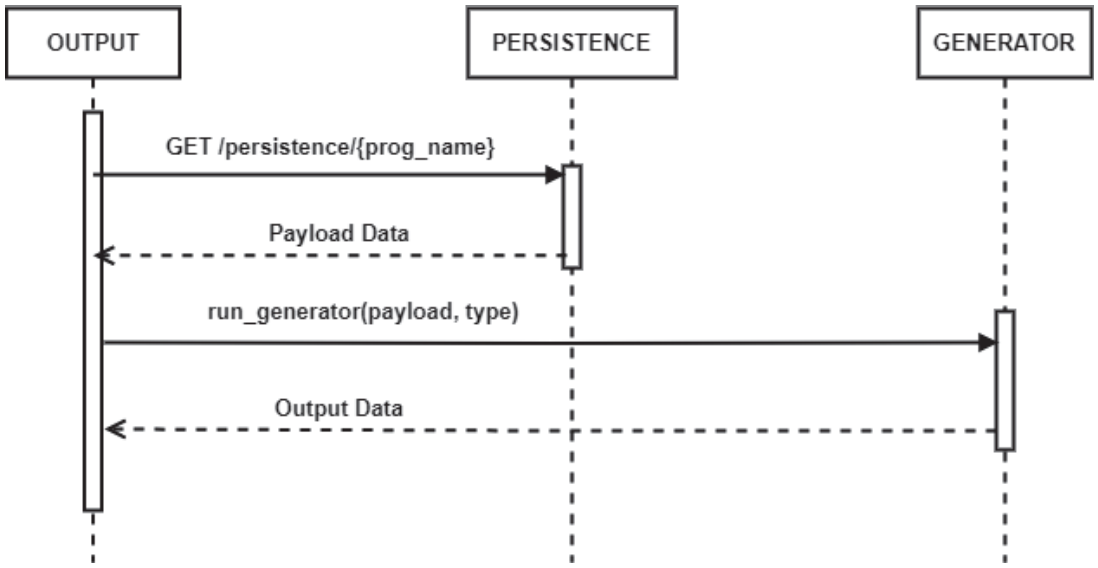
FIGURA 16 – DIAGRAMA DE CLASSES



FONTE: Autor (2023)

O diagrama da FIGURA 17 apresenta a sequência ordenada das ações executadas pelo *OUTPUT*.

FIGURA 17 – DIAGRAMA DE SEQUÊNCIA



FONTE: Autor (2023)

O processo *OUTPUT* foi projetado como uma *API* padrão *REST*. Somente um *end-point* foi definido, conforme pode ser verificado na TABELA 8. O parâmetro *type* indica o tipo de saída desejado: *csv* ou *cmf* (Canonical Machining Function). Se o valor de *type* não for especificado, assume *csv*.

TABELA 8 – API OUTPUT

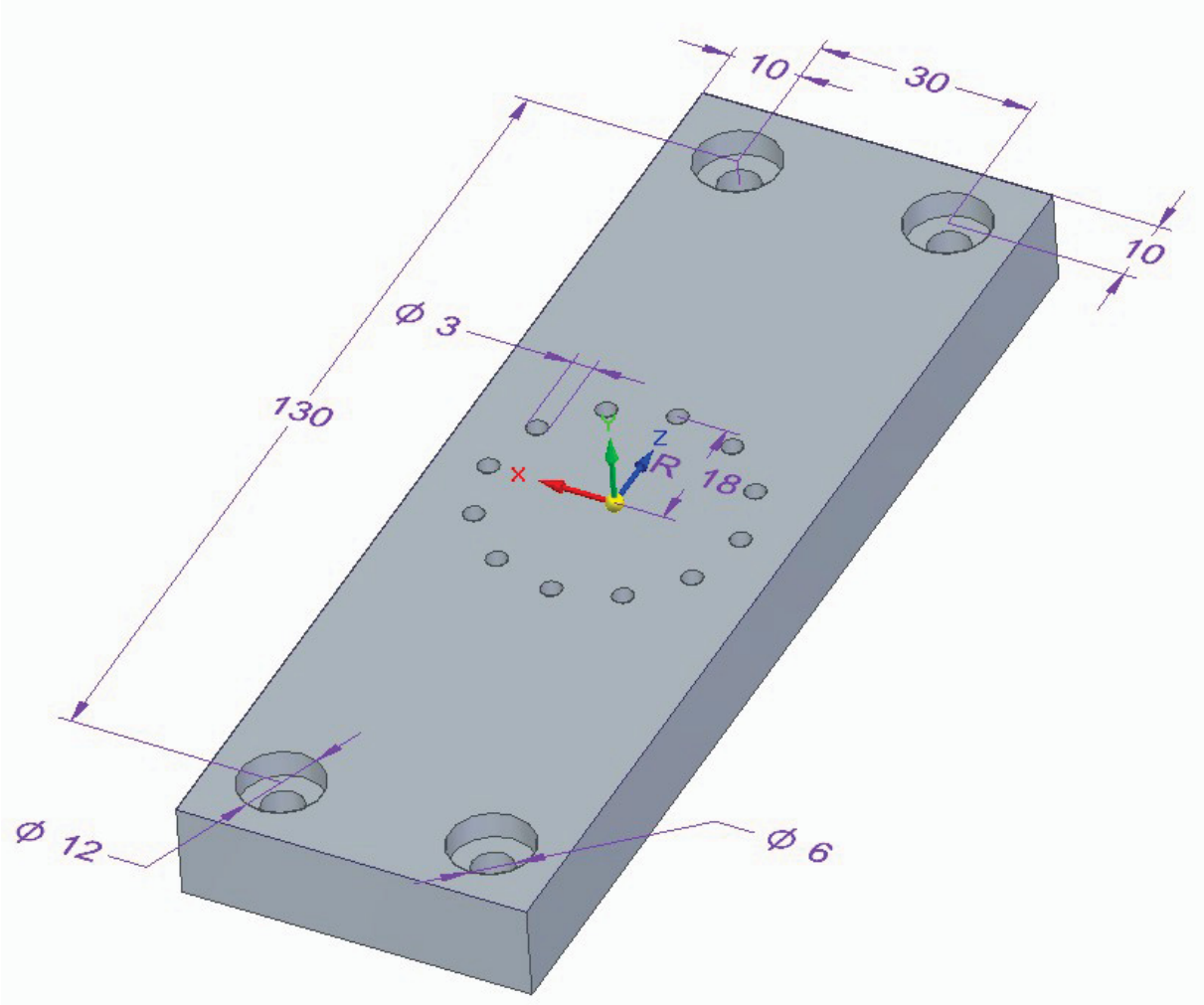
URL	**Método**	**Descrição**
/output/{prog_name}&type=value	*GET*	*Retorna a saída*

FONTE: **Autor (2023)**

4 ESTUDO DE CASO E IMPLEMENTAÇÃO

Para testar a metodologia proposta, foi preparado um experimento simples (ver FIGURA 18) para ser usinado em uma preparação única nas operações de fresamento e furação com ciclos fixos.

FIGURA 18 – TESTE DE USINAGEM



FONTE: Aguiar e Costa (2022)

O protótipo foi testado em um controlador de CNC modelo *Mach-9*, do fornecedor *Romi - Brasil*, disponível em um centro de usinagem de três eixos modelo *Discovery 4022*. O controlador *Mach-9* utiliza um dialeto do padrão *RS274-D*. O programa *CN* foi gerado no software *CAM Edgecam 2019 R2* (*Hexagon* - Suécia), que tem um pós-processador para o dialeto *Mach-9*.

A metologia proposta para o *TRANSPILER* foi implementada nas linguagens *C* e *C++*. O ambiente de desenvolvimento consistiu em um *Container Docker*, imagem

Debian Bullseye [1]. A *IDE* [2] utilizada foi o *Visual Studio Code* e o compilador utilizado foi o *GCC*. Não foi implementada a camada de banco de dados e nem a *API*. O foco da implementação foi testar o processo de transpilação e avaliar os resultados obtidos com esse processo. Para avaliar a implementação, o transpilador foi testado em um programa *CN* escrito no dialeto *Mach-9*, para a peça demonstrada na FIGURA 18. A TABELA 9 mostra uma porção do programa *CN*. O programa completo pode ser visto no APÊNDICE 1 (Capítulo 6).

TABELA 9 – TRECHO DO ARQUIVO *CN*

NÚMERO DA LINHA	**G-CODE**
1	T3M6
2	O3S1857M3
3	G0X-15.Y-65.Z3.
4	X-15.Y-65.Z5.
5	G81Z-8.R3.F186
6	G25X30.Y130.I2J2
7	G80
8	M5

FONTE: **Autor (2023)**

A implementação do *SCANNER* utilizou o software *Flex* e Expressões Regulares para extrair os símbolos. A seguir alguns exemplos de Expressões Regulares que foram definidas para o *Scanner*:

- Select tool: *[T][0]?[1-9]+*
- Change tool: *[M][0]?[6]*
- Set spindle speed: *[S][0-9]+*
- Spindle clockwise: *[M][0]?[3]*

Para a implementação do *PARSER* foi utilizado o software *Bison*. Foram definidas regras gramaticais (*grammar*) de acordo com o dialeto *Mach-9* e um algoritmo *Shift-Reduce* para processar os símbolos. A seguir alguns exemplos de regras gramaticais definidas para o *PARSER*, escritas na notação *BNF*.

```
spindle : SET_SPINDLE_SPEED
| SPINDLE_CLOCKWISE
| SPINDLE_COUNTER_CLOCKWISE
| SET_SPINDLE_SPEED SPINDLE_CLOCKWISE
| SET_SPINDLE_SPEED SPINDLE_COUNTER_CLOCKWISE
```

[1] O *Debian Bullseye* corresponde a última versão do sistema operacional *Debian Linux* no momento em que o estudo de caso foi implementado.

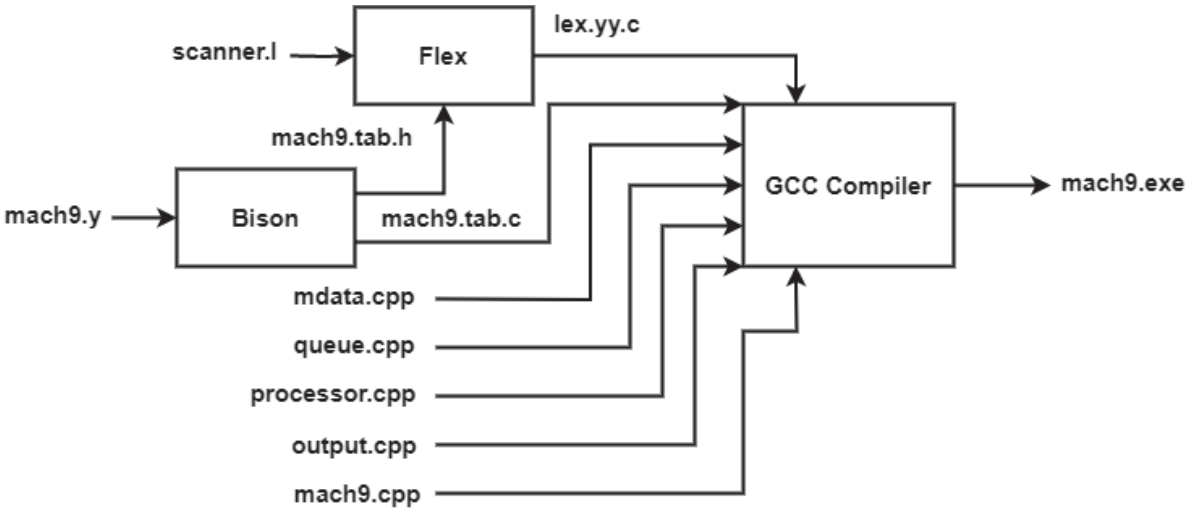
[2] Integrated Development Environment.

```
| STOP_SPINDLE_TURNING

tool : SELECT_TOOL
| SELECT_TOOL CHANGE_TOOL
```

Para os testes, foi implementada uma interface de linha de comando para executar a transpilação. A FIGURA 19 demonstra a compilação e a ligação dos códigos-fonte. O arquivo *lex.yy.c* é o código-fonte *Autômato de Estado Finito* gerado no *Flex* a partir das Expressões Regulares implementadas no arquivo *scanner.l* (classe *Scanner*). O *mach9.tab.h* é o arquivo *header* e o *mach9.tab.c* é o código-fonte do algoritmo *Analisador LR*, ambos gerados no *Bison* a partir das regras gramaticais do *Mach-9* implementadas no arquivo *mach9.y* (classe *Parser*). O arquivo *mdata.cpp* implementa a estrutura de dados e os métodos para implementar a memória *cache* que guarda dados ao longo do processamento dos *tokens* (classe *MachiningData*). O arquivo *queue.cpp* consiste no código-fonte que implementa uma *Fila*, que é uma estrutra de dados dinâmica para armazenar os dados do *Payload*. O arquivo *processor.cpp* corresponde ao código-fonte para a implementação das funções de usinagem (classe *Processor*). O arquivo *output.cpp* implementa o processo *OUTPUT*, no qual o *Payload* gerado pelo *PROCESSOR* é convertido para um arquivo *CSV* ou para um arquivo de Funções Canônicas de Usinagem. O arquivo *mach9.cpp* é o código-fonte que implementa o ponto de entrada para a execução da transpilação (classe *Transpiler*).

FIGURA 19 – COMPILAÇÃO DO TRANSPILADOR



FONTE: Autor (2023)

A compilação resultará no programa *mach9.exe*, que recebe os seguintes argumentos via linha de comando:

- O endereço do arquivo de entrada (arquivo *CN*).

- O endereço do arquivo de saída (arquivo *Payload*).
- O nome da máquina-ferramenta.
- O nome do dialeto *RS274-D*.

Os atributos da máquina-ferramenta (classe *Machine*) foram definidos no arquivo *machine.json*. A TABELA 10 apresenta os valores dos atributos da máquina-ferramenta utilizada nos testes.

TABELA 10 – ATRIBUTOS DA MÁQUINA

ATRIBUTO	**VALOR**	**DESCRIÇÃO**
name	romi	Nome da máquina
tool_change_time	7	Tempo para a troca de ferramenta em segundos
max_feed	15000	Velocidade de avanço máxima em milímetros por minuto
start_spindle_time	2	Tempo em segundos para atingir a velocidade de rotação do eixo

FONTE: **Autor (2023)**

A próxima seção apresenta os resultados obtidos com a transpilação.

5 RESULTADOS E DISCUSSÕES

Na primeira etapa da transpilação o arquivo *CN* é passado como argumento para o programa *mach9.exe*. O dialeto *RS274-D* é convertido para um *Payload* pré-processado, o qual ainda não foi submetido ao processamento das funções de usinagem. Para a persistência do *Payload* foi utilizado o formato *JSON* (BOURHIS et al., 2017), para o intercâmbio dos dados. O trecho a seguir corresponde ao resultado da primeira etapa da transpilação dos blocos 5, 6 e 7 da TABELA 9.


```
{
    "prog_name": "teste.cn",
    "machine_name": "romi",
    "dialect": "mach9",
    "blocks": [
        {
            "gcode": "G81",
            "function_name": "START_CYCLE",
            "x0": -15.0,
            "y0": -65.0,
            "z0": 5.0,
            "x1": -15.0,
            "y1": -65.0,
            "z1": 5.0,
            "cut_offset": 0,
            "feed": 186,
            "speed": 1857,
            "tool": 3,
            "dx": 0.0,
            "dy": 0.0,
            "dz": 0.0,
            "distance": 0.0,
            "time": 0.0,
            "timestamp": 0.0,
            "parameters": [
                {
                    "z": -6.0,
                    "r": 3.0,
                    "f": 191
                }
            ]
        },
        {
            "gcode": "G25",
```

```
    "function_name": "RECTANGULAR_LAYOUT",
    "x0": -15.0,
    "y0": -65.0,
    "z0": 5.0,
    "x1": -15.0,
    "y1": -65.0,
    "z1": 5.0,
    "cut_offset": 0,
    "feed": 0,
    "speed": 0,
    "tool": 3,
    "dx": 0.0,
    "dy": 0.0,
    "dz": 0.0,
    "distance": 0.0,
    "time": 0.0,
    "timestamp": 0.0,
    "parameters": [
        {
            "x": 30.0,
            "y": 130.0,
            "i": 2,
            "j": 2
        }
    ]
},
{
    "gcode": "G80",
    "function_name": "END_CYCLE",
    "x0": -15.0,
    "y0": -65.0,
    "z0": 5.0,
    "x1": -15.0,
    "y1": -65.0,
    "z1": 5.0,
    "cut_offset": 0,
    "feed": 0,
    "speed": 1857,
    "tool": 3,
    "dx": 0.0,
    "dy": 0.0,
    "dz": 0.0,
    "distance": 0.0,
    "time": 0.0,
    "timestamp": 0.0,
    "parameters": [{}]
}
```

```
    ]
}
```

Na segunda etapa da transpilação, o *Payload* é enviado para o *PROCESSOR* (*processor.cpp* da FIGURA 19) para o processamento das funções de usinagem e dos dados específicos da máquina-ferramenta (*machine.json*). O software percorre cada bloco do *Payload* e define todos os movimentos e os demais eventos que vão ocorrer durante a usinagem. Além disto, o tempo de duração de cada evento é calculado (atributo *time* do *Payload*) e são rotulados os momentos em que cada evento vai ocorrer (atributo *timestamp* do *Payload*), a partir de uma marcação inicial em zero.

Para facilitar a demonstração, o resultado final foi organizado na TABELA 11, que apresenta os dados mais relevantes da transpilação dos blocos 5, 6 e 7 da TABELA 9.

TABELA 11 – TRECHO DA TRANSPILAÇÃO DO ARQUIVO *CN*

function_name	x1	y1	z1	feed	speed	tool	time	timestamp
SELECT_TOOL	-9.0	15.59	3.0	0	0	3	7	91.191
CHANGE_TOOL	-9.0	15.59	3.0	0	0	3	0	91.191
SET_SPINDLE_SPEED	-9.0	15.59	3.0	0	1857	3	0	91.191
START_SPINDLE_CLOCKWISE	-9.0	15.59	3.0	0	1857	3	2	93.191
STRAIGHT_TRAVERSE	-15.0	-65.0	3.0	15000	1857	3	0.323	93.515
STRAIGHT_TRAVERSE	-15.0	-65.0	5.0	15000	1857	3	0.008	93.523
STRAIGHT_TRAVERSE	-15.0	-65.0	3.0	15000	1857	3	0.008	93.531
SET_FEED_RATE	-15.0	-65.0	3.0	186	1857	3	0	93.531
STRAIGHT_FEED	-15.0	-65.0	-8.0	186	1857	3	3.548	97.079
STRAIGHT_TRAVERSE	-15.0	-65.0	3.0	15000	1857	3	0.044	97.123
STRAIGHT_TRAVERSE	15.0	-65.0	3.0	15000	1857	3	0.12	97.243
SET_FEED_RATE	15.0	-65.0	3.0	186	1857	3	0	97.243
STRAIGHT_FEED	15.0	-65.0	-8.0	186	1857	3	3.548	100.791
STRAIGHT_TRAVERSE	15.0	-65.0	3.0	15000	1857	3	0.044	100.835
STRAIGHT_TRAVERSE	15.0	65.0	3.0	15000	1857	3	0.52	101.355
SET_FEED_RATE	15.0	65.0	3.0	186	1857	3	0	101.355
STRAIGHT_FEED	15.0	65.0	-8.0	186	1857	3	3.548	104.904
STRAIGHT_TRAVERSE	15.0	65.0	3.0	15000	1857	3	0.044	104.948
STRAIGHT_TRAVERSE	-15.0	65.0	3.0	15000	1857	3	0.12	105.068
SET_FEED_RATE	-15.0	65.0	3.0	186	1857	3	0	105.068
STRAIGHT_FEED	-15.0	65.0	-8.0	186	1857	3	3.548	108.616
STRAIGHT_TRAVERSE	-15.0	65.0	3.0	15000	1857	3	0.044	108.660
STOP_SPINDLE_TURNING	-15.0	65.0	3.0	0	0	3	2	110.660

FONTE: **Autor (2023)**

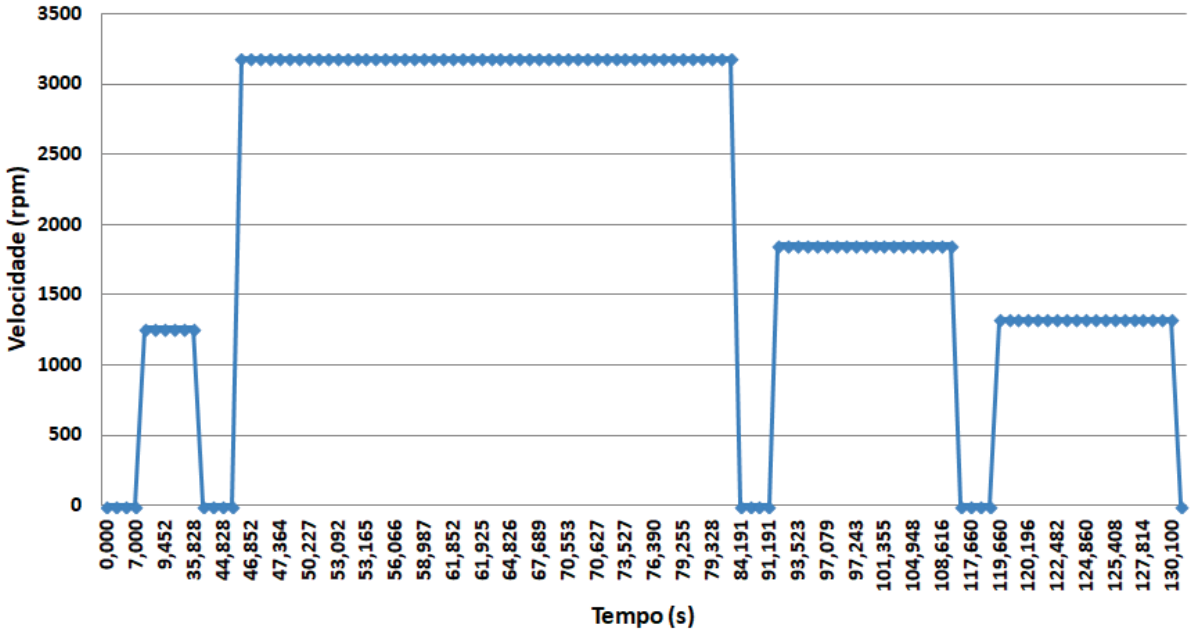
A coluna *time* da TABELA 11 corresponde ao tempo de duração do evento definido pela coluna *function_name*. A coluna *timestamp*, por sua vez, consiste na marcação de tempo para a execução de cada evento. Isto possibilita a definição do momento em que os eventos devem ocorrer durante a usinagem.

O arquivo *CN* transpilado para um arquivo *Payload* em formato *JSON* é enviado para o processo *OUTPUT*, que pode gerar um arquivo do tipo Funções Canônicas de Usinagem (para a integração com outros sistemas) e um arquivo de dados contendo os eventos que vão ocorrer durante a usinagem e o momento em que cada evento vai acontecer (para fins de análise e simulação da usinagem).

Para a análise dos resultados, foi gerado um arquivo de dados em formato *CSV* a partir da transpilação, o qual foi importado no software Microsoft Excel.

O gráfico da FIGURA 20 relaciona as mudanças na velocidade de rotação do eixo ao longo da usinagem, com base nos valores do campo *timestamp*.

FIGURA 20 – VELOCIDADE DE ROTAÇÃO AO LONGO DO TEMPO

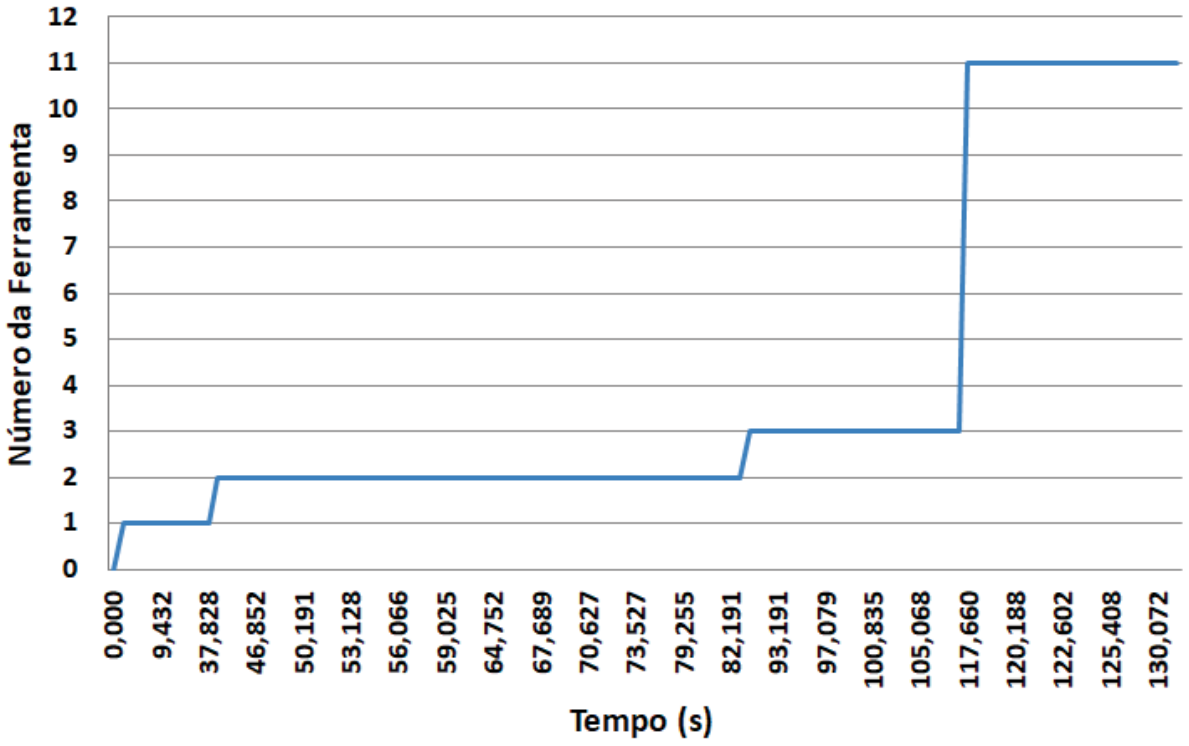


FONTE: Autor (2023)

Observa-se que no primeiro pico do gráfico, a velocidade do eixo começou a mudar a partir de 7s e atingiu 1.263 rpm em 9s. Permaneceu nessa rotação até 35,828s e voltou a 0 rpm em 37,828s. E assim sucessivamente com os demais picos. A identificação dos eventos foi obtida pela leitura dos *G-codes S* (função *SET_SPINDLE_SPEED*), *M3* (função *START_SPINDLE_CLOCKWISE*) e *M5* (função *STOP_SPINDLE_TURNING*). O tempo para atingir a velocidade de rotação (conforme definido pelos atributos da máquina na TABELA 10) foi utilizado nos cálculos da rotulação do tempo.

A FIGURA 21 apresenta as trocas de ferramentas durante a usinagem, com base nos valores do *timestamp*.

FIGURA 21 – TROCA DE FERRAMENTAS AO LONGO DO TEMPO

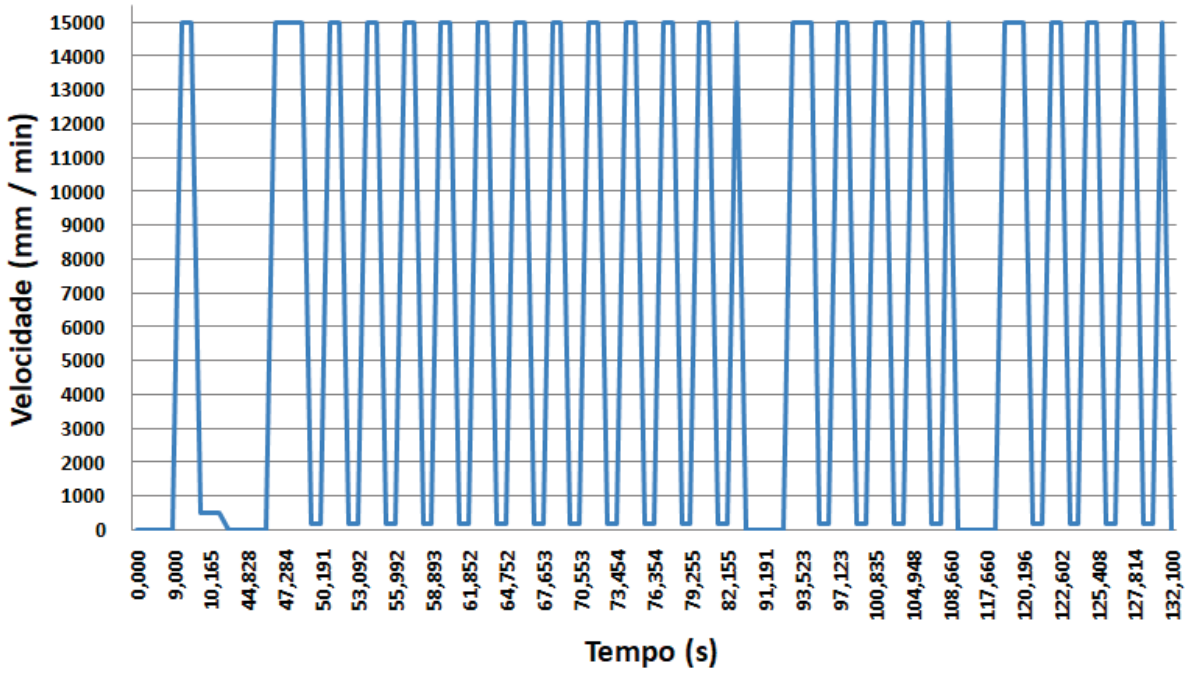


FONTE: Autor (2023)

Observa-se que a primeira troca de ferramenta ocorre em 0s e leva 7s, conforme definido na TABELA 10. Permaneceu com a ferramenta 1 até 37,828s e levou 7s para efetuar trocar para a ferramenta 2, em 44,828s. E assim sucessivamente com as demais trocas de ferramenta. Os *G-codes T* (função *SELECT_TOOL*) e *M6* (função *CHANGE_TOOL*) indicam a ocorrência destes eventos.

O gráfico da FIGURA 22 apresenta um gráfico com os valores da velocidade de avanço durante o processamento do programa *CN*.

FIGURA 22 – VELOCIDADE DE AVANÇO AO LONGO DO TEMPO



FONTE: Autor (2023)

É possível observar que em 9.432s a velocidade de avanço subiu para 15000mm/-min. Esta velocidade é uma característica da máquina utilizada nos testes, conforme definido na TABELA 10. O movimento é feito com a velocidade máxima quando o bloco é programado com o *G-code G0* (função *STRAIGHT_TRAVERSE*). Por outro lado, o *G-code G01* (função *STRAIGHT_FEED*) gera um movimento com velocidade programada, sendo definida pelo *G-code F* (função *SET_FEED_RATE*). Em 9,452s ocorre um movimento com velocidade programada de 505mm/min. Verifica-se vários pontos no gráfico com velocidade programada de 191mm/min e 186mm/min, além de vários outros pontos com a velocidade máxima do equipamento (15000mm/min). As marcações de tempo consideram as distâncias percorridas e a velocidade em que os deslocamentos são feitos.

A estrutura de dados gerada pela transpilação dos programas *CN* fornece um modelo virtual do que vai ocorrer durante a usinagem na máquina-ferramenta, levando em conta as características da máquina e o dialeto utilizado. O modelo pode ser atualizado em tempo real. Para isso, basta efetuar uma nova transpilação e um novo processamento do programa *CN*. Foram feitas simulações sobre as velocidades de rotação, as trocas de ferramenta e as velocidades de avanço. Mas outros eventos poderiam ser avaliados.

Para o cálculo dos tempos utilizados nos gráficos apresentados na FIGURA 20 e na FIGURA 22 não foi levado em conta a aceleração e o tranco (*jerk*) da máquina

Discovery 4022. Entretanto, esses parâmetros podem ser incluídos na classe *Machine*, tal como ilustrado na TABELA 10. Além disso, rotinas para cálculo do ajuste do contorno para fins de compensação do raio de corte (*CRC*) ainda não foram implementadas. Embora as *CRCs* não tenham sido utilizadas no estudo de caso aqui reportado, elas afetam os percursos de avanço e, por conseguinte, o tempo de usinagem e devem ser consideradas em implementações futuras.

A análise dos dados gerados pelo modelo podem fornecer informações valiosas para monitorar o processo e prever eventos futuros. Uma possível melhoria no modelo de simulação seria o emprego de sensores ou outros dispositivos para medir grandezas que pudessem ser relacionadas com os eventos que ocorrem na usinagem (como potência, força, entre outras).

Além da estrutrura de dados para simulação do programa *CN*, a transpilação foi capaz de gerar o arquivo de Funções Canônicas de Usinagem a partir do *Payload*. Um trecho da transpilação do programa *CN* no dialeto *Mach-9* é mostrado na TABELA 12. A coluna da esquerda contém os G-Codes originais (*Mach-9*) e a coluna da direita contém a conversão para Funções Canônicas de Usinagem. É possível observar que os comandos de Ciclos Fixos *G81* e *G25* foram convertidos para uma sequência de chamadas de funções *STRAIGHT_FEED* e *STRAIGHT_TRAVERSE*.

TABELA 12 – TRECHO DA TRANSPILAÇÃO DO ARQUIVO *CN*

Bloco *RS274-D*	**Funções Canônicas de Usinagem**
T3M6	SELECT_TOOL(3) CHANGE_TOOL(3)
O3S1857M3	SET_SPINDLE_SPEED(1857.0) START_SPINDLE_CLOCKWISE()
G0X-15.Y-65.Z3.	STRAIGHT_TRAVERSE(-15.0, -65.0, 3.0)
X-15.Y-65.Z5.	STRAIGHT_TRAVERSE(-15.0, -65.0, 5.0)
G81Z-8.R3.F186 G25X30.Y130.I2J2 G80	STRAIGHT_TRAVERSE(-15.0, -65.0, 3.0) SET_FEED_RATE(186.0) STRAIGHT_FEED(-15.0, -65.0, -8.0) STRAIGHT_TRAVERSE(-15.0, -65.0, 3.0) STRAIGHT_TRAVERSE(15.0, -65.0, 3.0) SET_FEED_RATE(186.0) STRAIGHT_FEED(15.0, -65.0, -8.0) STRAIGHT_TRAVERSE(15.0, -65.0, 3.0) STRAIGHT_TRAVERSE(15.0, 65.0, 3.0) SET_FEED_RATE(186.0) STRAIGHT_FEED(15.0, 65.0, -8.0) STRAIGHT_TRAVERSE(15.0, 65.0, 3.0) STRAIGHT_TRAVERSE(-15.0, 65.0, 3.0) SET_FEED_RATE(186.0) STRAIGHT_FEED(-15.0, 65.0, -8.0) STRAIGHT_TRAVERSE(-15.0, 65.0, 3.0)
M5	STOP_SPINDLE_TURNING()

FONTE: **Autor (2023)**

A decomposição de comandos de Ciclos Fixos em comandos detalhados é

uma característica interessante do *Payload* e das Funções Canônicas de Usinagem, obtida com a aplicação da metodologia proposta no projeto. Além disso, o arquivo das Funções Canônicas de Usinagem podem fornecer uma camada intermediária de abstração para a integração com outros sistemas, pois apresentam um formato portável e padronizado.

6 CONCLUSÃO

O presente trabalho propôs uma metologia para interpretar dialetos do formato *RS274-D* e extrair uma estrutura de dados que forneça um detalhamento a respeito dos eventos definidos no programa e a marcação de tempo em que cada um desses eventos deverá ocorrer na máquina-ferramenta.

A interpretação dos dialetos foi desenvolvida a partir de técnicas de Análise Léxica e de Análise Sintática, que foram definidas no módulo chamado de *TRANSPILER*. Os dois principais componentes do *TRANSPILER* foram chamados de *SCANNER* e *PARSER*. O *SCANNER* utiliza Expressões Regulares para extrair as palavras significativas (ou *tokens*) do arquivo. Já o *PARSER*, por sua vez, utiliza uma Gramática Livre de Contexto, codificada em uma variante do *BNF*, para definir como os símbolos devem ser arranjados em blocos de comando que sejam válidos para o dialeto. O *TRANSPILER* converte o arquivo *RS274-D* em uma estrutura de dados Orientada a Objetos que é processada pelo componente chamado *PROCESSOR*, que faz os cálculos de movimentações, marcações de tempo e trata dos ciclos fixos e outras funções customizadas do dialeto. O modelo de dados, que foi chamado de *Payload*, pode ser atualizado sempre que houver alguma mudança no programa. Para isto, basta recarregar o programa *CN* no *TRANSPILER*. O *PARSER* e o *PROCESSOR* podem ser adaptados para diferentes dialetos, o que garante flexibilidade ao sistema proposto. A arquitetura do sistema é baseada em Microserviços e contempla computação em nuvem, banco de dados orientado a documentos e APIs para a integração dos módulos. Representações abstratas dos componentes do sistema foram criadas através de diagramas UML.

Foi implementado um protótipo capaz de interpretar programas escritos no dialeto *Mach-9* do fornecedor *Romi*. A base de dados obtida pela interpretação do arquivo *CN* que forneceu um modelo virtual da usinagem da peça no equipamento CNC. Gráficos foram gerados a partir do modelo de dados e foi possível simular os momentos em que aconteceriam os seguintes eventos: as mudanças de velocidade de rotação, as trocas de ferramentas e as mudanças de velocidade de avanço.

Além da estrutrura de dados para simulação do programa *CN*, outro resultado obtido foi a geração de um arquivo de Funções Canônicas de Usinagem a partir da estrutura de dados *Payload*.

Conclui-se que a abordagem proposta foi capaz de fornecer um modelo virtual do que vai ocorrer durante a usinagem na máquina-ferramenta, levando em conta as características da máquina e o dialeto utilizado. Além disso, o arquivo das Funções

Canônicas de Usinagem podem fornecer uma camada intermediária de abstração para a integração com outros sistemas, pois apresentam um formato portável e padronizado.

Por fim, acredita-se que os seguintes trabalhos são interessantes para serem explorados futuramente:

- O desenvolvimento de um Digital Twin para o monitoramento do processo de usinagem, com o emprego de sensores ou outros dispositivos para medir grandezas que possam ser relacionadas com os eventos que ocorrem na usinagem (por exemplo, potência elétrica e força).
- A integração da base de dados, obtida pela interpretação do arquivo *CN*, com outros sistemas (ERP, por exemplo).
- O emprego de Tradução Automática Neural para a interpretação dos arquivos *CN* ao invés de Análise Léxica e Análise Sintática. Essa modificação facilitaria a extensão do sistema para novos dialetos.

REFERÊNCIAS

ADAMCZYK, B. S.; SZEJKA, A. L.; JÚNIOR, O. C. Knowledge-based expert system to support the semantic interoperability in smart manufacturing. **Computers in Industry**, Elsevier, v. 115, p. 103161, 2020. Citado 1 vez na página 12.

AGUIAR, F. R. T.; COSTA, D. D. d. **Transpilation from NC Files to Canonical Machining Functions**. [S.l.], mai. 2022. Disponível em: http://www.labusig.ufpr.br/projetos/COBEM-21-FRANCISCO.pdf. Acesso em: 1 jun. 2022. Citado 1 vezes nas páginas 37, 51.

AHO, A. V.; KERNIGHAN, B. W.; WEINBERGER, P. J. Awk - A pattern scanning and processing language. **Software: Practice and Experience**, Wiley Online Library, v. 9, n. 4, p. 267–279, 1979. Citado 1 vez na página 29.

AHO, A. V.; SETHI, R.; ULLMAN, J. D. et al. **Compilers: principles, techniques, and tools**. [S.l.]: Addison-wesley Reading, 2007. v. 2. Citado 1 vez na página 33.

AHO, A. V.; JOHNSON, S. C. LR parsing. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 6, n. 2, p. 99–124, 1974. Citado 5 vezes nas páginas 31–33.

ÁLVARES, A. J.; RODRIGUEZ, E.; JAIMES, C. I. R.; TOQUICA, J. S.; FERREIRA, J. C. STEP-NC Architectures for Industrial Robotic Machining: Review, Implementation and Validation. **IEEE Access**, IEEE, v. 8, p. 152592–152610, 2020. Citado 1 vez na página 23.

BOTKINA, D.; HEDLIND, M.; OLSSON, B.; HENSER, J.; LUNDHOLM, T. Digital Twin of a Cutting Tool. **Procedia CIRP**, v. 72, p. 215–218, 2018. 51st CIRP Conference on Manufacturing Systems. ISSN 2212-8271. DOI: https://doi.org/10.1016/j.procir.2018.03.178. Disponível em: http://www.sciencedirect.com/science/article/pii/S2212827118303378. Citado 2 vezes nas páginas 11, 18.

BOURAQADI, N.; MASON, D. Mocks, Proxies, and Transpilation as development strategies for web development. In: PROCEEDINGS of the 11th edition of the International Workshop on Smalltalk Technologies. [S.l.: s.n.], 2016. P. 1–6. Citado 1 vez na página 27.

BOURHIS, P.; REUTTER, J. L.; SUÁREZ, F.; VRGOČ, D. JSON: data model, query languages and schema specification. In: PROCEEDINGS of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems. [S.l.: s.n.], 2017. P. 123–135. Citado 1 vez na página 55.

BROWN, P. F.; PIETRA, V. J. D.; PIETRA, S. A. D.; MERCER, R. L. The Mathematics of Statistical Machine Translation: Parameter Estimation. **Comput. Linguist.**, MIT Press, Cambridge, MA, USA, v. 19, n. 2, p. 263–311, jun. 1993. ISSN 0891-2017. Citado 1 vez na página 35.

BYSIEK, M.; DROZD, A.; MATSUOKA, S. Migrating legacy Fortran to Python while retaining Fortran-level performance through transpilation and type hints. In: IEEE. 2016 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC). [S.l.: s.n.], 2016. P. 9–18. Citado 1 vez na página 27.

CAMARGO, L. G.; SOUZA, A. F. de; PODDA, L.; SCHAPPO, F.; RODRIGUES, A. R. Influencia das trajetórias de usinagem e da tolerância de cálculo no tempo real de fresamento de formas complexas. In: 5º Congresso brasileiro de engenharia de fabricação. [S.l.: s.n.], 2015. Citado 1 vez na página 11.

CAMPOS, J. G.; MIGUEZ, L. R. Standard process monitoring and traceability programming in collaborative CAD/CAM/CNC manufacturing scenarios. **Computers in Industry**, Elsevier, v. 62, n. 3, p. 311–322, 2011. Citado 1 vez na página 23.

CHEN, D.; DOUMEINGTS, G.; VERNADAT, F. Architectures for enterprise integration and interoperability: Past, present and future. **Computers in industry**, Elsevier, v. 59, n. 7, p. 647–659, 2008. Citado 1 vez na página 12.

CHOMSKY, N. On certain formal properties of grammars. **Information and control**, Elsevier, v. 2, n. 2, p. 137–167, 1959. Citado 1 vez na página 31.

CHOMSKY, N.; SCHÜTZENBERGER, M. P. The algebraic theory of context-free languages. In: STUDIES in Logic and the Foundations of Mathematics. [S.l.]: Elsevier, 1959. v. 26. P. 118–161. Citado 1 vez na página 30.

COSTA, C. H.; MAIA, P.; CARLOS, F. et al. Sharding by hash partitioning. In: PROCEEDINGS of the 17th International Conference on Enterprise Information Systems. [S.l.: s.n.], 2015. v. 1, p. 313–320. Citado 1 vez na página 38.

DANJOU, C.; LE DUIGOU, J.; EYNARD, B. Manufacturing knowledge management based on STEP-NC standard: a Closed-Loop Manufacturing approach. **International Journal of Computer Integrated Manufacturing**, Taylor & Francis, v. 30, n. 9, p. 995–1009, 2017. Citado 1 vez na página 23.

DAVOUDIAN, A.; CHEN, L.; LIU, M. A survey on NoSQL stores. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 51, n. 2, p. 1–43, 2018. Citado 1 vez na página 47.

DHARMAWARDHANA, M.; OANCEA, G.; RATNAWEERA, A. A review of STEP-NC compliant CNC systems and possibilities of closed loop manufacturing. In: IOP PUBLISHING, 1. IOP Conference Series: Materials Science and Engineering. [S.l.: s.n.], 2018. v. 399, p. 012014. Citado 4 vezes nas páginas 12, 15, 21, 23.

DILLON, T.; WU, C.; CHANG, E. Cloud computing: issues and challenges. In: IEEE. 2010 24th IEEE international conference on advanced information networking and applications. [S.l.: s.n.], 2010. P. 27–33. Citado 1 vez na página 38.

DIZDAREVIĆ, J.; CARPIO, F.; JUKAN, A.; MASIP-BRUIN, X. A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 51, n. 6, jan. 2019. ISSN 0360-0300. DOI: 10.1145/3292674. Disponível em: https://doi.org/10.1145/3292674. Citado 1 vez na página 11.

DRAGONI, N.; LANESE, I.; LARSEN, S. T.; MAZZARA, M.; MUSTAFIN, R.; SAFINA, L. Microservices: How to make your application scale. In: SPRINGER. PERSPECTIVES of System Informatics: 11th International Andrei P. Ershov Informatics Conference, PSI 2017, Moscow, Russia, June 27-29, 2017, Revised Selected Papers 11. [S.l.: s.n.], 2018. P. 95–104. Citado 1 vez na página 38.

EARLEY, J. An efficient context-free parsing algorithm. **Communications of the ACM**, ACM New York, NY, USA, v. 13, n. 2, p. 94–102, 1970. Citado 3 vezes nas páginas 31, 32.

FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; BERNERS-LEE, T. **RFC2068: Hypertext Transfer Protocol–HTTP/1.1**. [S.l.]: RFC Editor, 1997. Citado 1 vez na página 39.

GHOMI, E. J.; RAHMANI, A. M.; QADER, N. N. Load-balancing algorithms in cloud computing: A survey. **Journal of Network and Computer Applications**, Elsevier, v. 88, p. 50–71, 2017. Citado 1 vez na página 38.

GODEFROID, P.; HUANG, B.-Y.; POLISHCHUK, M. Intelligent REST API data fuzzing. In: PROCEEDINGS of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. [S.l.: s.n.], 2020. P. 725–736. Citado 1 vez na página 41.

GOLDBERG, Y. A primer on neural network models for natural language processing. **Journal of Artificial Intelligence Research**, v. 57, p. 345–420, 2016. Citado 1 vez na página 35.

GREEN, O. HashGraph—Scalable hash tables using a sparse graph data structure. **ACM Transactions on Parallel Computing (TOPC)**, ACM New York, NY, USA, v. 8, n. 2, p. 1–17, 2021. Citado 1 vez na página 44.

GUO, X.; LIU, Y.; DU, D.; YAMAZAKI, K.; FUJISHIMA, M. A universal NC program processor design and prototype implementation for CNC systems. **The International Journal of Advanced Manufacturing Technology**, v. 60, n. 5, p. 561–575, mai. 2012. ISSN 1433-3015. DOI: 10.1007/s00170-011-3618-6. Disponível em: https://doi.org/10.1007/s00170-011-3618-6. Citado 3 vezes nas páginas 21, 27.

INTERNATIONAL STANDARDS ORGANIZATION. **ISO 6983-1:2009**: Automation systems and integration — Numerical control of machines — Program format and definitions of address words — Part 1: Data format for positioning, line motion and contouring control systems. [S.l.], 2009. Citado 2 vezes nas páginas 19, 20.

JAIDER, O.; EL MESBAHI, A.; RECHIA, A.; ZARKTI, H. An automatic Feature-based tool selection approach for turning process based on data from Sandvik Coromant. In: XÈME CONFÉRENCE INTERNATIONALE : CONCEPTION ET PRODUCTION INTÉGRÉES. Tanger, Morocco: [s.n.], dez. 2015. Disponível em: https://hal.archives-ouvertes.fr/hal-01260833. Citado 1 vez na página 23.

JOHNSON, S. C. Language Development Tools on the Unix System. **IEEE Computer**, v. 13, n. 8, p. 16–21, 1980. Citado 1 vez na página 31.

JOHNSON, S. C.; SETHI, R. Yacc: a parser generator. **UNIX Vol. II: research system**, p. 347–374, 1990. Citado 2 vez na página 31.

KALCHBRENNER, N.; ESPEHOLT, L.; SIMONYAN, K.; OORD, A. v. d.; GRAVES, A.; KAVUKCUOGLU, K. Neural machine translation in linear time. **arXiv preprint arXiv:1610.10099**, 2016. Citado 1 vez na página 35.

KASSIM, N.; YUSOF, Y.; AWANG, M. Z. Reviewing iso 14649 through iso10303. **ARPN J. Eng. Appl. Sci**, v. 11, n. 10, p. 6599–6603, 2016. Citado 2 vez na página 22.

KLEENE, S. Representation of events in nerve nets and finite automata (No. RAND-RM-704). **Rand Project Air Force Santa Monica Ca**, 1951. Citado 1 vez na página 29.

KNUTH, D. E. On the translation of languages from left to right. **Information and control**, Elsevier, v. 8, n. 6, p. 607–639, 1965. Citado 2 vezes nas páginas 32, 33.

_____. Semantics of context-free languages. **Mathematical Systems Theory**, Springer-Verlag, v. 2, n. 2, p. 127–145, 1968. Cited by: 1304; All Open Access, Green Open Access. DOI: 10.1007/BF01692511. Disponível em: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0001538613&doi=10.1007%2fBF01692511&partnerID=40&md5=586b69933f2efc7a191728e83f4d Citado 1 vez na página 33.

KOZA, J. R. Genetic programming as a means for programming computers by natural selection. **Statistics and computing**, Springer, v. 4, p. 87–112, 1994. Citado 1 vez na página 32.

KRAMER, T.; PROCTOR, F. The NIST RS274/VGER Interpreter. US Department of Commerce, National Institute of Standards e Technology, set. 1998. Citado 4 vezes nas páginas 20, 21, 24, 25.

KULKARNI, R.; CHAVAN, A.; HARDIKAR, A. Transpiler and it's Advantages. **International Journal of Computer Science and Information Technologies**, Citeseer, v. 6, n. 2, p. 1629–1631, 2015. Citado 2 vez na página 27.

KUMAR, S.; NASSEHI, A.; NEWMAN, S. T.; ALLEN, R. D.; TIWARI, M. K. Process control in CNC manufacturing for discrete components: A STEP-NC compliant framework. **Robotics and Computer-Integrated Manufacturing**, Elsevier, v. 23, n. 6, p. 667–676, 2007. Citado 1 vez na página 22.

LE SCOUARNEC, N.; NEUMANN, C.; STRAUB, G. Cache policies for cloud-based systems: To keep or not to keep. In: IEEE. 2014 IEEE 7th International Conference on Cloud Computing. [S.l.: s.n.], 2014. P. 1–8. Citado 1 vez na página 38.

LESK, M. E.; SCHMIDT, E. **Lex: A lexical analyzer generator**. [S.l.]: Bell Laboratories Murray Hill, NJ, 1975. Citado 2 vezes nas páginas 28, 29.

LEVINE, J. **Flex and Bison**. Sebastopol, CA: O'Reilly Media, ago. 2009. ISBN 978-0-596-15597-1. Citado 2 vezes nas páginas 28, 43.

LEWIS, B. T. Coroutine. In: ENCYCLOPEDIA of Computer Science. [S.l.: s.n.], 2003. P. 465–466. Citado 1 vez na página 43.

LI, Y.; HUANG, Q.; HEDLIND, M.; SIVARD, G.; LUNDGREN, M.; KJELLBERG, T. Representation and exchange of digital catalogues of cutting TOOLS. In: AMERICAN SOCIETY OF MECHANICAL ENGINEERS. INTERNATIONAL Manufacturing Science and Engineering Conference. [S.l.: s.n.], 2014. v. 45806, v001t04a027. Citado 1 vez na página 18.

LIU, C.; XU, X. Cyber-physical machine tool–the era of machine tool 4.0. **Procedia Cirp**, Elsevier, v. 63, p. 70–75, 2017. Citado 1 vez na página 13.

LIU, C.; XU, X.; PENG, Q.; ZHOU, Z. MTConnect-based Cyber-Physical Machine Tool: a case study. **Procedia CIRP**, v. 72, p. 492–497, 2018. 51st CIRP Conference on Manufacturing Systems. ISSN 2212-8271. DOI: https://doi.org/10.1016/j.procir.2018.03.059. Disponível em: http://www.sciencedirect.com/science/article/pii/S2212827118301616. Citado 2 vezes nas páginas 11, 22.

LU, Y.; LIU, C.; KEVIN, I.; WANG, K.; HUANG, H.; XU, X. Digital Twin-driven smart manufacturing: Connotation, reference model, applications and research issues. **Robotics and Computer-Integrated Manufacturing**, Elsevier, v. 61, p. 101837, 2020. Citado 2 vezes nas páginas 17, 18.

MAGAMBO, S.; YING, L. The NC machining post-processing technology based on UG. **International Journal of Science and Research (IJSR)**, Citeseer, v. 2, n. 9, p. 131–134, 2013. Citado 1 vez na página 12.

MAHESHWARI, Y.; REDDY, Y. R. Transformation of flash files to html5 and javascript. In: PROCEEDINGS of the ASWEC 2015 24th Australasian Software Engineering Conference. [S.l.: s.n.], 2015. P. 23–27. Citado 1 vez na página 27.

MARUF, S.; SALEH, F.; HAFFARI, G. A Survey on Document-Level Neural Machine Translation: Methods and Evaluation. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 54, n. 2, mar. 2021. ISSN 0360-0300. DOI: 10.1145/3441691. Disponível em: https://doi.org/10.1145/3441691. Citado 1 vez na página 35.

MCCRACKEN, D. D.; REILLY, E. D. Backus-Naur Form (BNF). In: ENCYCLOPEDIA of Computer Science. GBR: John Wiley e Sons Ltd., 2003. P. 129–131. ISBN 0470864125. Citado 1 vez na página 31.

MTCONNECT-INSTITUTE. **MTConnect Standard**. [S.l.], abr. 2021. Disponível em: https://www.mtconnect.org. Acesso em: 15 abr. 2021. Citado 1 vez na página 24.

NASSEHI, A.; NEWMAN, S. T.; XU, X. W.; ROSSO JR, R. Toward interoperable CNC manufacturing. **International Journal of Computer Integrated Manufacturing**, Taylor & Francis, v. 21, n. 2, p. 222–230, 2008. Citado 1 vez na página 13.

NAUR, P.; BACKUS, J. W.; BAUER, F. L.; GREEN, J.; KAFZ, C.; MCCARTHY, J.; PERLIS, A. J.; RUTISHAUSER, H.; SAMELSON, K.; VAUQUOIS, B. et al. Revised Report on the Algorithmic Language Algol 60. In: ALGOL-LIKE Languages. [S.l.]: Springer, 1997. P. 19–49. Citado 1 vez na página 31.

NAVAS, C. F. E.; YEPES, A. E.; ABOLGHASEM, S.; BARBIERI, G. MTConnect-based decision support system for local machine tool monitoring. **Procedia Computer Science**, Elsevier, v. 180, p. 69–78, 2021. Citado 1 vez na página 22.

NIEMANN, T. A Compact GUIDE TO LEX & YACC. **Portland, Oregon**, 2018. Citado 2 vez na página 29.

NUNNARI, F.; HELOIR, A. Write-once, transpile-everywhere: re-using motion controllers of virtual humans across multiple game engines. In: SPRINGER. INTERNATIONAL Conference on Augmented Reality, Virtual Reality and Computer Graphics. [S.l.: s.n.], 2018. P. 435–446. Citado 1 vez na página 27.

ODENDAHL, D.; VENKATESH, S.; MICHALOSKI, J.; PROCTOR, F. Standardization of auxiliary equipment for next generation CNC machining. In: ISA EXPO Technical Conference,(Houston, TX). [S.l.: s.n.], 2008. Citado 1 vez na página 24.

OPC-FOUDATION. **OPC Unified Architecture**. [S.l.], abr. 2021. Disponível em: https://reference.opcfoundation.org/v104/MachineTool/v100/docs. Acesso em: 15 abr. 2021. Citado 1 vez na página 24.

PAI T, V.; AITHAL, P. A Systematic Literature Review of Lexical Analyzer Implementation Techniques in Compiler Design. **International Journal of Applied Engineering and Management Letters (IJAEML)**, v. 4, n. 2, p. 285–301, 2020. Citado 2 vezes nas páginas 28, 29.

PARR, T.; FISHER, K. LL(*): The foundation of the ANTLR parser generator. **Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)**, p. 425–436, 2011. Cited by: 161. DOI: 10.1145/1993498.1993548. Disponível em: https://www.scopus.com/inward/record.uri?eid=2-s2.0-79959911655&doi=10.1145%2f1993498.1993548&partnerID=40&md5=25c0fc8883b8c0696a3fe7c Citado 1 vez na página 32.

POBOZNIAK, J.; SOBIESKI, S. Extension of STEP-NC data structure to represent manufacturing process structure in CAPP system. **Procedia Manufacturing**, Elsevier, v. 11, p. 1692–1699, 2017. Citado 0 vez na página 23.

PRESLEY, A.; LILES, D. H. The use of IDEF0 for the design and specification of methodologies. In: PROCEEDINGS of the 4th industrial engineering research conference. [S.l.: s.n.], 1995. P. 442–448. Citado 1 vez na página 37.

PROCTOR, F. M.; PROCTOR, F. M.; MICHALOSKI, J. L. Canonical machining commands. US Department of Commerce, National Institute of Standards e Technology, 1997. Citado 3 vezes nas páginas 25, 44.

RABELO, R. J.; MAGALHÃES, L. C.; CABRAL, F. G. Uma proposta de arquitetura de referência de gêmeo digital para sistemas ciberfísicos em um cenário de indústria 4.0. In: 1. CONGRESSO Brasileiro de Automática-CBA. [S.l.: s.n.], 2020. v. 2. Citado 1 vez na página 17.

RODRIGUES, E.; LOPES, H. S. Inferência de gramáticas livres de contexto usando programação genética. **I Simpósio Brasileiro de Inteligência Computacional**, p. 26, 2007. Citado 1 vez na página 31.

SASANO, I.; CHOI, K. A text-based syntax completion method using LR parsing and its evaluation. **Science of Computer Programming**, Elsevier B.V., v. 228, 2023. Cited by: 0. DOI: 10.1016/j.scico.2023.102957. Disponível em: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85156143578&doi=10.1016%2fj.scico.2023.102957&partnerID=40&md5=6d3353d4f99391761276 Citado 2 vezes nas páginas 32, 33.

SCHREIER, S. Modeling restful applications. In: PROCEEDINGS of the second international workshop on restful design. [S.l.: s.n.], 2011. P. 15–21. Citado 1 vez na página 38.

SCHROEDER, T.; HOFFMANN, M. Flexible automatic converting of NC programs. A cross-compiler for structured text. **International Journal of Production Research**, Taylor & Francis, v. 44, n. 13, p. 2671–2679, 2006. DOI: 10.1080/00207540500455841. eprint: https://doi.org/10.1080/00207540500455841. Disponível em: https://doi.org/10.1080/00207540500455841. Citado 1 vez na página 30.

SEMERARO, C.; LEZOCHE, M.; PANETTO, H.; DASSISTI, M. Digital twin paradigm: A systematic literature review. **Computers in Industry**, Elsevier, v. 130, p. 103469, 2021. Citado 4 vezes nas páginas 17, 18.

SPERBER, M.; THIEMANN, P. The essence of LR parsing. In: PROCEEDINGS of the 1995 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation. [S.l.: s.n.], 1995. P. 146–155. Citado 1 vez na página 32.

STAHLBERG, F. Neural machine translation: A review. **Journal of Artificial Intelligence Research**, v. 69, p. 343–418, 2020. Citado 1 vez na página 35.

SUTSKEVER, I.; VINYALS, O.; LE, Q. V. Sequence to sequence learning with neural networks. **Advances in neural information processing systems**, v. 27, 2014. Citado 1 vez na página 35.

THEORIN, A.; BENGTSSON, K.; PROVOST, J.; LIEDER, M.; JOHNSSON, C.; LUNDHOLM, T.; LENNARTSON, B. An event-driven manufacturing information system

architecture for Industry 4.0. **International journal of production research**, Taylor & Francis, v. 55, n. 5, p. 1297–1311, 2017. Citado 1 vez na página 18.

TOMITA, M. An efficient augmented-context-free parsing algorithm. **Computational linguistics**, v. 13, p. 31–46, 1987. Citado 1 vez na página 33.

_____. An Efficient Context-Free Parsing Algorithm for Natural Languages. v. 85, p. 756–764, 1985. Citado 3 vez na página 32.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, Ł.; POLOSUKHIN, I. Attention is all you need. **Advances in neural information processing systems**, v. 30, 2017. Citado 1 vez na página 35.

WEGNER, P. Concepts and paradigms of object-oriented programming. **ACM Sigplan Oops Messenger**, ACM New York, NY, USA, v. 1, n. 1, p. 7–87, 1990. Citado 1 vez na página 38.

WITTERN, E.; YING, A. T.; ZHENG, Y.; DOLBY, J.; LAREDO, J. A. Statically checking web API requests in JavaScript. In: IEEE. 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). [S.l.: s.n.], 2017. P. 244–254. Citado 1 vez na página 41.

WU, Y.; SCHUSTER, M.; CHEN, Z.; LE, Q. V.; NOROUZI, M.; MACHEREY, W.; KRIKUN, M.; CAO, Y.; GAO, Q.; MACHEREY, K. et al. Google's neural machine translation system: Bridging the gap between human and machine translation. **arXiv preprint arXiv:1609.08144**, 2016. Citado 1 vez na página 35.

YOUNGER, D. H. Recognition and parsing of context-free languages in time n3. **Information and Control**, v. 10, n. 2, p. 189–208, 1967. ISSN 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(67)80007-X. Disponível em: https://www.sciencedirect.com/science/article/pii/S001999586780007X. Citado 1 vez na página 32.

ZAKAI, A. Emscripten: an LLVM-to-JavaScript compiler. In: PROCEEDINGS of the ACM international conference companion on Object oriented programming systems languages and applications companion. [S.l.: s.n.], 2011. P. 301–312. Citado 1 vez na página 27.

ZHANG, X.; NASSEHI, A.; NEWMAN, S. T. A meta-model of computer numerical controlled part programming languages. **Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture**, Sage Publications Sage UK: London, England, v. 229, n. 7, p. 1243–1257, 2015. Citado 5 vezes nas páginas 12, 14, 15, 21, 24.

ZULKUFLI, N. L. M.; TURAEV, S.; TAMRIN, M. I. M.; MESSIKH, A. Watson-crick context-free grammars: Grammar simplifications and a parsing algorithm. **Computer Journal**, Oxford University Press, v. 61, n. 9, p. 1361–1373, 2018. Cited by: 2. DOI: 10.1093/comjnl/bxx128. Disponível em: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85055204433&doi=10.1093%2fcomjnl%2fbxx128&partnerID=40&md5=bf9fb14f32f279d9acd32d2b Citado 1 vez na página 32.

APÊNDICE 1 - PROGRAMA CN COMPLETO UTILIZADO NO ESTUDO DE CASO

```
;FACEAMENTO:
G99
G90
G71
G17
G66
GZ0
T1M6
O1S1263M3
GX0.Y108.
Z5.
G1Z-1.F505
Y-108.
M5
GZ0
;FURAÇÃO 3mm :
T2M6
O2S3183M3
GZ5.
GXY
G81Z-6.R3.F191
G24XY18.B30.L12
G80
M5
GZ0
;FURAÇÃO 6mm
T3M6
O3S1857M3
G0X-15.Y-65.
Z5.
G81Z-8.R3.F186
G25X30.Y130.I2J2
G80
M5
GZ0
;REBAIXO FRESA 12mm
T11M6
O11S1326M3
G0X-15.Y-65.
Z5.
G81Z-4.R3.F199
G25X30.Y130.I2J2
G80
```

M5
GZO
M2