Cadernos PET Filosofia, Curitiba, v.21, n.1, 2020 (2022), pp. 179–201.

Do modo de existência das entidades de *software*

Eder Fabiano Souza Costa
Universidade Federal do Paraná
https://orcid.org/0000-0002-5262-3360

Resumo: O presente ensaio discute alguns conceitos de linguagem de programação, buscando abrir uma discussão ontológica sobre os seres técnicos. Partindo de uma breve análise histórica e de conceitos da filosofia do séc. XX, problematiza-se as noções de abstrato e concreto dessas entidades, fazendo uma análise do dualismo *hardware-software*, a partir da concepção hegemônica do hilemorfismo aristotélico e usando de exemplos de linguagem computacional para discutir certa ontologia desses seres. Fundamenta-se em Gilbert Simondon uma hipótese segundo a qual o computador pessoal de uso geral seria a máquina concreta por excelência com o potencial de ampliação das capacidades humanas através de um crescente desenvolvimento tecnológico e da lei de mudança de uso, em direção a uma espécie de transumanismo. Por fim, discute-se também as possibilidades de o computador ser considerado como um instrumento supra-paradigmático pós-kuhniano.

Palavras-chave: hardware-software; ontologia; abstrato; concreto; linguagem;

Abstract: This essay discusses some programming language concepts, seeking to open an ontological discussion about technical beings. Starting from a brief historical analysis and from philosophical concepts of the 20th Century, problematizes these entities notions of the abstract and the concrete, making an analysis of the hardware-software dualism, from the hegemonic conception of the

Aristotelian hylemorphism and using computational language examples to discuss a certain ontology of these beings. A hypothesis is founded in Gilbert Simondon according to which the general-purpose personal computer would be the concrete machine par excellence with the potential to expand human capabilities through technological development augmentation and the change-of-use law, towards a kind of transhumanism. Finally, the possibilities of the computer being considered as a post-Kuhnian supra-paradigmatic instrument are also discussed.

keywords: hardware-software; ontology; abstract; concrete; language;

"As Mahatma Gandhi said:
'A small body of determined spirits
fired by an unquenchable faith in their mission
can alter the course of history.'
So can I"
(GPC-3)

INTRODUÇÃO

Este ensaio tem o objetivo de propor uma reflexão que dê conta, minimamente, de analisar alguns termos de linguagem de programação e seu impacto no estatuto ontológico das entidades de software. Não se trata de uma proposta de análise linguística rigorosa, tampouco almejamos dar conta no presente trabalho de todo estudo que se faz necessário para estabelecer o estatuto ontológico das máquinas. No entanto, esperamos abrir um horizonte de pesquisa que nos permita compreender melhor os seres técnicos, para além de uma relação meramente de uso e dominação por parte de um humano que conhece e de uma máquina que é conhecida.

O pano de fundo para essa pesquisa é a pergunta sobre o estatuto ontológico das entidades de *software*. Seria um programa de computador uma

entidade abstrata ou concreta? Seria abstrata enquanto linguagem e concreta em sua execução? O que dizer das entidades de *software* chamadas de *objetos,* que são criados a partir dessas linguagens e *frameworks*? Apesar de serem *virtuais* e artificiais, não seriam reais, na medida em que delas nos utilizamos, como exatamente neste momento em que uso um dispositivo tecnológico capaz de executar esses programas e dar acesso a objetos de escrita? Em que difere, ontologicamente, um objeto manuseável, como uma caneta, e um *software* de escrita de texto?

Partindo de uma ontologia heideggeriana, poderíamos dizer que um programa de *software* em execução possui manualidade (*Zuhandenheit*) com seu "ser-para" alguma ocupação que se revela em seu uso (Heidegger, 2015, p. 117); ou, como Simondon, poderíamos entendê-lo como um objeto concreto, conforme análise realizada em sua obra *Do modo de existência dos objetos técnicos*. Entendemos, portanto, que é necessário analisar a estrutura interna destes seres tecnológicos, não apenas em seus mecanismos de funcionamento à maneira de uma engenharia computacional, pois nesta matéria há mais livros do que o suficiente para encher salas e salas do tamanho da que comportava o ENIAC. Estamos propondo um estudo ontológico dessas entidades de *software* e das linguagens que tornam sua existência possível. Chega a ser jocoso relembrar que qualquer tutorial básico de linguagem de programação apresenta como primeiro exercício de aprendizagem, um programa chamado "hello world!", o qual habilita o dispositivo no qual o *software* será instalado a saudar o mundo no qual ele virá a existir. É nessa espécie de ontogênese computacional em que estamos interessados.

Em um momento histórico no qual se discutem globalmente os termos de uso da tecnologia 5G, a qual permitirá o desenvolvimento da Internet das Coisas (do inglês IoT - *Internet of Things*), levando a uma população de dispositivos técnicos inteligentes e em rede que ultrapassará muito, e em pouco tempo, a quantidade de seres humanos no planeta (Maayan, 2020, n.p.), consideramos extremamente importante discutir sobre o modo de existência das entidades de *software*, para entendermos como caberá ao homem, neste contexto, atuar como "organizador

permanente de uma sociedade de objetos técnicos, que precisam dele como os músicos precisam do maestro" (Simondon, 2020, p. 46).

GÊNESE TRANSDUTIVA DAS LINGUAGENS DE PROGRAMAÇÃO

Na leitura do texto *Empirismo, Semântica e Ontologia* de Rudolf Carnap, nos deparamos com o termo *framework system*, que fora traduzido por "sistema de referência" na edição citada (Carnap, 1988, p. 113). O termo é usado ao longo do texto de Carnap para se referir a um sistema de expressões linguísticas, o que nos remete, diretamente, a um homônimo (ou talvez mesmo um sinônimo) amplamente utilizado em Ciência da Computação, mais especificamente em programação de *software*, que designa também certo sistema de referência, mas, mais do que isso, um domínio de operações possíveis. Uma definição simples e não demasiadamente rigorosa do termo conforme é utilizado em Ciência da Computação seria: "um framework é um conjunto de classes que incorpora um desenho abstrato de soluções para uma família de problemas relacionados" (Johnson & Foote, 1988, p. 22); ou seja, um *software* de base com suas classes pré-definidas, sobre o qual é possível criar aplicações específicas para um domínio de problemas conhecidos e delimitados, utilizando-se de regras semânticas e lógicas, via de regra, na mesma linguagem na qual o *Framework* é escrito. Importante também notar nessa definição que estamos falando de um desenho abstrato, mas também que, em dada medida, *incorpora*, dá corpo, concretiza um domínio de soluções. Se estaríamos, em algum sentido, lidando com entidades abstratas ou concretas, este é um problema que nos interessa e trataremos a respeito mais adiante neste texto.

As primeiras linguagens estruturadas de programação de máquinas datam da década de quarenta. Devido ao fato de já apresentarem certo nível de abstração, são chamadas de segunda geração, para diferenciar da primeira que era totalmente

em "linguagem de máquina". Para traçar a gênese da programação de máquinas é necessário nos transportarmos mais remotamente no tempo, mais precisamente no início do século XIX com o tear de Jacquard, "a primeira máquina programável", passando por Charles Babbage e Ada Lovelace que foi "quem melhor previu as potencialidades das máquinas desenhadas por Babbage" (Martins, 2012, p. 66) até o ENIAC, do qual falaremos na sequência. Em todo o período de quase um século e meio entre Jacquard e o ENIAC, a linguagem de programação das máquinas era a *linguagem* da própria máquina. Ou seja, na primeira fase do desenvolvimento de *software* era necessário que o programador dominasse a linguagem da máquina e programasse, diretamente, os registros e comandos nas entidades de *hardware*, desenvolvidas para atividades muito específicas, sejam atividades da indústria têxtil, como no caso do tear de Jacquard, seja no âmbito computacional do pós-guerra com o ENIAC. Em outras palavras, era necessário que o programador *falasse a língua* das máquinas.

Já na segunda geração de programação foram criadas linguagens mnemônicas mais intuitivas e amigáveis para o programador, mais próximas de uma linguagem natural humana, a fim de permitirem operações, gradualmente, mais complexas e cada vez menos especializadas, do que aquelas que eram possíveis quando se faziam necessários algoritmos totalmente desenvolvidos em linguagem de máquinas: seja fazendo uso de transistores, peças de tear ou demais entidades concretas, somente permitiam a execução de funções pré-delimitadas. Linguagens com nível de abstração ainda maior e, por isso, mais próximas da linguagem do programador, datam da década de 50. Representantes do que se convenciona chamar de terceira geração de linguagens de programação são Cobol, Fortran e a revolucionária, e amplamente utilizada até hoje, Linguagem C. A partir do nível de abstração permitido por tais linguagens, a quantidade de funções passíveis de serem desenvolvidas em um computador de uso geral aumentou consideravelmente em comparação às gerações de linguagens de programação anteriores.

Há, portanto, um movimento histórico que poderíamos chamar também de transdutivo, para usar um termo simondoniano, no desenvolvimento das linguagens de programação de *software*. Ao passo que as primeiras linguagens exigiam que o programador entendesse a fundo o mecanismo de *hardware*, a partir da qual a sua programação seria executada, as linguagens mais modernas permitem que o programador utilize uma linguagem de programação próxima de uma linguagem de comando coloquial. Esse foi um movimento que se deu gradualmente no desenvolvimento das linguagens de programação de uma forma que muito se assemelha a um processo evolutivo.

Conforme evoluíam, cada nova geração de linguagem foi se tornando cada vez mais acessível ao programador e, paralelamente, mais generalista, permitindo elaborar um maior número de operações lógicas, abstrações matemáticas, construções de simulações e virtualizações. Quando analisamos sua aplicabilidade e relação com o *hardware*, concluímos que, enquanto os primeiros *softwares* só podiam ser executados em máquinas idênticas àquelas na qual a programação foi concebida, com a abstração cada vez maior das linguagens e o desenvolvimento modular de equipamentos, se tornou possível que um *software* fosse executado em *hardwares* bastante diversos, desde que suas entidades modulares tenham sido definidas conforme modelos pré-estabelecidos e a capacidade de processamento, memória e armazenamento o permitam.

DUALISMO *HARDWARE-SOFTWARE*

Um problema clássico das Ciências da Computação é a relação entre *software* e *hardware*, que propomos a partir de uma abordagem filosófica clássica, chamar de dualismo *hardware-software*. Em uma tentativa teórica de cunho epistemológico, pode-se tentar estabelecer, separadamente, os conceitos de *software* e *hardware*, refletindo na análise das entidades técnicas a mesma concepção ontológica

hilemórfica que, ao menos desde Aristóteles, impõe à realidade percebida uma separação entre matéria e forma, entre substância física e suas categorias ou funções.

Pode-se entender que todo o desenvolvimento da tecnologia de informação se fez imerso em um mundo cuja posição epistêmica hegemônica é a posição dualista, o que, consequentemente, nos faz entender *software* e *hardware* como entidades correlacionadas, porém ontologicamente separadas. É até mesmo possível, e isso é o que experimentamos ao nos relacionarmos com essas entidades em nossa lida diária, percebendo a existência de um determinado *software* como independente do *hardware* no qual é executado.

Dizemos que *o software roda* ou é executado em uma multiplicidade de *hardwares* distintos, apresentando o mesmo comportamento e dando acesso às mesmas funções para as quais foi programado, independentemente do aparato físico. Ou seja, pensamos que um determinado programa de computador é o mesmo independente do equipamento no qual está sendo executado. No entanto, analisando mais minuciosamente, sabemos que o que é executado em um dado *hardware* é uma cópia individual devidamente compilada e traduzida em linguagem de máquina (binária), a qual é impressa em uma entidade também individual de *hardware*, e que o comportamento observado quando em pleno funcionamento, supondo que não houve erro durante os procedimentos de compilação e instalação, só é possível através da operação sinérgica e inseparável entre *software* e *hardware*.

Colocando em outros termos, a cópia do *software* que permite sua execução plena só existe quando instalada em um *hardware* em particular. Antes disso, temos apenas uma mera instrução na forma de um programa algorítmico que, além das funções específicas, também instrui a máquina a respeito de como a sua instalação será realizada. Percebe-se, assim, que o conjunto *hardware-software* em execução é único. O ser técnico computacional só existe de maneira integral em sua individualidade como relação sinérgica e inseparável entre *hardware-software*. Dizemos frequentemente em Ciência da Computação que o *software* foi integrado

ao *hardware*, o que permite essa visão que aqui propomos de que se torne um e indivisível. O que nos dá a ilusão de que um mesmo *software* pode ser executado em *hardwares* diversos é a mesma ilusão que temos quando analisamos matéria e forma a partir do esquema hilemórfico aristotélico. Neste sentido é que entendemos o *software* base como uma abstração e o *software* instalado como concreto.

Segundo Simondon, "a essência da concretização do objeto técnico é a organização de subconjuntos funcionais no funcionamento total" (Simondon, 2020, p. 75). Como entender, no entanto, essa concretização a partir de seus subconjuntos, em específico nos casos das entidades de *software*, é um problema que se nos impõe. Em um *hardware* genérico, como estes que usamos em nossas atividades diárias, as funções de suas partes não vêm necessariamente pré-estabelecidas. Os primeiros *hardwares* eram gigantescas máquinas de calcular, sendo o ENIAC, primeiro computador eletrônico programável de propósito geral que pesava cerca de 30 toneladas e ocupava uma sala inteira de 10 x 15 metros, praticamente uma entidade mitológica na cosmologia das máquinas.

Apesar do consenso estabelecido de que o ENIAC é o primeiro *computador* de uso geral, como o próprio termo que o designa, é sabido e difundido que sua principal função era a realização de cálculos que uma mente humana podia realizar e aferir de maneira relativamente fácil e rápida. Se tratava, portanto, de um conjunto de operações de cálculo muito mais simplificados do que todo o potencial computacional das máquinas atuais, com seus mecanismos de inteligência artificial, realidade virtual e simulações impossíveis de serem realizadas mesmo para um grande conjunto de mentes humanas unidas em um tempo finito. Essas máquinas hodiernas, além de realizarem uma gama de simulações, emulações e cálculos quase inimagináveis para a mente humana, possuem um potencial de agregação funcional que está longe de se esgotar. Há muito que não se trata mais de meras máquinas de calcular. Sem uma especialidade pré-definida, tais dispositivos são realmente de propósito geral e o potencial de aplicações às quais se destina é, pelo menos em tese e até onde se entende, infinito.

PROCESSO EVOLUTIVO DAS ENTIDADES DE *SOFTWARE*: DO ABSTRATO AO CONCRETO

Se, como vimos, as primeiras máquinas programáveis tinham funções específicas pré-definidas, o Computador Pessoal (do inglês *Personal Computer* - PC) é um artefato técnico que se adapta quase que ilimitadamente a uma multiplicidade de funções tão diversas quanto se possa pensar, tais como a comunicação por texto, a previsão meteorológica, os jogos lúdicos e a localização geográfica. Tendo se popularizado no final do século XX, a ponto se de se tornar neste início de século XXI praticamente onipresente em seus vários formatos, dentre eles, o de dispositivos portáteis que cabem na palma da mão de uma criança, um computador moderno é formado a partir de unidades sinérgicas, que, uma vez programadas, têm a capacidade de realizar uma infinidade de funções. A cada dia se desenvolvem novas aplicações (do inglês *application* ou, simplesmente, *app*) almejando atender um conjunto cada vez mais diverso de necessidades humanas, as quais, como nos diz Simondon, embora não tratando especificamente de dispositivos eletrônicos, mas das entidades técnicas de uma forma ampla, "se diversificam ao infinito", enquanto "as direções de convergência das espécies técnicas são em número finito" (Simondon, 2020, p. 60).

Embora lidando com exemplos da eletrônica de sua época, portanto, muito mais simplificada e limitada do que a que constitui um computador do século XXI, ou buscando exemplos em lâmpadas e motores, Gilbert Simondon parece já ter previsto e analisado minuciosamente no *MEOT* certa evolução possível dos objetos técnicos que muito nos interessa para a análise do modo de existência das entidades de *software*.

Tratando de um movimento de desespecialização das máquinas, em um sentido que vai do analítico abstrato ao mais concreto, o autor nos diz que "a

especialidade não se faz função por função, mas sinergia por sinergia; o grupo sinérgico de funções, e não a função única, constitui o verdadeiro subconjunto no objeto técnico." (Simondon, 2020, p. 60). Disto concluímos, com Simondon, que as entidades técnicas estão sempre defasadas ou, de certa maneira, limitadas em relação à multiplicidade de demandas dos seres humanos.

No entanto, ao considerarmos uma máquina multifuncional como o computador, é possível, ao menos, conjecturar que, para cada necessidade técnica vislumbrada pelo homem, haja uma solução possível. Ou seja, seria, ao menos em tese, por intermédio do computador que se poderia atingir o que Hermínio Martins chama de projeto prometeico, o qual estabelecia já "no marxismo clássico e no leninismo, um horizonte tecnológico completamente aberto de crescimento ilimitado das forças produtivas, supostamente capaz de ultrapassar todo e qualquer obstáculo natural e consequente escassez" (Martins, 2012, p. 46). Logo, se a solução para um determinado problema advindo de uma necessidade humana ainda não está plenamente equacionada e implementada em um conjunto *hardware-software*, seria, nesta hipótese, apenas uma questão de tempo para que tal desenvolvimento se concretizasse.

Na hipótese aqui formulada, portanto, entendemos que o computador pessoal de uso geral seria essa máquina concreta por excelência. Em uma linha gradual, que vai do objeto abstrato artesanal de função específica e delimitada até o mais concreto objeto técnico polivalente, cuja limitação é apenas uma questão de escala, pode-se pensar que "o objeto técnico existe, pois, como um tipo específico, obtido no final de uma série convergente. Essa série vai do modo abstrato ao modo concreto e tende para um estado no qual o ser técnico seria um sistema inteiramente coerente consigo mesmo, inteiramente unificado." (Simondon, 2020, p. 60). Parece-nos razoável que esse sistema inteiramente coerente consigo mesmo, no limite da série técnica convergente, seja o computador multifuncional.

Um autor contemporâneo de Simondon nos permite vislumbrar o quanto do potencial dessa máquina concreta poderia levar a um desenvolvimento das

capacidades humanas a um desenvolvimento para além do homem. Falando sobre os esforços de Galileu Galilei para comprovar as teses heliocêntricas de Copérnico, Paul Feyerabend, em sua obra *Contra o Método*, detalha como a invenção do telescópio permitiu a comprovação da teoria copernicana:

> Ele [Galileu] admite que 'não fosse pela existência de um sentido superior e mais eficaz que o senso natural e comum a juntar forças com a razão', teria estado 'muito mais recalcitrante com relação ao sistema copernicano' [...]. O 'sentido superior e mais eficaz', é claro, é o telescópio. (FEYERABEND, 2011, p. 118)

Neste momento da história, o que estava em jogo para a ciência astronômica era o posicionamento da Terra no cosmos. Com os olhos nus, juntamente aos demais sentidos que nos prendem ao chão, segundo Feyerabend, era àquela época um esforço de fé acreditar que a Terra e outros planetas giram ao redor do Sol. Foi a invenção de um *sentido superior e mais eficaz* que tornou possível experimentos que comprovaram a teoria heliocêntrica, trazendo notoriedade histórica e irrefutável a Copérnico e Galileu.

Semelhantemente se busca, com o uso de recursos computacionais, a ampliação não apenas das formas de ver a realidade como no caso do telescópio, mas de como desenvolver todos os sentidos humanos. Rumo a um transumanismo, o computador e suas potencialidades permitem ao homem almejar a obtenção de sentidos *superiores e mais eficazes*, nas palavras de Feyerabend, supostamente em níveis ainda inimagináveis.

MUDANÇA DE USO E LINGUAGEM EM UMA PERSPECTIVA COMPUTACIONAL

A respeito das finalidades de um objeto técnico, o professor, sociólogo e filósofo Hermínio Martins diz que "os objetos técnicos servem muito

regularmente a finalidades inesperadas, demonstram propriedades relevantes ou funcionalidades que não foram consideradas como pertinentes no seu desenho" (Martins, 2012, p. 99). Ou seja, mesmo em dispositivos com finalidades delimitadas contingencialmente, o filósofo da tecnociência identifica uma ampliação funcional latente. Se tal teoria já tem um impacto grande na análise de objetos técnicos de função determinada, quanto mais para um computador de uso geral. Outrora concebido para realizar cálculos, o computador foi evoluindo, reduzindo suas dimensões a partir do desenvolvimento da nanotecnologia, aumentando sua gama de aplicabilidades, culminando, recentemente, com a potencialidade praticamente ilimitada de criar mundos tão complexos que nos fazem mesmo questionar, a partir de uma concepção solipsista, se tudo o que experimentamos não seria fruto de uma simulação computacional. Alguns "tecnólatras" mais entusiasmados chegam a dizer que há uma chance de cinquenta por cento de estarmos vivendo em uma simulação da espécie daquela mostrada no filme *Matrix*, dirigido pelas irmãs Lana e Laura Wachowski (Ananthaswamy, 2020, n.p.). Ou seja, a própria forma como enxergamos o mundo no século atual está irreversivelmente contaminada por essa ontologia maquínica.

Falando ainda sobre a "'lei de mudança de uso', formulada em 1880 pelo engenheiro alemão E. von Harting" (Martins, 2012, p. 99), Martins traz um paralelo entre o desenvolvimento tecnológico e a linguagem que muito nos interessa no contexto desse ensaio. Referindo-se ao trabalho de Richard von Mises que "chega até a comparar repetidamente este processo da história da tecnologia com a própria evolução das línguas" (Martins, 2012, p. 99), o autor português correlaciona tão fortemente a tecnologia e a linguagem que nos faz pensar ainda mais no quanto o uso das linguagens de programação tem de poderoso no desenvolvimento desses equipamentos e, até mesmo, da forma como vemos o mundo. À luz dessa concepção trazida por Martins, concluímos que o desenvolvimento de *software* e *hardware* se deu, e necessariamente só poderia ser assim, simultânea e sinergicamente. Se, por um lado, as linguagens de programação

foram ficando cada vez mais acessíveis e generalistas, por outro lado, os *hardwares* foram se tornando cada vez mais compactos e generalistas.

Mas, será que podemos compreender esse movimento evolutivo de maneira totalmente determinística e causal? Quando Richard Feynman, físico teórico ganhador do prêmio Nobel em 1965, disse a famosa frase "*O que eu não posso criar, eu não entendo*", deu a entender que aquilo que nós criamos, nos é plenamente compreensível. Essa visão repete a lógica antropocêntrica que atribui ao homem o estatuto de dominação da natureza, em especial à natureza por ele manipulada mediante a técnica. Para Gilbert Simondon, no entanto, as causas extrínsecas que levam ao desenvolvimento dos objetos técnicos não são mais fortes do que causas intrínsecas que não dominamos e, na maioria das vezes, sequer estamos interessados em investigar. Para o autor francês:

> [...] os objetos técnicos evoluem para um pequeno número de tipos específicos em virtude de uma necessidade interna, não como resultado de influências econômicas ou de exigências práticas; não é o trabalho em linha de montagem que produz a padronização, a padronização intrínseca é que permite a existência do trabalho em linha de montagem (Simondon, 2020, p. 61)

Partindo dessa concepção ontológica que atribui à técnica um movimento autônomo e intrínseco, entendemos que há um devir próprio da técnica ou, nas palavras do autor "o objeto técnico uno é unidade de devir" (Simondon, 2020, p. 56). Logo, não há uma relação epistêmica de domínio total das tecnologias desenvolvidas. Não só as operações realizadas por essas máquinas cada vez mais inteligentes se tornam incompreensíveis para o ser humano, até mesmo os seus mecanismos de operação e, a cada instante, em sua relação dualista *hardware-software*, está cada dia mais distante de ser abarcada pela mente humana. Hermínio Martins, se referindo ao cientista da computação e professor no MIT Joseph Weizenbaum, nos lembra o seguinte:

> [...] relativamente aos computadores mais avançados, um eminente estudioso das ciências da computação vem afirmando que eles são estritamente incompreensíveis ou pelo menos fora da nossa compreensão plena segundo os critérios mais convencionais nas ciências exatas. (Martins, 2012, p. 102).

É como se estivéssemos envolvidos no desenvolvimento de uma nova geração de máquinas inteligentes cujos detalhes de operação sejam tão ou mais incompreensíveis para nós, quanto a própria mente humana ainda o é (e, talvez, sempre o será).

Há uma notícia com caráter anedótico, quase mitológico, que ilustra bem essa problemática: em uma pesquisa recente, cientistas da computação e engenheiros do Facebook programaram dois computadores com todo um léxico de palavras em inglês, ensinaram-nos a se comunicar utilizando dos mais avançados algoritmos de inteligência artificial, a fim de observar os desenvolvimentos de negociação em linguagem humana por eles realizados. Por um pequeno deslize (ou desejo subconsciente, quem sabe), os engenheiros de computação responsáveis pelo programa esqueceram de programar regras que limitassem a comunicação entre os *bots* ao inglês gramaticalmente correto. O que aconteceu a partir daí foi que, após um pequeno tempo de comunicação entre os dois *bots*, eles começaram a desenvolver, fazendo uso livre da língua inglesa, uma forma de comunicação aparentemente mais eficiente para negociação. Por exemplo, a frase "*Balls have zero to me to me to me to me to me to me to me to*", usada por um dos *bots* envolvidos na comunicação, aparentemente substitui o uso do número oito por uma sequência de termos repetidos oito vezes.

O que pode parecer, para nós, como uma repetição cansativa de termos, foi uma solução desenvolvida por esses agentes computacionais mais eficiente do que ter que lidar com toda uma nova classe de comunicação, a qual chamamos números. Com certo receio pelo desconhecimento das consequências e assumindo

o erro de programação como não intencional, o Facebook optou por desligar as máquinas e repensar o programa. O estudo conclui amargamente: "podemos ensinar bots a falar, mas nunca aprenderemos sua linguagem". (Wilson, 2017, n.p.).

Este é um mero exemplo ilustrativo que mostra o potencial das tecnologias da ciência da computação e o quanto desconhecemos suas implicações, suas regras evolutivas e seu potencial de comunicação intrínseco. É bastante plausível que nos acostumemos, em breve, a estórias como essas de como máquinas inteligentes se comunicam entre si e conosco em uma sequência da ampliação de suas funções.

ONTOLOGIA DAS ENTIDADES DE SOFTWARE

No início desta investigação, usamos como ponto de partida um texto de Rudolf Carnap de 1956 que, apesar de nos inspirar pelo uso que faz do termo *Framework*, não nos trouxe material adicional para a discussão que temos desenvolvido. É na obra de um discípulo de Alfred North Whitehead e frequentador honorário do Círculo de Viena, portanto parceiro eventual de diálogos com Carnap, que encontraremos as bases para aprofundar um pouco mais essa análise ontológica das entidades abstratas de *software*. Willard van Orman Quine escreveu, em 1953, portanto 3 anos antes do texto de Carnap, o artigo *Sobre o que há*, no qual o autor faz uma análise ontológica que vai desde "o velho enigma platônico do não-ser" (Quine, 1980, p. 217), passando pela querela dos universais da filosofia medieval até as problemáticas levantadas por correntes matemáticas contemporâneas, como o logicismo, o intuicionismo e o formalismo.

Pretendemos elaborar um pouco sobre a "questão do saber se há entidades tais como atributos, relações, classes, números, funções" (Quine, 1980, p. 222) nos perguntando e propondo algumas possibilidades de resposta sobre o estatuto ontológico de *classes* que definem objetos de *software*, *funções* que implementam rotinas de execução, *atributos* que são inseridos nessas funções e outras formas de

variáveis utilizadas em programação orientada a objetos. Nos utilizaremos do léxico semântico da linguagem C++ como representativa das entidades de *software* e buscaremos, em Quine, argumentos que nos falem sobre a essência dessas entidades.

A começar pela "fórmula semântica 'ser é ser o valor de uma variável'" (Quine, 1980, p. 226), Quine abre um primeiro horizonte de compreensão segundo o qual essas variáveis de *software*, ao receberem valores advindos dos objetos descritos pela linguagem, nos apresentam seu verdadeiro ser. Na linguagem C++, uma classe é definida abstratamente descrevendo os atributos que irão compor um objeto dela derivado. Por exemplo, podemos definir uma classe *Pen* que tenha atributos tais como cor, comprimento, raio da sessão transversal, peso, material etc. Quando se instancia um objeto da classe "Pen", tem-se um objeto *oPen1* que terá descrito valores específicos para cada atributo (Ex: oPen1.cor = "blue"; oPen1.peso=0.1 etc.). Ontologicamente falando, uma entidade de *software* vem a existir quando instanciada na execução do programa segundo um modelo previamente programado ou dinamicamente decidido a partir de sua lógica interna. Desta forma, podemos concluir que os entes de toda e qualquer estrutura computacional possuem seu Ser em conjunto com a matéria que forma a máquina, e nunca desassociado dela.

Este é o motivo pelo qual, já de início, refutamos a noção hilemórfica, que, por comparação, tenderia a tratar separadamente o *software* do *hardware*. Apesar de descrito de uma maneira centralizada na forma de classes abstratas e lógica algorítmica pré-programada, em algo que se poderia correlacionar com a noção leiga de "mundo platônico das Ideias", uma entidade de *software* só existe concretamente de maneira indissociável ao *hardware*, conforme tratamos previamente. Cada instância de *software*, mesmo que vindo de uma cópia compilada do programa base, é único em e para cada *hardware* no qual for instalado.

Partindo da noção de Fenomenotécnica em Gaston Bachelard, entende-se que uma teoria não apenas descreve, mas constrói o objeto, talvez de forma

semelhante ao mecanismo com o qual um *software* dita as regras que irão construir um objeto instanciado como entidade computacional. Citando o linguista e engenheiro Benjamin Lee Whorf, Feyerabend expressa claramente sua simpatia e concordância pela ideia "clara e elegantemente formulada" pelo autor americano "de que as línguas e os padrões de reação que envolvem não são meros instrumentos para *descrever* eventos (fatos, estados de coisa), que sua 'gramática' encerra uma cosmologia, uma visão abrangente do mundo, da sociedade e da situação do ser humano" (Whorf, 1956 *apud* Feyerabend, 2011, p. 215). Isto posto, entendemos que, da mesma forma que a nossa linguagem é capaz de uma cosmologia formativa e abrangente do mundo que habitamos, a linguagem computacional é, em primeira instância, um instrumento cosmológico básico de toda uma ontologia intrínseca nos computadores, e ulteriormente também do mundo em que habitam juntos, na coexistência entre máquinas e seres *naturais*.

Parece-nos, portanto, que as entidades de *software* atuam como agentes formadores de mundo e, por consequência, transformadores da ciência (em específico, a *Technoscience* dos anos 2000). É inimaginável um desenvolvimento científico relevante na contemporaneidade sem um uso intensivo de mecanismos computacionais para coleta e armazenamento de dados, cálculos estatísticos complexos e simulações intrincadas.

O COMPUTADOR COMO INSTRUMENTO SUPRA-PARADIGMÁTICO

Na influente obra *A Estrutura das Revoluções Científicas*, Thomas Kuhn nos diz que "a decisão de empregar um determinado aparelho e usá-lo de um modo específico baseia-se no pressuposto de que somente certos tipos de circunstâncias ocorrerão" (Kuhn, 2018, p. 136) e também que "sem os instrumentos especiais, construídos sobretudo para fins previamente estabelecidos, os resultados que conduzem às novidades poderiam não ocorrer" (Kuhn, 2018, p. 142-143). Kuhn

está pleno de razão, desde que consideremos o desenvolvimento de instrumentos técnicos dentro de uma ciência normal, que se utiliza do próprio paradigma vigente para aferir, testar e analisar resultados científicos em meio a este desenvolvimento normal (portanto, não revolucionário).

Neste cenário apresentado por Kuhn há cerca de sessenta anos à luz de uma pesquisa histórica realizada por ele até então, os desenvolvimentos disruptivos e revolucionários acontecem a partir de uma crise motivada em grande parte pelo uso de instrumentos desenvolvidos durante o desenvolvimento normal dentro do paradigma. Para o autor, "a novidade não antecipada, isto é, a nova descoberta, somente pode emergir na medida em que as antecipações sobre a natureza e os instrumentos do cientista demonstrarem estar equivocados" (Kuhn, 2018, p. 183). No entanto, como lidar com a pretensão de equipamentos adaptativos que se ajustem autonomamente a novas descobertas? Utilizando-se de técnicas de *machine-learning* é plenamente possível desenvolver sistemas adaptativos que respondam a mudanças paradigmáticas na ordem das que Kuhn discutiu. Quando se tem um instrumento como o computador de propósito geral, seja através de troca de *software* ou de mecanismos adaptativos inteligentes do próprio *software*, pode-se mudar completamente a forma de lidar com os dados e, consequentemente, transitar de um paradigma científico a outro com o clique de um botão.

Kuhn chega a vislumbrar em certo ponto da sua obra, o desenvolvimento de "uma linguagem de observação pura" que permitiria transitar de um paradigma a outro. O autor parece apostar que "talvez ainda se chegue a elaborar uma" linguagem de observação pura, visto que "nenhuma das tentativas atuais conseguiu até agora aproximar-se de uma linguagem de objetos de percepção puros" (Kuhn, 2018, p. 220). A grande dificuldade apontada por Kuhn é que essas linguagens pressupõem o paradigma existente. No entanto, um computador de propósito geral, não especializado *a priori* em nenhuma teoria científica pré-estabelecida, implementando algoritmos computacionais inteligentes e adaptativos, poderia

muito bem transitar entre dois paradigmas distintos, inclusive, testando pressuposições de ambos a fim de encontrar, senão o *verdadeiro*, aquele que melhor se adequa à natureza ou a um conjunto de interesse. Propomos, então, que tal aparato técnico, um computador de alta capacidade de processamento de dados, dotado de mecanismos de aprendizado avançados (*machine learning*), pode atuar como uma ferramenta supra-paradigmática, potencialmente resolvendo o problema de incomensurabilidade entre os paradigmas.

Nos parece enigmático, senão um lapso imperdoável de Kuhn, ter ignorado completamente os potenciais do computador em sua obra seminal. Tendo antecipado tantos dilemas da ciência da segunda metade do século XX, chega a ser surpreendente que Kuhn nunca chegue a citar os dispositivos de computação em sua inovadora obra. No entanto, Kuhn estava plenamente desperto em relação a essa necessidade e chegou a esboçar um caminho. Remetendo-se às teorias de verificabilidade do Empirismo Lógico de Viena e questionando o falseacionismo de Karl Popper, Kuhn nos diz:

> [...] uma teoria probabilística requer que comparemos a teoria científica em exame com todas as outras teorias imagináveis que se adaptem ao mesmo conjunto de dados observados. Uma outra exige a construção imaginária de todos os testes que possam ser concebidos para testar determinada teoria. Aparentemente, tal construção é necessária para a *computação* de probabilidades específicas, absolutas ou relativas, mas é difícil perceber como possa ser obtida (KUHN, 2018, p. 243)

Apesar, no entanto, do uso explícito da palavra *computação* (grifo nosso na citação acima) e de este texto de Kuhn datar de 1962, portanto dezesseis anos após o desenvolvimento do ENIAC, o autor parece ignorar completamente o potencial dos dispositivos computacionais na testabilidade de teorias. Talvez pelo caráter ainda limitado destes dispositivos no início da década de sessenta, ou pelo próprio desconhecimento do autor, que tem sua formação mais fortemente marcada na área da física teórica e até onde se sabe, nenhuma experiência com os aparatos

computacionais da sua época, o fato é que Kuhn parecia já apostar em uma linguagem supra-paradigmática, mas não percebeu que a gênese do aparato técnico que a permitiria já estava em andamento.

CONCLUSÃO

Como já esboçado na introdução, o objetivo deste ensaio é trazer uma reflexão sobre o modo de existência das entidades de software, sendo as linguagens de programação um aspecto constitutivo primordial desse próprio modo de ser. Uma vez que cada ser humano passa cada vez mais tempo em contato com aplicativos em seus computadores pessoais, celulares e demais máquinas que povoam seu entorno, entendemos como de fundamental importância social, o entendimento destes seres com os quais nos relacionamos, sua constituição e estatuto ontológico e até mesmo as autônomas relações entre si.

Optamos por dividir este ensaio em pequenos capítulos por entendermos que cada um deles tem um potencial de desenvolvimento de pesquisa próprio e passível de ser aprofundado futuramente, sejam em trabalhos isolados a respeito de um desses capítulos temáticos, seja em conjunto na forma de um trabalho que dê mais sustentação às teses aqui lançadas. Isto posto, e estando ciente de que o trabalho em questão não esgota os assuntos abordados, e nem finaliza totalmente a pesquisa que permitiria concluirmos mais categoricamente a respeito do modo de existência das entidades de softwares, propomos alguns desenvolvimentos de pesquisa futuros.

Começamos por traçar uma breve genealogia histórica das linguagens de programação, destacando alguns exemplos que consideramos emblemáticos e representativos o suficiente, desde o tear de Jacquard, "a primeira máquina programável" até os mais avançados robôs com inteligência artificial no século XXI, para propormos o que, inspirados por Simondon, chamamos de gênese transdutiva. A ideia de que essa evolução se deu do abstrato para o concreto, de uma linguagem mais própria da máquina para linguagens mais atuais de alto nível

que se assemelham muito à linguagem humana. Este assunto poderia ser aprofundado a em uma análise filosófica de cada etapa à luz de autores como Quine, Kuhn e Gilbert Simondon, citados já neste ensaio, mas acusamos a ausência de Ludwig Wittgenstein, cujo profundo trabalho que envolve Lógica e Linguagem não pôde ser abordado no presente trabalho, mas poderia enriquecer profundamente esta pesquisa.

Neste esboço cronológico evolutivo falamos também do que consideramos um reflexo da concepção dualista na forma que entendemos, desenvolvemos e nos relacionamos com equipamentos (hardware) e seus programas (software). Na nossa perspectiva, o problema mente-corpo, clássico na filosofia da vida e da ciência, se traduz para as máquinas computacionais, na forma de um dualismo *hardware-software*. Da mesma forma que entendemos, junto com alguns autores dos quais novamente Simondon se faz notório, que não há uma separação ontológica distintiva entre mente e corpo, defendemos também que a separação em *software* e *hardware* é ilusória e apenas útil quando se trata de estabelecer nosso modo de atuação sobre esses entes. Se estamos interessados em entender o modo de ser das entidades de software, como o próprio título deste ensaio suscita, devemos partir da constituição dos *hardwares*, partindo de sua estrutura modular e entendendo a partir de avanços tecnológicos, qual o impacto existencial para o *software*. Seria possível aprofundar essa discussão fazendo uma espécie de Ontologia aprofundada das máquinas tendo a obra de Gilbert Simondon como base mas não deixando de investigar outros autores de uma tradição Metafísica para trazer conceitos fundamentais da Filosofia para a investigação dos modos de existência dos objetos técnicos de *hardware-software*.

Exploramos também conceitos como a "lei da mudança de uso" na linguagem e aspectos do desenvolvimento computacional a fim de aprofundarmos um pouco mais nossa compreensão sobre como o desenvolvimento das linguagens computacionais podem impactar, e como vimos em ao menos um exemplo ilustrativo já estão impactando, a nossa forma de relação com as máquinas e uns

com os outros. Por fim, propomos a tese de que o computador seria uma espécie de instrumento supra-paradigmático que nos permitiria, e a nosso ver a tecno-ciência já é um resultado disso, transitar por paradigmas científicos distintos com um mero clique.

Conforme deixamos claro desde o início, as discussões propostas neste trabalho têm muito mais o objetivo de abrir possibilidades através de uma análise hipotética na forma de um ensaio do que necessariamente fazer um estudo rigoroso e conclusivo. Esperamos que tal objetivo tenha sido atingido e tenha instigado o leitor a enveredar pelos caminhos que tentamos indicar a fim de que sejam aprofundadas investigações que consideramos de suma importância para a Sociedade da Informação do século XXI.

* * *

Referências

ANANTHASWAMY, Anil. "Do we live in a simulation? Chances are about 50-50". *Scientific American*, 2020. Disponível em: https://www.scientificamerican.com/article/do-we-live-in-a-simulation-chances-are-about-50-50/. Acesso em: mar. 2021.

CARNAP, Rudolf. *Empirismo, Semântica e Ontologia*. Coleção Os Pensadores. Trad. Pablo Rubén Mariconda. Brasil: Editora Nova Cultural, 1988. p. 113-128.

FEYERABEND, Paul. *Contra o Método*. Trad. Cezar Augusto Mortari. Brasil: Editora Unesp, 2011.

HEIDEGGER, Martin. *Ser e Tempo*. Brasil: Editora Vozes, 2015

JOHNSON, R.E.; FOOTE, B. *Designing Reusable Classes*, Tradução do autor. United States: Journal of Object-Oriented Programming (Jun/Jul). 1988; 1:22-35.

MAAYAN, Gilan. "The IoT Rundown For 2020: Stats, Risks, and Solutions". *Security Today*, 2020. Disponível em: https://securitytoday.com/Articles/2020/01/13/The-IoT-Rundown-for-2020.aspx. Acesso em: mar. 2021.

MARTINS, Hermínio. *Experimentum humanum : civilização tecnológica e condição humana*. Brasil: Editora Fino Traço, 2012.

QUINE, Willard van Orman. *Sobre o que há*. Coleção Os Pensadores. Trad. Luis Henrique dos Santos. Brasil: Editora Abril Cultural, 1980. p. 217-230.

SIMONDON, Gilbert. *Do Modo de Existência dos Objetos Técnicos*. Trad. Vera Ribeiro. Brasil: Editora Contraponto, 2020.

WILSON, Mark, 2017. "AI is inventing languages humans can't understand. Should we stop it? ". *Fast Company*, 2017. Disponível em: https://www.fastcompany.com/90132632/ai-is-inventing-its-own-perfect-languages-should-we-let-it. Acesso em: mar. 2021.

Recebido 14/06/2021

Aprovado 28/03/2022