UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Elétrica e de Computação

Wandemberg Santana Pharaoh Gibaut

DCT: Uma Ferramenta para a Construção de Arquiteturas Cognitivas Distribuídas e o Desenvolvimento de *Cognitive Twins*

Campinas

2024

Wandemberg Santana Pharaoh Gibaut

DCT: Uma Ferramenta para a Construção de Arquiteturas Cognitivas Distribuídas e o Desenvolvimento de *Cognitive Twins*

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Doutor em Engenharia Elétrica, na Área de Engenharia de Computação.

Orientador: Prof. Dr. Ricardo Ribeiro Gudwin

Este trabalho corresponde à versão final da tese defendida pelo aluno Wandemberg Santana Pharaoh Gibaut, e orientada pelo Prof. Dr. Ricardo Ribeiro Gudwin.

Campinas

2024

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

G35d

Gibaut, Wandemberg Santana Pharaoh, 1992-
DCT: uma ferramenta para a construção de arquiteturas cognitivas distribuídas e o desenvolvimento de *cognitive twins* / Wandemberg Santana Pharaoh Gibaut. – Campinas, SP : [s.n.], 2024.

Orientador: Ricardo Ribeiro Gudwin.
Tese (doutorado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Arquiteturas cognitiva. 2. Sistemas cognitivos. 3. Inteligência artificial. 4. Sistemas distribuídos. 5. Gêmeos digitais. I. Gudwin, Ricardo Ribeiro, 1967-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações Complementares

Título em outro idioma: DCT: a tool for building distributed cognitive architectures and developing cognitive twins
Palavras-chave em inglês:
Cognitive architectures
Cognitive systems
Artificial intelligence
Distributed systems
Cognitive twins
Área de concentração: Engenharia de Computação
Titulação: Doutor em Engenharia Elétrica
Banca examinadora:
Ricardo Ribeiro Gudwin [Orientador]
José Reinaldo Silva
Alexandre da Silva Simões
Paula Dornhofer Paro Costa
Denis Gustavo Fantinato
Data de defesa: 24-01-2024
Programa de Pós-Graduação: Engenharia Elétrica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0001-7322-5399
- Currículo Lattes do autor: http://lattes.cnpq.br/6572854063431912

COMISSÃO JULGADORA - Tese de Doutorado

Candidato: Wandemberg Santana Pharaoh Gibaut **RA:** 106992
Data da defesa: 24 de janeiro de 2024

Titulo da Tese: DCT: Uma Ferramenta para a Construção de Arquiteturas Cognitivas Distribuídas e o Desenvolvimento de *Cognitive Twins*

Prof. Dr. Ricardo Ribeiro Gudwin (Presidente, FEEC/UNICAMP)
Prof. Dr. José Reinaldo Silva (Escola Politécnica - USP)
Prof. Dr. Alexandre da Silva Simões (UNESP - ICT - Sorocaba)
Profa. Dra. Paula Dornhofer Paro Costa (FEEC/UNICAMP)
Prof. Dr. Denis Gustavo Fantinato (FEEC/UNICAMP)

A ata de defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no SIGA (Sistema de Fluxo de Dissertação/Tese) e na Secretaria de Pós-Graduação da Faculdade de Engenharia Elétrica e de Computação.

Dedico esta tese principalmente a Maria Aparecida Santos Santana e, in memoriam, a Robson Pharaoh Gibaut. Dedico ainda a todos aqueles que vieram antes de mim, vivos ou mortos, sem os quais eu não estaria onde estou. Devemos seguir em frente sem esquecer o passado.

Agradecimentos

Em minha dissertação eu agradeci a um pequeno número de pessoas, aquelas que estiveram mais próximas de mim durante o período do mestrado. Dessa vez resolvi fazer um pouco diferente: agradecer a todos (ou quase todos) aqueles que fizeram parte de minha trajetória. Já peço desculpas de antemão a quem eu por ventura não mencionar, seja por esquecimento, seja por "espaço" Saibam que amo todos vocês.

Como não poderia ser diferente, queria começar agradecendo a minha mãe, Maria Aparecida, chamada carinhosamente por mim de Cidão por todo o sacrifício e preocupação nesses quase 32 anos. Por se colocar em situações (no mínimo) ruins para que eu pudesse ter uma boa educação e algum conforto, mesmo que pouco. Um apoio que não foi perfeito, mas que me fez em parte ser quem eu sou. Agradeço também ao meu pai (*in memoriam*) Robson por me servir de exemplo - positivo e negativo - do que é ser um homem adulto. Princípios como "não dependa de ninguém para cuidar de uma casa" "seja legal com as pessoas" e "defenda quem necessita" poderiam muito bem ter saído de algum quadrinho de heróis, mas veio de um homem que não tinha vergonha de chorar.

Em seguida gostaria de agradecer aos meus professores do ensino fundamental e médio. Agradeço à família Cezimbra (Isabel, Ismênia e Iolanda) a qual basicamente formou minhas habilidades iniciais em matemática. Esse Agradecimento também se estende a Graça Martins. Agradeço a Joelson Batista, meu primeiro professor de física, que me fez me encantar pelas aplicações da área e descobertas do mundo e foi um dos poucos professores negros (em Salvador, pasmem) que eu tive na escola elitista onde estudei. Se estiver lendo isso, saiba que aos 15 eu queria ser como você. Agradeço a Carla, professora de Biologia, pelo incentivo a buscar vestibulares "fora". Agradeço a Marli por ver um jovem cheio de si e não ter encarado como um desafio a sua autoridade de professora e sim como uma oportunidade de me ajudar a desenvolver um pensamento crítico que carrego para a vida. Agradeço a Wladimir por também por ter me incentivado, me orientado e ainda me presenteado com livros (digitais haha) que usaria na graduação. O último agradecimento do bloco é especial e vai para Sandinei Ugo, meu professor de laboratório de física do 2º ano. Mais do que apenas incentivar e me desafiar a ir mais longe, Sandinei me deu aulas gratuitas em preparação para o vestibular da Unicamp, me emprestou material, me pôs em contato com os programas de permanência estudantil e usou de favores para que eu fosse minimamente acolhido ao chegar em uma terra onde não conhecia ninguém. Sério, não estaria na Unicamp se não fosse ele.

Agradeço aos amigos de infância. E aqui a lista é reduzida. Reduzida àqueles que pude olhar nos olhos e ver verdadeira alegria quando tive minha maior conquista até 2012

(ser aprovado para fazer intercâmbio) em algo que não os envolvia e nenhum tinha feito até então, mesmo com dinheiro. São esses: Lula Bonfim, que conheço desde os meus 6 anos, Naira Mota e Gabriel Garcia.

Ao professor Romis um especial agradecimento por, como meu primeiro professor da engenharia da faculdade (EA772 - Circuitos Lógicos), estabelecer o que é ser um pesquisador, mostrar genuína alegria em seu trabalho e que o aprendizado não precisa vir necessariamente com sofrimento, com ensino (ou orientação) punitivista. Aos amigos da graduação, Rafael Sampaio, Thiago Cazarine (Capitão) e Rafael Calixto, saibam que vocês também tem lugar especial.

Agradeço muitíssimo a Sergio Bueno e Gustavo Menezes por conviverem comigo por anos na moradia e dividido um bom pedaço de minha vida, meus amores e minhas angústias. A Eugênio Rodrigues e a Jéssica Pereira por serem grandes amigos mesmo quando eu não fui o mesmo para eles. Agradeço a Rudolfo Batista pela amizade estabelecida em terras estrangeiras que até hora perdura. Se eu posso abrir a boca e falar que sei dançar forró, foi porque aprendi no FdC.

Agradeço a Flávio Franco, Élida Franco, Jordana Barbosa e Guilherme Renan que acompanharam o meu doutorado e ficam felizes por mim, mesmo que não entendam nada do que eu faço. A Helena Amélia e Isa Oliveira um agradecimento por me verem trocar tempo de qualidade por algoritmos rodando e estresse e ainda assim quererem meu bem. Um agradecimento especial a Triz Avelino - minha melhor amiga - por estar comigo de maneira muito próxima nesses momentos de solidão em Campinas. Agradeço ainda Isa Cassis, Joanna Rocha, Guilherme Barbosa e Leonardo Oliveira pelo amor e carinho mútuo que cultivamos.

Deixo em Salvador um "muito obrigado" a Ricardo Coutinho (Couts), Gab Caldas, Manu, Luma e Isadora pela amizade e amor nesses últimos anos.

Agradeço também a André Paraense pelos conselhos acadêmicos e de vida, sem os quais minha vida seria bastante mais difícil. Para mim você é um mentor. Aos colegas de trabalho Fábio Grassiotto, Alexandre Osório e Cláudio Santos, um agradecimento por aprender com vocês coisas novas que são/serão importantes daqui pra frente. Agradeço a Allan James, amigo e instrutor de Jiu-jitsu, por ter feito parte desse aspecto de minha vida (o jiu) que me remete a cura.

Agradeço ao professor Ricardo Gudwin, que foi meu orientador por longos (quase) 9 anos entre mestrado e doutorado, me proporcionou participação em projetos de alto nível — que eventualmente gerou patentes — e que sem o qual o caminho teria sido diferente.

Não poderia deixar de agradecer também a Leo Vilela — meu primeiro orientado — pela confiança em me permitir guiá-lo.

Por fim, agradeço aos Orixás e àqueles que antes de mim vieram e não estão mais entre nós.

Agradeço ainda à CAPES pelo apoio financeiro dado durante este período e ao Instituto Eldorado pelas horas liberadas para a dedicação à tese.

O presente trabalho foi parcialmente realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001, número do processo 88882.329471/2019-01.

"Ubuntu."
(palavra bantu)

Resumo

Este trabalho apresenta um framework para Arquiteturas Cognitivas Distribuídas e uma técnica para construir *Cognitive Twins* baseados em dados de interação de um agente externo usando treinamento supervisionado e uma Estratégia de Evolução em cima de tal framework. Aqui, mostramos que é possível orquestrar muitos dispositivos físicos e virtuais para obter boas aproximações do comportamento de interação de uma pessoa por treinar o sistema de ponta a ponta e apresentar métricas de desempenho. O *Cognitive Twin* gerado pode ser usado posteriormente para automatizar tarefas, gerar agentes artificiais semelhantes a humanos mais realistas ou ainda mais investigar seus comportamentos.

Palavras-chaves:Sistemas Cognitivos; *Cognitive Twin*; Agentes Inteligentes; Sistemas Distribuídos.

Abstract

This work presents a framework for Distributed Cognitive Architectures and a technique for building *Cognitive Twins* based on interaction data of an external agent using supervised training and an Evolution Strategy on top of such framework. Here, we show that it is possible to orchestrate many physical and virtual devices to obtain good approximations of a person's interaction behavior by training the system end-to-end and presenting performance metrics. The *Cognitive Twin* generated can be subsequently used to automate tasks, generate more realistic human-like artificial agents, or further investigate their behaviors.

Keywords: Cognitive Systems; Cognitive Twin; Intelligent Agents; Distributed Systems.

Lista de ilustrações

Lista de tabelas

Lista de Acrônimos e Abreviações

AC-DT *Associative Cognitive Digital Twin*

ACT-R *Adaptive Control of Thought—Rational*

AGI *Artificial General Intelligence*(Inteligência Artificial Geral)

C-DT *Cognitive Digital Twin*

CIoT *Cognitive Internet of Things*

CPS *Cyber-Physical Systems*

CST *Cognitive Systems Toolkit*

CT *Cognitive Twin*

CVO *Composite Virtual Object*

DCT *Distributed Cognitive Toolkit*

DT *Digital Twin*

ECT *Enhanced Cognitive Twin*

IA Inteligência Artificial

IoT *Internet of Things*

JVM *Java Virtual Machine*

LIDA *Learning Intelligent Distribution Agent*

MECA *Multipurpose Enhanced Cognitive Architecture*

RFID *Radio Frequency Identification*

RWO *Real-World Object*

SoS *Systems of Systems*

SoSE *Systems of Systems Engineering*

TIC *Tecnologia da informação e comunicação*

VO *Virtual Object*

Sumário

1 Introdução

Sistemas inteligentes — e de controle inteligente — vêm se tornando cada vez mais presentes na vida cotidiana das pessoas, sejam tais sistemas virtuais ou ciber-físicos como carros autônomos, casas inteligentes, *bots* de internet entre outros. Só em assistentes de voz (como Alexa, Siri, *Google Assistant* e Cortana) registrou-se um uso de cerca de 3.25 bilhões em 2019 com projeções de 8.4 bilhões em 2024 e estimativa de cerca de 298 bilhões em valor de mercado investidos em Inteligência Artificial (IA) (LARICCHIA, 2022; THORMUNDSSON, 2023; THORMUNDSSON, 2024). Esses sistemas são muitas vezes responsáveis por tarefas sensíveis a erros e/ou por casos onde é impossível (ou inviável) antever todas as situações a serem encontradas. Tendo em mente isso, faz-se necessário um controle inteligente adequado para os sistemas atuais e vindouros, com especial interesse na robustez destes sistemas e no quanto podemos confiar em sua integridade. Aspectos como explicabilidade, reprodutibilidade, transparência e alinhamento com valores humanos ajudam a mitigar problemas (LI *et al.*, 2023).

1.1 Motivações e Objetivos

Nesta seção, apresentamos as motivações que levaram ao desenvolvimento de nossa pesquisa, bem como justificativas específicas para os diversos temas estudados e sumarizados nessa tese, considerando aspectos técnicos, éticos, econômicos e conjunturais. Na subseção 1.1.1 chamamos a atenção para o AAAI-20, um importante congresso que contou com figuras relevantes do meio acadêmico e cujas discussões sintetizaram o incômodo presente de parte da academia em relação aos caminhos da IA. As subseções 1.1.2 e 1.1.3 explicam por que os temas principais que norteiam nossa pesquisa: Sistemas Cognitivos Distribuídos e *Cognitive Twins* foram identificados como temas relevantes aos problemas apresentados na subseção 1.1.1. Por fim, a subseção 1.1.4 elenca outros fatores relevantes ao desenvolvimento deste trabalho e que não deveriam — por compromisso moral — ficar de fora, como questões de "pegada de carbono" e a lógica do "mais é melhor".

1.1.1 O AAAI-20 e o Futuro da Inteligência Artificial

A 34ª conferência em Inteligência Artificial da *Association for the Advancement of Artificial Intelligence* realizado em 2020 (AAAI-20) representa um ponto de importante nos rumos dos campos de pesquisa — no plural — da área de Inteligência Artificial. Neste congresso, nomes de peso, como os laureados com o *Turing Awards*: Yann LeCun, Yoshua

Bengio e Geoffrey E. Hinton, discutiram sobre os possíveis rumos da IA, argumentando abertamente que, para a construção de sistemas de IA mais ricos e robustos, isto é, semanticamente consistentes, explicáveis — e portanto confiáveis — seria necessário o uso de componentes de *Reasoning* em adição aos tão populares métodos de *Deep Learning* (HINTON *et al.*, 2020). Isso representa basicamente um (ou o começo de um) reconhecimento, por parte da comunidade de pesquisadores, da importância do processamento simbólico, e que os caminhos da Inteligência Artificial Geral (AGI, do inglês *Artificial General Intelligence*) não serão trilhados simplesmente por meio do mero crescimento de tamanho em modelos conexionistas ou do aumento do volume de dados, ainda que se acredite que os modelos conexionistas façam parte dessa trilha.

Embora a ideia apresentada no congresso citado acima tenha sido do acoplamento progressivo de sistemas conexionistas com sistemas simbólicos, o autor dessa tese argumenta que princípios teóricos e práticos de Arquiteturas Cognitivas podem representar bons caminhos quanto a explicabilidade e confiança na IA, bem como uma forma de combinar os sistemas mencionados.

1.1.2 Por Quê Sistemas Cognitivos Distribuídos?

Conforme mencionado na seção anterior, algumas ideias oriundas da linha de pesquisa em Arquiteturas Cognitivas podem indicar bons caminhos na integração entre sistemas conexionistas e simbólicos, bem como contribuir para a explicabilidade e a confiança em sistemas de IA. Essa explicabilidade vem do fato de que, diferente de modelos puramente compostos por Redes Neurais, onde uma única rede é responsável pelo comportamento completo do sistema, podem existir especificidades no comportamento cognitivo de cada componente do sistema. Por exemplo, A Figura 1, extraída de Gudwin *et al.* (2017), mostra o MECA — acrônimo para *Multipurpose Enhanced Cognitive Architecure* — um modelo baseado em Codelets (este conceito será melhor abordado no capítulo 4) onde certos grupos de Codelets sintetizam funções cognitivas específicas. É possível investigar não apenas o sistema na totalidade, como uma caixa-preta, mas também focar em partes específicas do sistema, caso necessário. Um especialista, utilizando o MECA, poderia, em tese, avaliar o comportamento de cada Codelet. O conceito de explicabilidade será abordado mais adiante neste trabalho.

Tipicamente, Arquiteturas Cognitivas (KOTSERUBA; TSOTSOS, 2020) são implementadas em sistemas executados apenas de forma centralizada (em um dispositivo único), mesmo aquelas baseadas em teorias que discutem o paralelismo dos processos cognitivos, tais como Mitchell e Hofstadter (1994), Franklin *et al.* (2014) e Gudwin *et al.* (2017). De forma geral, em aplicações que não exijam agentes altamente complexos isto não seria um problema, mas, à medida que se buscasse escalar o agente (mais Codelets ou mais processamento), limitações relativas ao hardware único começariam a aparecer,

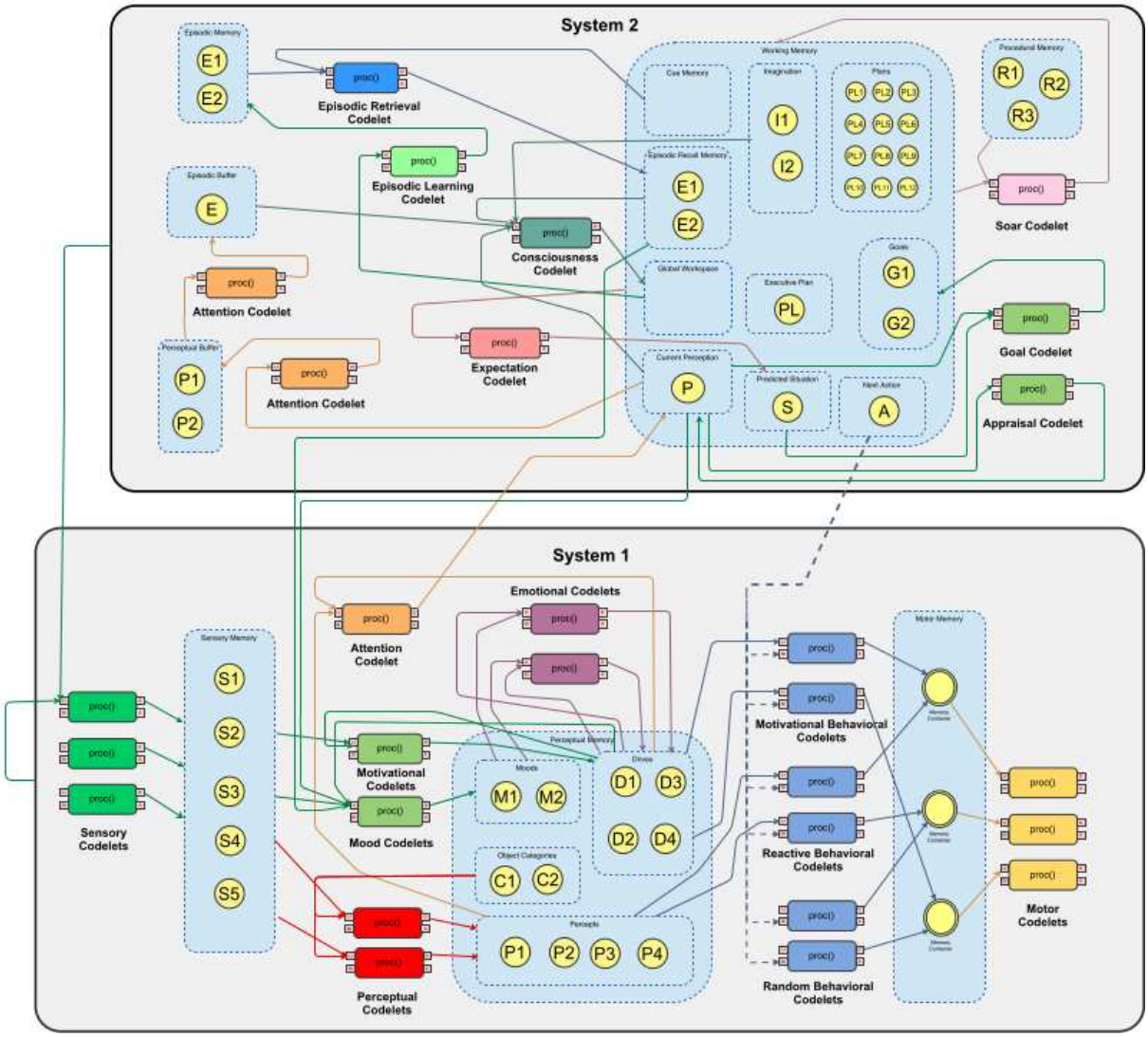


Figura 1 – A arquitetura MECA. Extraída de (GUDWIN *et al.*, 2017)

como agendamento de tarefas e gargalos de tempo de processamento. E mais, um sistema rodando de forma centralizada exige especial cautela contra falhas e ataques, principalmente aqueles que lidam com tarefas sensíveis onde falhas podem custar tempo, recursos ou mesmo vidas.

Em contra-partida, sistemas que possuem estrutura distribuída apresentam maior robustez a falhas e ataques, uma vez que a queda de um nó não implica necessariamente a inutilização de todo o sistema. Ademais, essa estrutura distribuída permite um paralelismo verdadeiro, uma vez que o número de processos concomitantes em dispositivos distintos é polinomialmente maior que aquele apresentado por sistemas centralizados, mesmo no caso de sistemas multiprocessados.

Adicionalmente, o modelo de Sistemas Cognitivos Distribuídos pode incluir, como componentes, dispositivos físicos e virtuais distintos de computadores tradicionais (GIBAUT; GUDWIN, 2020). Tal possibilidade está de acordo com Wu *et al.* (2014), que

introduz o paradigma da Internet das Coisas Cognitiva, indo além da simples conexão entre dispositivos, para enfatizar a importância de sistemas capazes de percepção, aprendizado e adaptação.

1.1.3 Por Quê *Cognitive Twins*?

Na tentativa de contribuir para o campo de pesquisa da AGI, o autor deste trabalho defende que a melhor maneira de abordar o problema de projetar máquinas que *pensam* como humanos (no sentido de possuirem estruturas internas não observáveis) é primeiro projetar máquinas que se *comportam* como humanos. Ou seja, se visamos atingir um objetivo tão surpreendente como ter uma mente artificial semelhante à humana — com todos os seus processos internos e representações ainda desconhecidas — devemos primeiro conseguir entender e reproduzir o comportamento humano observável de maneira satisfatória e progressivamente prosseguir com investigação mais aprofundada, compreendendo detalhes sobre o funcionamento da mente na totalidade. Observe que nesse estudo não se presume perfeição: como disse o pai da Ciência da Computação, Alan Turing: "Se é esperado de uma máquina que seja infalível, ela também não pode ser inteligente" (TURING, 1995). A pesquisa em máquinas semelhantes a humanos (*human-like*) é mais sobre falhar como um humano do que gerar máquinas formidavelmente habilidosas. É mais sobre ser seguro e confiável do que ser preciso e infalível.

Neste trabalho, toma-se a centralidade humana como uma filosofia: máquinas construídas PARA humanos. Isso significa tanto considerar a ética e a justiça — existir para o benefício humano — quanto definir representações adequadas, similares às que povoam a mente humana, incluindo-se sinais sensoriais, linguagem, conhecimento, habilidades, motivos, intenções etc, para que assim essas máquinas *sejam* como humanos.

O conceito de *Cognitive Twin* será melhor explorado no capítulo 3, mas por hora podemos defini-lo como uma réplica digital dos processos dinâmicos e cognitivos de uma pessoa, voltada a uma representação parcial desta pessoa, não apenas em relação à duplicação de seu comportamento observável, na forma de um agente virtual, que represente tal pessoa, ou a substitua em atividades específicas, mas que permita também um aprofundamento na investigação de seus processos internos, que a fazem ser como é. O estudo sobre tal assunto representa um caminho um pouco mais direto na busca por máquinas que pensem como humanos.

Entre as possíveis dificuldades em construir *Cognitive Twins* partindo de uma pessoa real — e aqui diz-se possíveis porque não foi feito — estão os diversos níveis de necessidades humanas e as influências de fatores cognitivos não diretamente relacionados a uma plena racionalidade, como emoções, humores e necessidades que parecem intrínsecas à nossa condição. Emoções, sentimentos e humores afetam as decisões humanas

de modo ainda não totalmente conhecido, mas estudos em computação afetiva já são uma realidade. Quanto às necessidades humanas, existem alguns modelos que acabam por nortear a Literatura, como a demasiadamente simplista Hierarquia de Necessidades de Maslow (MASLOW, 1981). Esse modelo, ilustrado na Figura 2 na forma de uma pirâmide, aparenta ser um modelo razoável de necessidades que poderia ser implementado computacionalmente com poucos problemas. No entanto, uma série de críticas são feitas a este modelo, tais como:

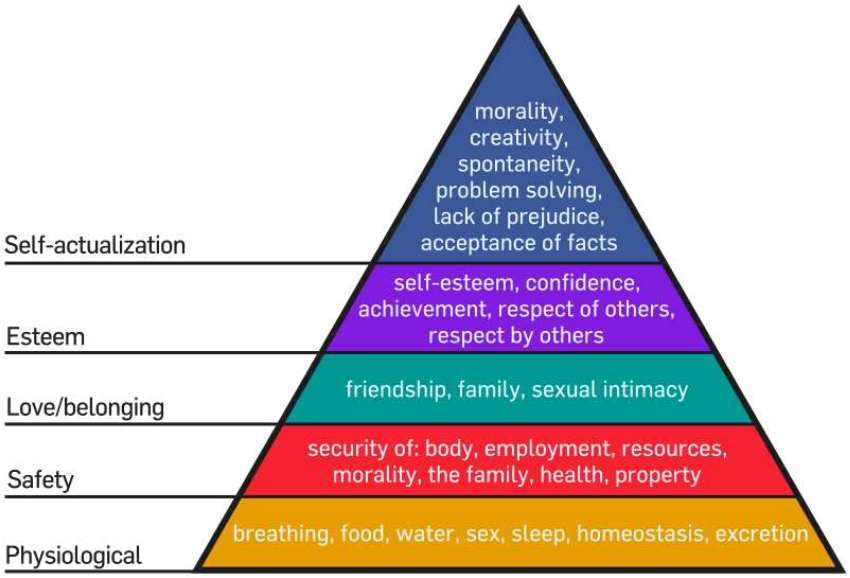


Figura 2 – Hierarquia de Necessidades de Maslow. Créditos *Creative Commons*

1. Abordagem não científica: estudo foi feito a partir das biografias de 18 indivíduos escolhidos diretamente pelo autor.
2. Falta de suporte empírico que valide a teoria.
3. Estrutura classista e elitista, derivada da forma como o estudo foi conduzido.
4. Alta quantidade de exceções, reconhecidas pelo próprio autor.
5. Forte enviesamento étnico e cultural nas pessoas estudadas.
6. Necessidades nem sempre são organizadas de forma hierárquica.

Sobre os pontos 5 e 6, Tay e Diener (2011) mostram que, apesar de certas necessidades serem razoavelmente universais nos grupos estudados, a cultura e a sociedade onde os indivíduos vivem influencia bastante nas ênfases das necessidades. A busca pela satisfação dessas necessidades também tem influência independente no bem-estar subjetivo, com indivíduos buscando o atendimento de suas necessidades psicológicas, antes mesmo de ter

suas necessidades mais básicas atendidas. Tem-se inclusive na grande Carolina Maria de Jesus, escritora negra, um exemplo disto, ao escrever algumas de suas obras *enquanto* passava fome. Como modelar essas necessidades sem recair em interpretações simplistas de como um ser humano deveria ser?

É inegável a influência de ambos os pontos levantados nas tomadas de decisão humanas, em especial quando essas influências passam desapercebidas, como quando toma-se uma decisão que se sabe que é ruim, mas feita com a intenção de estreitar laços ou impressionar alguém. É neste ponto que o uso de Arquiteturas Cognitivas fornece meios de mitigar esse problema: ao adotar e implementar computacionalmente teorias vindas diretamente das ciências cognitivas, esses modelos fornecem um norte para alcançar as respostas para esses problemas.

Assim, nos contextos já citados, o autor deste trabalho defende que a pesquisa em *Cognitive Twins* e suas diversas abordagens, em especial utilizando Arquiteturas Cognitivas, é um importante caminho para o avanço em AGI.

1.1.4 Outros Fatores

Saindo um pouco das motivações técnicas, tem-se também a questão do acesso à capacidade de criação, em relação à tecnologia moderna, por aqueles que não se encontram nas chamadas "*Big Techs*" ou que não habitam países centrais do Capitalismo. Como a direção que se toma em relação à IA consiste basicamente em modelos cada vez maiores, treinados em *datasets* igualmente grandes — muitas vezes construídos com contribuição aberta, mas sintetizados com privilégios privados, como aqueles relativos ao site "StackOverflow" por exemplo — isso exige máquinas poderosas com custos altíssimos e que consomem energia em quantidades abundantes, consequentemente, possuem grandes pegadas de carbono (do original em inglês, *carbon footprint*) (BENDER *et al.*, 2021; STRUBELL *et al.*, 2019; HENDERSON *et al.*, 2020). Como consequência dessa direção e centralização de poder, o desenvolvimento da IA fica cada vez mais distante das mãos do cidadão comum, em especial daqueles pertencentes a grupos não-privilegiados da sociedade, seja isto em decorrência de raça, gênero, religião ou outros fatores discriminatórios.

O parágrafo acima se relaciona com a tese em questão no ponto em que foge da ideia simplista de *Bigger is Better* e visa modelar agentes inteligentes seguindo padrões estabelecidos nas ciências cognitivas. Essa tomada de direção divergente é não apenas benéfica do ponto de vista técnico, mas também do ponto de vista social, em especial se considerarmos a crescente onipresença dos sistemas computacionais que se valem, em algum nível, de inteligência artificial. Esses sistemas estão presentes desde serviços bem simples, onde sua existência é quase transparente para olhos destreinados, como provadores virtuais em lojas de departamento, até sistemas bastante sofisticados que vêm ga-

nhando popularidade, como assistentes domésticos (Alexa, Google Home etc) capazes de reconhecer voz, diferenciar cada morador e mesmo tomar ações levemente diferenciadas, caso o mesmo comando seja executado por pessoas diferentes.

Soma-se a esses fatores a crescente conectividade entre sistemas — físicos ou digitais — na qual a sociedade está atualmente imersa, com o acesso ampliado e popularizado a dispositivos da Internet das Coisas (IoT, sigla em inglês para *Internet of Things*), múltiplos dispositivos móveis, bem como diversos serviços online, que se somam aos já existentes computadores e tablets, tudo isso facilitado por tecnologias habilitadoras como redes de comunicação 5G e serviços que fornecem de forma simplificada acesso a modelos de Aprendizado de Máquina.

Em tempo, no momento da escrita deste trabalho, o mundo passa por uma crise de circuitos integrados devido, em parte, à pandemia de COVID-19, agravando em muito os custos de acesso a computadores de alta capacidade computacional (HOWLEY, 2021; SWENEY, 2021; LEPRINCE-RINGUET, 2021).

Por essas motivações acima descritas, algumas técnicas e outras não-técnicas, o autor deste trabalho defende o uso de modelos de IA distribuídos — em especial de Sistemas Cognitivos — em dispositivos de baixa capacidade computacional, como uma direção de pesquisa válida, enriquecedora e que se apresenta como alternativa de possível solução aos problemas e questões levantados.

1.2 Sobre este Trabalho

Além desta introdução, na qual são apresentadas algumas das motivações de pesquisa, o restante deste trabalho está dividido em outros cinco capítulos. No capítulo 2 é apresentada uma revisão bibliográfica acerca dos temas mais relevantes para o entendimento da pesquisa. O capítulo 3 dedica-se inteiramente a um único tema (*Cognitive Twins*, ou *Cognitive Twins*, em português) pois o mesmo apresenta grande importância na aplicação desenvolvida. O capítulo 4 descreve a ferramenta base para sistemas cognitivos desenvolvida no escopo deste projeto, enquanto o capítulo 5 traz uma proposta de método de desenvolvimento de *Cognitive Twin* utilizando uma abordagem de Sistemas Cognitivos Distribuídos. O capítulo 6 apresenta a metodologia (softwares utilizados, métricas, considerações do design dos experimentos etc), os resultados e discussões acerca dos experimentos propostos. Em todos os capítulos citados acima, existe uma sessão de resumo ao final. Por fim, o capítulo 7 traz uma conclusão e especulações sobre trabalhos futuros relacionados a este trabalho.

2 Revisão Bibliográfica

Neste capítulo, apresentamos uma revisão bibliográfica sobre alguns temas transversais, que julgamos relevantes para o bom entendimento deste trabalho, dedicando uma seção para cada um deles. Particularmente, dentre os diversos temas transversais que subsidiam essa tese, um tema que consideramos que merece um destaque especial "*Cognitive Twins*" será apresentado à parte no capítulo 3.

2.1 Arquiteturas Cognitivas

A área de pesquisa em Arquiteturas Cognitivas tem suas origens na década de 70 do século passado, quando se diferenciou dos estudos sobre sistemas especialistas e engenharia do conhecimento, sub-áreas da assim chamada inteligência artificial clássica. Em seus primórdios, de maneira análoga a suas bases, tinha como objetivo investigar como softwares podem tratar, racionalmente, problemas ao longo de diferentes domínios, trazer *insights*, adaptar-se a novas situações e, em última instância, refletir sobre si mesmos. Porém, evoluindo a partir de suas origens na inteligência artificial clássica, a área ganhou um rumo próprio, e hoje podemos dizer que o objetivo último da pesquisa em Arquiteturas Cognitivas é fornecer modelos computacionais para teorias cognitivas que expliquem a mente humana e, eventualmente, construir IAs que apresentem nível similar ao humano. Assim, as Arquiteturas Cognitivas têm uma grande interface com as Ciências Cognitivas, de onde buscam inspiração e fornecem evidências sobre os mecanismos que demonstram sucesso em emular comportamento inteligente, contribuindo, tanto para as pesquisas em computação (inteligência artificial) como de fato para a própria área de Ciências Cognitivas.

Entre as bases precursoras da área encontram-se os trabalhos de Newell e Simon (NEWELL; SIMON, 1961; SIMON; NEWELL, 1971), os sistemas especialistas, os sistemas de produção e os sistemas baseados em conhecimento. Uma das mais tradicionais Arquiteturas Cognitivas, a ACT-R (ANDERSON *et al.*, 2004; ANDERSON, 2009), ilustrada na Figura 3, foi desenvolvida por Anderson baseada em sua teoria para a origem do conhecimento humano (ANDERSON, 1989). Concomitantemente, Laird, baseado nas ideias de Newell *et al.* (1989), lança as primeiras versões de outra importante Arquitetura, o SOAR (LAIRD; ROSENBLOOM, 1996; LAIRD, 2012). Ao final da década de 90, mais consolidada, a área vivencia importantes contribuições acerca das definições e do que se espera de um modelo para que o mesmo seja considerado uma Arquitetura Cognitiva: um conjunto essencial de estruturas e processos necessários para a geração de um modelo cognitivo-computacional, passível de ser utilizado em diversas áreas relacionando

cognição e comportamento. Neste período, Sloman (2002), começa a lidar com a questão "arquitetural" para representar conceitos mentais, e Sun (2004) descreve os requisitos necessários para a construção de uma Arquitetura Cognitiva. Na sequência, o próprio Sun (2007) aborda questões e desafios no desenvolvimento de arquiteturas cognitivas. Langley *et al.* (2009) analisam diversas Arquiteturas Cognitivas, avaliando os progressos na área de pesquisa. Estes estudos indicaram que a principal vantagem em se caracterizar uma arquitetura como cognitiva é a possibilidade de se obter um *framework* concreto para a modelagem cognitiva, de tal forma que esses modelos pudessem ser testados de maneira concreta. Tal caracterização permitiria a definição de um conjunto de estruturas, módulos e processos básicos, criando uma linguagem comum para que diferentes pesquisadores pudessem explorar suas propostas de modelos cognitivos.

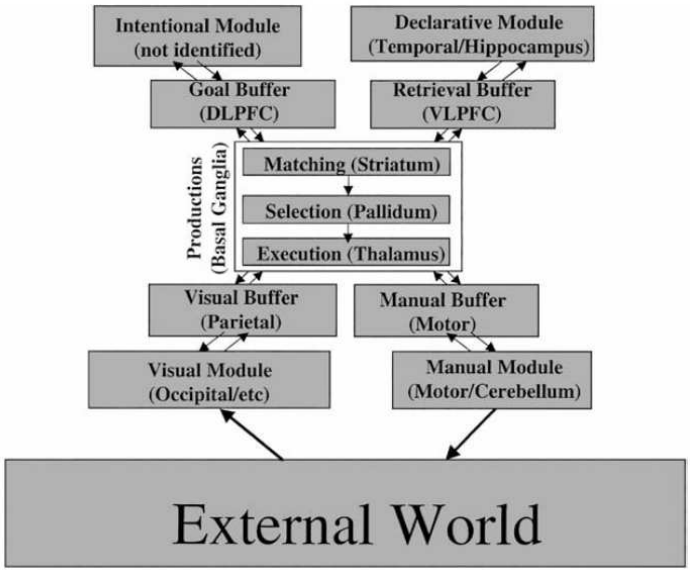


Figura 3 – Modelo estrutural da Arquitetura Cognitiva ACT-R 5.0. Extraído de Anderson *et al.* (2004)

Devido à variedade de abordagens que podem (em tese) colaborar para o desenvolvimento de uma IA similar à inteligência humana e, na falta de uma definição única, geral e clara de uma Teoria da Cognição, cada Arquitetura Cognitiva é construída assumindo seu próprio conjunto de premissas e considerações, sempre inspiradas em teorias vindas das Ciências Cognitivas, mas podendo ou não concordar entre si. Desta forma, se faz necessário definir o entendimento do autor sobre o que constitui uma Arquitetura Cognitiva. Podemos entender uma Arquitetura Cognitiva como um sistema de controle de propósito geral, inspirado em teorias científicas desenvolvidas para explicar o processo de cognição no ser humano e em animais. O desenvolvimento de uma Arquitetura Cognitiva envolve todo um espectro de capacidades do fenômeno cognitivo, tais como percepção, atenção, memória, raciocínio, aprendizagem, geração de comportamento e outros. Além disso, tais arquiteturas funcionariam também como modelos teóricos de processos cog-

nitivos, permitindo que tais teorias pudessem ser testadas e reutilizadas em diferentes aplicações.

Dentre as aplicações na qual Arquiteturas Cognitivas são empregadas temos: Robótica (HUNTSBERGER; WOODWARD, 2011; BROOKS, 1989; GUDWIN *et al.*, 2020; GIBAUT; GUDWIN, 2019), Experimentos Psicológicos (SUN; ZHANG, 2019), Modelagem de Desempenho Humano (ALLENDER, 2000; JONES *et al.*, 1999; LAIRD *et al.*, 1998), Interações Humano-Computador (FAN *et al.*, 2010), Classificação e Agrupamento (CUI *et al.*, 2016; MELIS *et al.*, 2009) e Agentes Virtuais (EVERTSZ *et al.*, 2009; SCHAAT *et al.*, 2013; HELJAKKA *et al.*, 2007; GIBAUT, 2018; GUDWIN *et al.*, 2018), dentre outras.

Embora não tão popular como os estudos da IA que lidam com modelos puramente conexionistas (como *Deep Learning*), a área de Arquiteturas Cognitivas mantém-se ativa como uma possível direção rumo à AGI: enquanto alguns acadêmicos especialistas em Redes Neurais voltam-se apenas agora para a importância de um Processamento Simbólico na emulação mais acurada da mente humana (mais especificamente na união entre Conexionismo e Simbolismo), diversas propostas de Arquiteturas Cognitivas surgiram ao longo desses anos para tentar sanar este problema. Aqui, a ideia é aproveitar-se das representações explícitas de conhecimento, do aprendizado dedutivo e da explicabilidade dos Sistemas Simbólicos, unido-as ao aprendizado indutivo de métodos conexionistas. Apesar disso, várias questões permanecem em aberto, tais como quais tipos de memória são necessários, como modelar processos de formação de hábitos e adaptação, dentre outras.

Apesar de tais questões permanecerem inconclusas, diversas arquiteturas cognitivas foram propostas (GOERTZEL *et al.*, 2014). Em 2010, Samsonovich (2010) desenvolveu uma tabela comparativa apresentando uma revisão sistemática de inúmeras arquiteturas cognitivas conhecidas e documentadas. Kotseruba e Tsotsos (2020) trazem ainda um grande *Review* com um recorte de 84 itens (de um total de 195, segundo os próprios autores) no qual considerações de taxonomia, mecanismos cognitivos implementados etc., são ressaltados.

2.1.1 Agentes Inteligentes e Agentes Cognitivos

Um conceito recorrente quando tratamos de Arquiteturas Cognitivas — devido à própria natureza dessas entidades — é o conceito de Agente Inteligente. De entendimento intuitivo, quando nos referimos a agentes humanos e de compreensão um pouco mais nebulosa conforme se passa a agentes artificiais, a tal conceito foram atribuídas diversas definições, ao longo dos anos. A mais clássica delas, apresentada por Russell e Norvig (2020), define um Agente como qualquer coisa que possa ser vista como *percebendo* o seu ambiente através de sensores e *agindo* sobre ele através de atuadores, conforme ilustrado

na Figura 4.

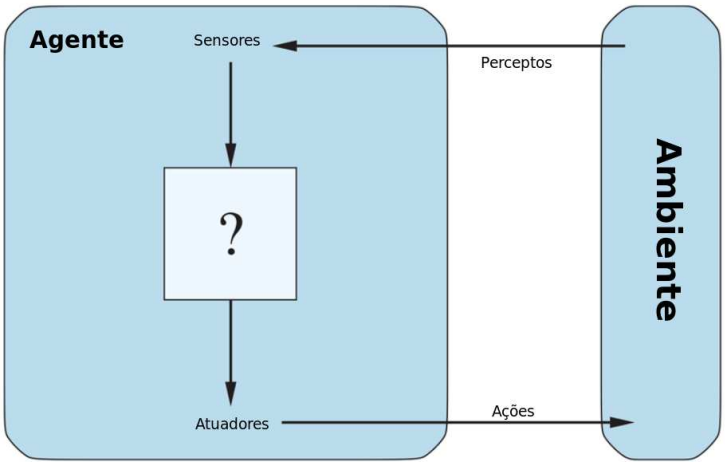


Figura 4 – Modelo de Agente (Adaptado) apresentado por Russell e Norvig (2020)

Segundo essa definição, um humano pode ser visto como um agente cujos sensores são olhos, ouvidos e demais órgãos sensoriais e cujos atuadores são as mãos, pernas, aparelho vocal etc. Um robô poderia ter câmeras e lasers como sensores e diversos motores como atuadores. Um agente em software recebe arquivos, pacotes de rede e entradas humanas como entradas sensoriais e age sobre o ambiente na forma de arquivos enviados, exibição de informação etc. É importante tomar nota, porém, que no caso do software o mesmo pode depender de entidades externas para perceber ou atuar sobre o ambiente, e isso pode trazer uma certa estranheza. Pensando nisso, Franklin e Graesser (1997) definem um *agente autônomo* como um sistema que simultaneamente é parte integrante e está situado em um ambiente e percebe e age sobre o mesmo ao longo do tempo, buscando atingir seus próprios objetivos em um momento futuro. Neste trabalho, Franklin e Graesser discutem um pouco sobre as definições existentes antes de fornecer definição própria.

Ainda assim Franklin e Graesser (1997) definem *agentes autônomos*, o que não "encerra" a questão da definição e não escapa completamente de entidades simples que atendem à definição. Um termostato cumpre essa definição de agente: percebe o ambiente através de um termômetro e age sobre o mesmo através do chaveamento de um ar-condicionado ou aquecedor, tendo o objetivo de manter a temperatura ambiente em uma faixa, atendendo simultaneamente a ambas as definições. O autor deste trabalho defende que ambas as definições, Agentes Autônomos e Agentes, operam sob uma ótica do modelo e não em uma verdade filosófica absoluta do que é inteligência. Não levar isto em consideração pode nos levar à falácia da projeção mental (JAYNES, 1990).

Existem diversas categorias nas quais poderíamos classificar e dividir os tipos de agente, incluindo categorias que se sobrepõe parcialmente. Dentre essas possibilidades citadas, destacam-se, para desenvolvimento do trabalho, as categorias de Agentes Reativos (ou reflexivos) que meramente executam uma função sobre os dados sensoriais — reagem

ao ambiente — e os Agentes Deliberativos, os quais atuam de forma total ou parcialmente planejada a fim de atingir seus objetivos (RUSSELL; NORVIG, 2020). Note que um Agente Inteligente não precisa enquadrar-se totalmente em uma ou outra categoria, mas pode encontrar-se em qualquer ponto entre esses polos. Ainda: a avaliação de um Agente Inteligente e de seu comportamento segue aproximadamente o que na filosofia entende-se como ”consequencialismo”: seu comportamento é avaliado conforme as consequências de seus atos sobre o ambiente no qual está inserido. Uma sequência de ações do agente levam o ambiente a uma determinada sequência de estados e, caso tal sequência seja desejável, diz-se que o agente possui um bom desempenho. Essa noção de desejável varia conforme as métricas de desempenho adotadas para avaliar a sequência de estados do ambiente.

Em seres humanos, é possível identificar diversas manifestações cognitivas, que podem diferir de um indivíduo para outro. Essas manifestações, tais como a percepção, a imaginação, a abstração, o reconhecimento e a associação de padrões, a emoção, o raciocínio, a tomada de decisões, a memória, a aprendizagem, a linguagem, o planejamento etc, também chamadas de capacidades cognitivas, envolvem processos (ainda em parte desconhecidos) por meio do qual o ser humano consegue conhecer seu ambiente e modelá-lo de tal forma a prever seu comportamento e atuar sobre ele para modificá-lo de acordo com seus propósitos, da maneira tão eficiente quanto se queira (e seja possível). Em Ciência Cognitiva, essas capacidades são modeladas de diversas maneiras, resultando em diferentes modelos cognitivos que tentam explicar como a mente humana funciona (JOHNSON-LAIRD, 1980). Quando nos servimos destes modelos cognitivos para a criação de mentes artificiais para Agentes Inteligentes, nos referimos a esses agentes como sendo Agentes Cognitivos.

2.2 Internet das Coisas

O termo ”Internet das Coisas” (IoT, do inglês *Internet of Things*) foi cunhado no final dos anos 90 por Kevin Ashton para descrever um sistema no qual os dispositivos poderiam ser conectados à Internet através da tecnologia de identificação por radiofrequência (RFID) (ROSE *et al.*, 2015). Atualmente, não existe uma definição fechada de IoT, mas certamente esse termo se aplica a um novo conceito de internet que deve evoluir para realmente se tornar a infraestrutura de dispositivos para conversar com serviços e interagir com outras entidades computacionais sem intervenção humana (WALDO, 2002). Além disso, está previsto que os seres humanos sejam gradualmente movidos para fora do circuito e possam se tornar uma minoria como geradores e receptores de informações. Os principais atores serão dispositivos que apresentam comportamento autônomo e proativo, conhecimento do contexto e comunicação colaborativa (ATZORI *et al.*, 2010).

A Internet das Coisas envolve uma infinidade de ideias entrelaçadas de diferentes

abordagens. Essas abordagens estão agrupadas em três categorias que se sobrepõem e se relacionam, sendo brevemente descritas da seguinte forma (ATZORI *et al.*, 2010):

- Orientadas a Coisas: concentra-se na visibilidade do objeto, como a rastreabilidade de um objeto e o conhecimento de seu status, localização atual etc.;
- Orientadas à Internet: foco na melhoria dos protocolos subjacentes da Internet em direção a uma infraestrutura global que conecta objetos genéricos virtuais e físicos;
- Orientadas à Semântica: concentra-se em questões relacionadas a como representar, armazenar, interconectar, pesquisar e organizar as informações geradas pela IoT, como soluções de modelagem apropriadas para descrição de itens, raciocínio sobre dados, armazenamento escalável etc.

A Figura 5 ilustra as abordagens mencionadas.

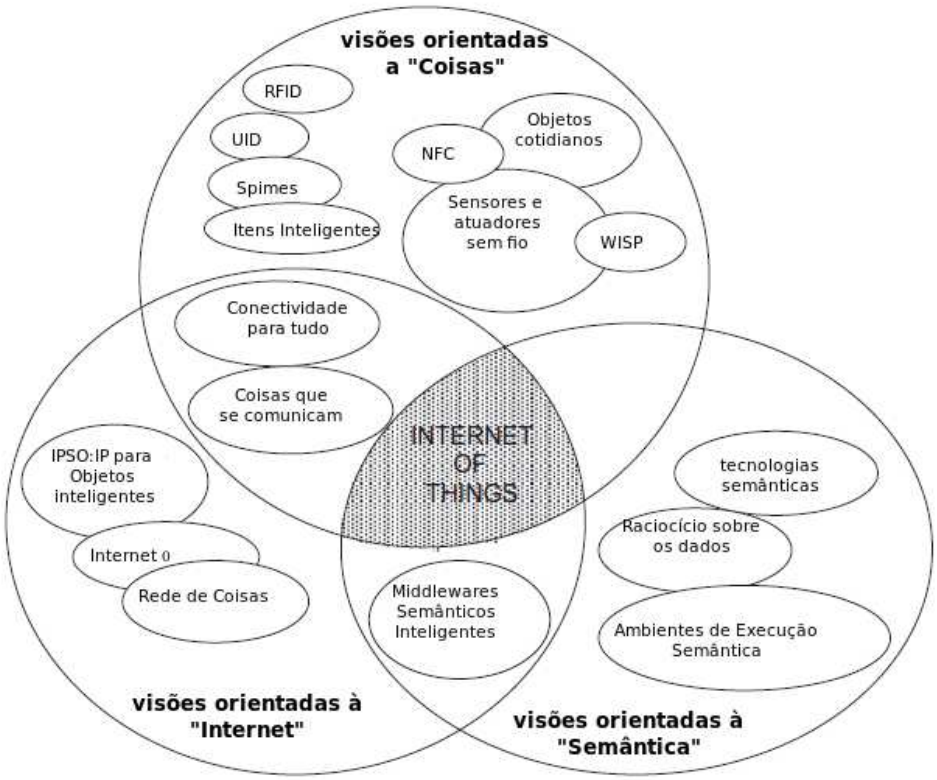


Figura 5 – As diferentes visões sobre IoT e a emergência de um paradigma. Adaptado de Atzori *et al.* (2010)

Embora essas abordagens difiram, todas elas envolvem cenários nos quais a conectividade de rede e a capacidade computacional se estendem a itens que não são considerados computadores. Esses "dispositivos inteligentes" geram, trocam e processam dados com mínima ou nenhuma intervenção humana. Além disso, os sistemas de IoT derivados

dessas diversas visões compartilham os desafios básicos de pesquisa que abrangem seus domínios técnicos.

A IoT fornece uma inovação profunda em tecnologias para casa, saúde, transporte e gerenciamento de recursos naturais, conforme ilustrado na Figura 6. Portanto, tem o potencial de transformar a maneira como vivemos e trabalhamos, pois permite que objetos do mundo real troquem informações e interajam com pessoas para apoiar processos de negócios e criar conhecimento. É importante notar que IoT não é uma nova internet, mas uma extensão dela (HOLLER *et al.*, 2014).

Figura 6 – Exemplos de domínios de aplicação de IoT. Adaptado de Holler *et al.* (2014)

Abaixo, as soluções de IoT mais notáveis (ou casos de uso) são descritas. Eles foram classificados em cinco categorias diferentes, cada uma com suas peculiaridades: dispositivos vestíveis inteligentes, casas inteligentes, cidades inteligentes, ambientes inteligentes e empresas inteligentes (PERERA *et al.*, 2015).

- **Dispositivos Vestíveis Inteligentes**

  Dispositivos vestíveis são definidos como dispositivos eletrônicos móveis que podem ser embutidos discretamente na roupa do usuário, como parte da roupa ou como um acessório. Ao contrário dos dispositivos móveis convencionais, eles podem funcionar e serem acessados sem ou com o mínimo de interferência dos usuários. Esses

dispositivos geralmente conseguem reconhecer a atividade ou o estado do usuário e vão desde microssensores integrados em tecidos até eletrônicos embutidos em acessórios da moda, como relógios, óculos e cintos (LUKOWICZ *et al.*, 2004). O desenvolvimento de uma arquitetura integrada de serviços domésticos inteligentes com dispositivos vestíveis para conforto doméstico e bem-estar é certamente uma solução interessante para manter e impulsionar a pesquisa nesta área (CHAN *et al.*, 2012). As principais questões em aberto são: eficiência do sistema, confiabilidade e discrição, percepção e aceitação do usuário, interoperabilidade, legislação, questões éticas entre outras. Mais detalhes em Chan *et al.* (2012);

- ***Smart Home* (Casa Inteligente)**

  As soluções de casa inteligente visam tornar a experiência de viver em casa mais conveniente e agradável para os moradores. Algumas soluções de casas inteligentes também se concentram em auxiliar idosos em suas atividades diárias e no monitoramento de cuidados de saúde (KWON *et al.*, 2010). O potencial de mercado é elevado, atraindo gradualmente mais investimentos e soluções inovadoras. Dispositivos e plataformas tais como Alexa (Amazon), Google Home (Google), lâmpadas e tomadas inteligentes, entre outras coisas, podem ser encontradas hoje no mercado a preços cada vez mais acessíveis;

- **Cidades Inteligentes**

  Não existe uma definição formal e amplamente aceita de *Smart City*, mas certamente seu conceito aborda um melhor uso dos recursos públicos, aumentando a qualidade dos serviços oferecidos aos cidadãos, e simultaneamente reduzindo os custos operacionais das administrações públicas (ZANELLA *et al.*, 2014). Algumas das características que permeiam o conceito de Cidades Inteligentes são (GIFFINGER; GUDRUN, 2010):

  – governança inteligente: participação na tomada de decisões, serviços públicos e sociais, governança transparente, estratégias e perspectivas políticas;

  – mobilidade inteligente: acessibilidade local, acessibilidade (inter)nacional, disponibilidade de infraestrutura de TIC, sistemas de transporte sustentáveis, inovadores e seguros;

  – ambiente inteligente: sem poluição sob condições naturais ou controle da poluição, proteção ambiental, gestão sustentável de recursos;

  – *smart living*: equipamentos culturais, condições de saúde, segurança individual, qualidade habitacional, equipamentos educativos, turismo e coesão social.

  A implementação dos conceitos que envolvem Cidades Inteligentes requer o desenvolvimento de novas infraestruturas de Internet, aplicativos em rede e parcerias com

partes interessadas. Uma vez que as estratégias de desenvolvimento e as avaliações de custo-benefício estejam disponíveis, as cidades devem estabelecer prioridades em relação a aplicações social e economicamente interessantes que devem estar alinhadas com os objetivos estratégicos de desenvolvimento econômico e social das áreas da cidade (KOMNINOS *et al.*, 2011).

Sob uma ótica das ciências sociais, Bria e Morozov (2020) ainda observam que muitos dos conceitos sobre o assunto que ganham espaços — acadêmicos, jornalísticos e de mercado — estão intrinsecamente ligados a conceitos neoliberais, mas que tecnologia, democracia, o direito à cidade e a soberania digital podem caminhar juntas em resistência (e como alternativa) a um capitalismo predatório;

- **Ambientes inteligentes**

  De acordo com Gubbi *et al.* (2013), ambiente inteligente é uma "Interconexão de dispositivos, sensores e atuadores que fornecem a capacidade de compartilhar informações entre plataformas por meio de uma estrutura unificada, desenvolvendo uma imagem operacional comum para permitir aplicativos inovadores, detecção em grande escala, análise de dados e representação de informações usando detecção onipresente de ponta e computação em nuvem". Monitoramento da qualidade do ar e da água, monitoramento de desastres naturais e fazendas inteligentes são alguns dos exemplos que aqui se enquadram;

- **Empresas Inteligentes**

  As soluções de IoT Corporativa são projetadas para oferecer suporte a infraestrutura e funcionalidades de uso geral em locais industriais, como gerenciamento e conectividade.

2.2.1 Tendências de IoT

Existem diversas tendências tecnológicas que, em algum nível, envolvem IoT. Abaixo, destacamos algumas.

2.2.1.1 IoT e *Big Data*

Os dispositivos de sistemas IoT fornecem grandes fluxos de dados contínuos sem intervenção humana. Portanto, minerar e analisar essa abundância de dados para obter informações valiosas requer habilidades sólidas de análise de *big data* (XU *et al.*, 2014). Os algoritmos de *big data* podem examinar grande volume de dados de IoT e procurar correlações estatísticas e semânticas para determinar grupos de características relacionadas entre os usuários. No entanto, esses algoritmos são suscetíveis a uma categorização injusta dos usuários, recrudescendo preconceitos e problemas presentes nos dados (em especial

em dados sociais). Além disso, a integração de dispositivos IoT com recursos externos, como sistemas de software tradicionais e serviços da Web, requer o desenvolvimento de várias soluções de *middleware*. Nesse sentido, construir aplicações práticas, nas quais dados heterogêneos relacionados à IoT são combinados com dados tradicionais, pode ser uma tarefa desafiadora para diversos provedores de serviços (ROSE *et al.*, 2015).

2.2.1.2 IoT e Computação em Nuvem

As plataformas em nuvem fornecem uma boa maneira de conectar os dispositivos e nos permitem acessá-los na Internet. Pesquisas futuras se concentrarão na implementação de novos modelos e/ou plataformas que forneçam "sensing as a service" na nuvem (XU *et al.*, 2014). O conceito de "sensing-as-a-service" surgiu do contexto de Computação em Nuvem e visava rentabilizar a enorme quantidade de dados de sensores que os sistemas IoT disponibilizam (GIFFINGER; GUDRUN, 2010). Outro aspecto interessante é a centralidade das redes baseadas em nuvem que criam oportunidades para novas pesquisas sobre sistemas de detecção/prevenção de anomalias e intrusões e controle de acesso, permitindo assim que a rede bloqueie o tráfego indesejado ou detecte comportamentos suspeitos (ALUR *et al.*, 2016).

2.2.1.3 IoT Cognitiva

Resumir todo o estudo por trás da IoT somente à questão da conectividade não é suficiente. Os objetos relacionados à IoT devem poder aprender, pensar e entender seus ambientes físicos e sociais sozinhos. Isso levou o mundo da IoT a um novo paradigma, chamado Internet das Coisas Cognitiva (CIoT, sigla do inglês *Cognitive Internet of Things*), a fim de dotar os atuais sistemas da IoT com capacidades para a inteligência de alto nível. Assim, Wu *et al.* (2014) propõem uma estrutura CIoT baseada principalmente nas interações entre cinco tarefas cognitivas fundamentais: ciclo de percepção-ação, análise massiva de dados, derivação semântica e descoberta de conhecimento, tomada de decisão inteligente e provisionamento de serviços sob demanda.

2.2.2 Principais Desafios e Problemas em Aberto da IoT

Embora os sistemas IoT sejam implantados em diferentes domínios, eles compartilham os desafios fundamentais de pesquisa que, em última análise, abordam as principais preocupações levantadas. A lista a seguir não é um catálogo abrangente de todos os desafios da IoT, mas visa destacar os principais problemas em cada área central (ROSE *et al.*, 2015; ALUR *et al.*, 2016; DOMINGUE *et al.*, 2011).

- Escalabilidade: Os protocolos de rede ainda são projetados principalmente para dispositivos estacionários com recursos computacionais razoáveis. A internet das coisas

requer uma enorme escalabilidade no espaço de rede para lidar com o enorme aumento do número de dispositivos. Além disso, muitos dos dispositivos são smartphones conectados intermitentemente, exigindo protocolos e arquiteturas de rede mais dinâmicos e adaptáveis para que os sistemas IoT enfrentem esses desafios de escalabilidade;

- Multilocação: Multilocação (do inglês *Multitenancy*) é uma arquitetura na qual um único aplicativo emula várias instâncias para atender a vários clientes. Cada cliente, o *guest*, pode personalizar partes do aplicativo, mas não seu código. Na IoT, os *guests* são dispositivos dinâmicos, conectados intermitentemente e, portanto, demandam uma infraestrutura de rede virtual para diferentes dispositivos e serviços. É análogo aos provedores de nuvem que fornecem, a cada *guest*, sua própria visão abstrata do *data center*. Cada rede virtual deve ter sua própria configuração e compartilhar recursos, mas sempre mapeada para uma infraestrutura física compartilhada ao nível inferior (AZEEZ *et al.*, 2010);

- Comunicação de baixa potência: Espera-se que os sistemas IoT integrem dispositivos com todas as formas de consumo de energia. Muitos desses dispositivos não têm acesso a uma fonte de alimentação contínua e o tamanho e o custo da bateria impõem restrições significativas sobre como esses dispositivos computam e se comunicam (ALUR *et al.*, 2016). Portanto, os dispositivos IoT devem ter interfaces de rede personalizadas, sistemas operacionais e modelos de programação que façam o uso mais eficaz de recursos limitados de computação e energia;

- Questões de segurança e privacidade de rede: os dispositivos IoT são heterogêneos, ocasionalmente conectados e podem não ser dotados de mecanismos de defesa adequados. Consequentemente, a relação de confiança entre dispositivos em constante mudança é difícil e brechas de segurança podem ser abertas, permitindo falsificações e ataques à rede. Os problemas de segurança de aplicativos IoT incluem acesso à informação e autenticação do usuário, privacidade da informação, destruição e rastreamento do fluxo de dados, estabilidade da plataforma IoT etc (JING *et al.*, 2014). Por outro lado, as características das arquiteturas de multilocação criam uma oportunidade para novas pesquisas sobre detecção de anomalias, sistemas de detecção/prevenção de intrusão e controle de acesso, para que a rede possa bloquear tráfego indesejado ou detectar comportamentos suspeitos (ALUR *et al.*, 2016). IoT geralmente se refere a uma grande rede de dispositivos projetados para coletar dados sobre seu ambiente, que inclui frequentemente dados relacionados a pessoas. É provável que esses dados forneçam benefícios para o proprietário do dispositivo, mas frequentemente também ao seu fabricante ou fornecedor. A coleta e o uso de dados da IoT se tornam uma questão de privacidade, quando os indivíduos têm expectativas de privacidade diferentes do fornecedor, em relação ao escopo e ao uso desses

dados. Esses tipos de problemas de privacidade são essenciais para resolver, porque eles têm implicações em nossos direitos básicos e capacidade coletiva de confiar na Internet (ROSE *et al.*, 2015). Em última análise, é desejável que as pessoas reconheçam sua privacidade e estejam cientes de quais dados podem ser coletados sobre elas e como outras partes podem usar esses dados;

- *Humans in the loop*: Certos aplicativos de IoT envolverão intimamente humanos que devem operar sinergicamente com dispositivos eletrônicos. No entanto, as dependências entre humanos e serviços executando simultaneamente devem ser abordadas. Prevê-se a necessidade de compreender as complexas dependências entre essas aplicações e seres humanos. Na metodologia formal de controle realimentado, existem várias áreas onde um modelo humano pode ser colocado: fora do loop, no controlador, no modelo do sistema e em vários níveis no controle hierárquico. Em algumas aplicações de IoT, os humanos desempenharão papéis em todas essas áreas simultaneamente. A teoria de controle tradicional e as soluções de controle supervisório não são adequadas para lidar com essas questões. Portanto, o desafio é encontrar uma maneira de incorporar o comportamento humano como parte do próprio sistema (ALUR *et al.*, 2016);

- Problemas de desenvolvimento de software

  O número de dispositivos e o volume de informações trocadas aumentarão consideravelmente nos sistemas IoT. Um desafio é fornecer suporte de ferramenta para garantir que os dispositivos sejam configurados corretamente em todos os momentos. As configurações, incluindo software, hardware e rede, devem estar constantemente ativas. Além disso, os sistemas IoT coletam dados e os processam continuamente e decisões críticas em tempo real podem ser exigidas com base no fluxo de informações. Os dispositivos IoT serão geralmente conectados intermitentemente à rede e podem ter restrições limitadas de largura de banda, energia e memória. Essas características trazem dificuldades na depuração interativa, além de limitar a possibilidade de rastrear e armazenar logs detalhados de suas atividades para análise (ALUR *et al.*, 2016). Esses são exemplos de preocupações relevantes ao desenvolver software para sistemas IoT complexos. Novas abordagens na arquitetura de software serão exigidas, onde capacidades cognitivas certamente serão de grande ajuda para dotar os dispositivos com o poder de aprender com os erros e se reparar automaticamente;

- Preocupações com Padrões

  Padrões são fundamentais para desenvolver mercados para novas tecnologias. A padronização e adoção de protocolos que especificam os detalhes de comunicação estão no centro da discussão de interoperabilidade para IoT. Se dispositivos de fabricantes diferentes não usarem os mesmos padrões, a interoperabilidade será mais

difícil, exigindo dispositivos ou mecanismos extras para traduzir de um padrão para outro (ROSE *et al.*, 2015). Além disso, se uma empresa controla diferentes partes de um mercado vertical (por exemplo, geração e aquisição de dados, integração com outros fluxos de dados), ela domina um mercado e, portanto, cria barreiras para participantes menores. Em um ambiente totalmente interoperável, o dispositivo IoT deve ser capaz de se conectar a outros dispositivos ou sistemas e trocar informações, conforme desejado. A interoperabilidade entre dispositivos e sistemas IoT ocorre em graus variados na camada de protocolo de comunicação entre os dispositivos. No entanto, a interoperabilidade total em todos os aspectos de um produto técnico nem sempre é viável ou necessária e, se imposta artificialmente (como por meio de imposições governamentais), provavelmente desencorajará investimentos de mercado e novos empreendimentos comerciais (GUBBI *et al.*, 2013).

2.3 Sistemas de Sistemas

O termo Sistemas de Sistemas (SoS, do original em inglês *Systems of Systems*) começou a aparecer nos anos 90 e continua sendo discutido até hoje. Embora o termo esteja em evidência atualmente, a comunidade ainda não convergiu para um entendimento comum quanto ao seu significado. A Figura 7 mostra uma linha do tempo de contribuições para a área de SoS.

O primeiro pesquisador a mencionar o SoS como um sistema que é mais do que a soma de suas partes foi Boulding (1956). Em resumo, Boulding (1956) defendeu ser necessário entender as partes para entender o todo. Além disso, ele concluiu que, no caso de um SoS, o todo é maior que a mera soma de suas partes, com um objetivo determinado (BOULDING, 1956). Posteriormente, Jackson e Keys propuseram que um SoS é uma inter-relação entre sistemas baseados na solução de problemas, ou seja, sistemas que se unem a outros sistemas, visando resolver problemas de ordem maior (JACKSON; KEYS, 1984). Seguindo Jackson e Keys, Ackoff entendia um SoS como um "conjunto unificado ou integrado" de sistemas (GOROD *et al.*, 2008). Jacob abordou a questão de maneira diferente, mais orientada à natureza e aos sistemas naturais, entendendo que o conceito de SoS está ligado a tudo o que a biologia estuda. Apesar da área apresentar alguns pioneiros, como mostrado acima, apenas em 1989 o termo Sistema de Sistemas foi usado pela primeira vez na Iniciativa de Defesa Estratégica na tentativa de descrever as tecnologias de engenharia de sistemas (JACOB; SPILLMANN, 1974; GPO, 1989; GOROD *et al.*, 2008).

A partir de 1989, a comunidade começou a se interessar mais em entender de maneira mais clara o que seria um Sistema de Sistemas. Nessa época, o conceito de SoS era ainda muito usado de forma ingênua ou mal definida. Foram Eisner et al. e She-

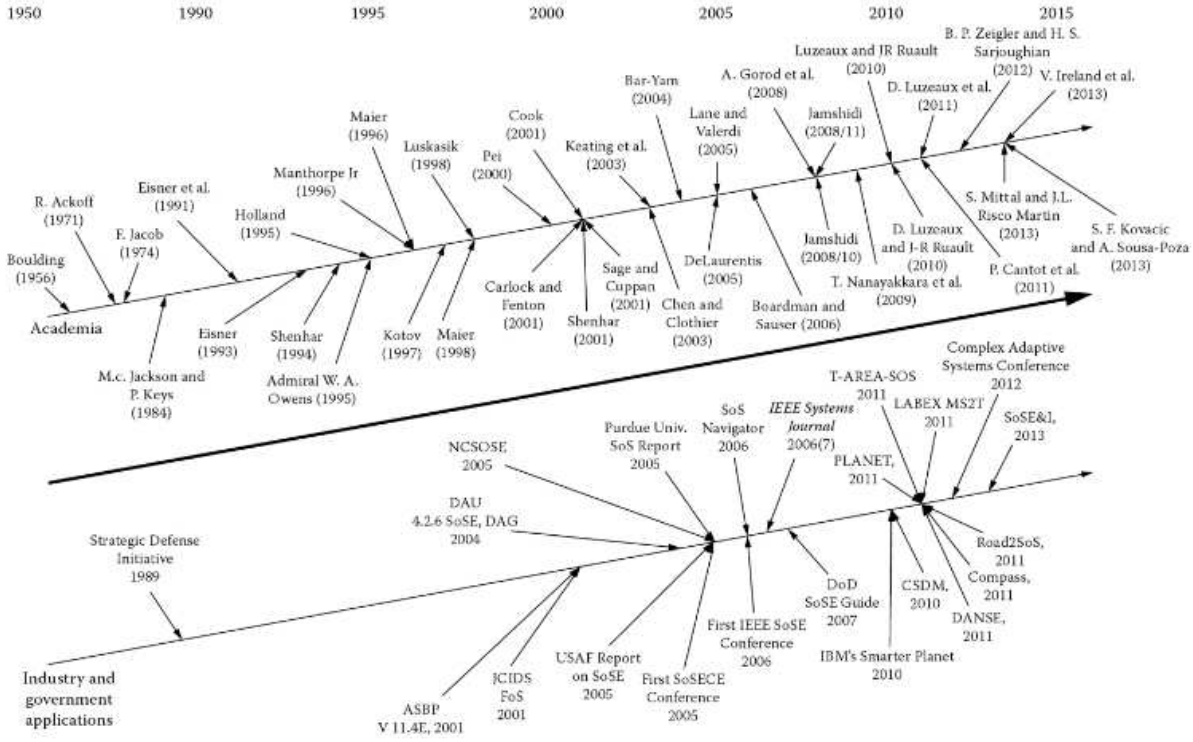


Figura 7 – Linha do tempo das principais contribuições em Sistemas de Sistemas. Extraído de Gorod *et al.* (2014)

nhar, que iniciaram o processo rumo uma concepção mais "moderna" de SoS (GOROD *et al.*, 2008). De maneira sumarizada, Eisner sistematizou um SoS como um conjunto de sistemas integrados, construídos deliberadamente a partir de um processo nominal de engenharia de sistemas. Este conjunto é composto por sistemas interdependentes combinados para operar em uma solução multifuncional visando alcançar um objetivo comum (EISNER *et al.*, 1991; EISNER *et al.*, 1993). Shenhar caracterizou um SoS como uma "matriz" de sistemas, definindo um SoS como uma grande coleção ou rede de sistemas trabalhando juntos para alcançar um objetivo comum (SHENHAR, 1994). Holland, em 1995, também abordou a questão dos SoS, sob diferentes perspectivas. Ele vislumbrou um SoS como um sistema adaptativo complexo artificial que, em mudanças contínuas, "se auto-organiza" com regras adaptativas, com a intenção de aumentar a sua complexidade (HOLLAND, 1995). No mesmo ano, Maier propôs uma abordagem de caracterização para diferenciar um sistema comum de um Sistema de Sistemas. Segundo Maier, um SoS deve incluir "interdependência operacional entre seus elementos, independência gerencial de cada um de seus elementos, um desenvolvimento evolutivo, comportamento emergente e distribuição geográfica". Atualmente, Maier é um dos principais pesquisadores no campo de SoS. Em 1998, Maier aperfeiçoou sua própria definição de SoS, enfatizando duas de suas propriedades fundamentais: Interdependência Operacional e Gerencial de seus componentes (MAIER, 1996; MAEIR, 1998). Kotov em 1997 e Lukasik em 1998, deram um

passo adicional, buscando desenvolver o projeto de um SoS. Curiosamente, Kotov foi o primeiro pesquisador a desenvolver um modelo formal de SoS, porque até 1997, os estudos envolvendo SoS eram mais conceituais do que formais (KOTOV, 1997; LUKASIK, 1998). Boardman e Sauser, em 2006, publicaram um artigo com uma análise comparativa envolvendo 40 diferentes visões sobre SoS. Através da comparação entre esses conceitos eles chegaram uma visão comum sobre SoS, percebendo que os principais elementos envolvendo a área de SoS podem ser sumarizados em cinco características principais: Autonomia, Pertencimento, Conectividade, Diversidade e Emergência (ou o ABCDE dos SoS, seguindo a terminologia em inglês: *Autonomy, Belonging, Conectivity, Diversity, Emergence*). A seguir, detalhamos o significado de cada uma dessas características (BOARDMAN; SAUSER, 2006; JAMSHIDI, 2011; GOROD *et al.*, 2014; GOROD *et al.*, 2008; DIMARIO, 2006).

- **Autonomia**: é a capacidade de um componente de um SoS tomar suas próprias decisões independentemente dos outros sistemas presentes no SoS. Assim, cada subsistema integrante do SoS é livre para tomar suas próprias decisões. No entanto, há algumas restrições quanto a essas decisões: não pode violar sua própria natureza e não pode violar o propósito do todo (do SoS). Se um subsistema viola o propósito do todo, pode ser excluído do SoS e, consequentemente, abrir espaço para outro sistema se integrar ao SoS em seu lugar;

- **Pertencimento**: é a capacidade que os componentes de um SoS têm de optar por pertencer ou não ao SoS, com base em suas próprias necessidades e finalidades. Em outras palavras, um componente escolhe pertencer ao SoS com base nos custos e benefícios de sua participação no mesmo.;

- **Conectividade**: é a capacidade que as partes de um SoS têm em manter conexões entre si, para integrar um SoS. Para garantir essa conectividade, os projetistas de um SoS precisam lidar com uma série de questões cruciais. Como garantir essa conectividade entre os componentes de um SoS, especialmente considerando-se a integração de sistemas legados (que não foram originalmente projetados para participar de um SoS) ? Como adicionar novos sub-sistemas a um SoS? Como fazer para excluir sub-sistemas, quando necessário? Como atingir uma interoperabilidade completa entre as partes de um SoS? Alguns pesquisadores defendem ser necessário definir e construir uma conectividade dinâmica, que cria interfaces e links para adicionar e/ou remover subsistemas, consoante o surgimento de novas necessidades;

- **Diversidade**: um SoS é implicitamente heterogêneo, ou seja, composto por subsistemas que diferem uns dos outros. Tendo componentes de diversos tipos, um SoS possui maiores chances de atingir seus objetivos de nível mais alto, uma vez que

diferentes capacidades e competências estarão disponíveis distribuídas em seus subsistemas. Uma estratégia muito comum é a formação de coalizações dinâmicas entre os subsistemas, que operando em conjunto conseguem atingir objetivos que não teriam condições de atingir, caso estivessem atuando somente de maneira isolada. Com maior diversidade de tipos de subsistemas, as chances de compor uma coalização de sucesso aumentam;

- **Emergência**: é a capacidade de se chegar a soluções originalmente imprevisíveis ou inesperadas, a partir de um processo desenvolvimentista ou evolutivo. Em sistemas tradicionais, a composição das partes de um sistema é feita intencionalmente, visando atingir um determinado objetivo. Em um SoS, o resultado dessa composição nem sempre pode ser previsto de maneira antecipada ou deliberada. Devido à diversidade de subsistemas, diferentes coalizões podem ser formadas, por tentativa e erro, até que uma determinada coalizão de sucesso seja formada, por meio de um processo evolutivo, onde a interoperação de comportamentos entre os subsistemas transforme-se em uma solução eficiente para a obtenção dos objetivos de mais alto nível do SoS.

Como pode-se depreender, essas características exercem influências umas sobre as outras, levando a uma interdependência entre características. A tabela 1 mostra essa interdependência entre as características.

Um SoS não pode ser classificado e definido apenas com base em medidas convencionais de engenharia de sistemas, pois um SoS, além de ter um propósito diferente, precisa considerar tanto a interação entre os subsistemas que o constituem como a interação entre esses subsistemas e seus usuários. Com isso, toda uma área de pesquisa em "Engenharia de Sistema de Sistemas" (SoSE, *Systems of Systems Engineering*, no original em inglês), se desenvolveu. Um SoS requer uma nova abordagem devido à sua dinâmica e ao ambiente dinâmico em que surge. Além disso, exige uma visão diferente em relação à arquitetura de software, já que a engenharia de sistemas convencional não é suficiente para sustentá-lo. É importante distinguir entre os conceitos de SoS e SoSE, considerando-se suas habilidades para fornecer diferentes serviços

É preciso, ainda, diferenciar entre um sistema complexo tradicional (ou seja, também constituído por subsistemas, mas construído de maneira tradicional, centralizada) e o que estamos chamando aqui de um SoS. Embora existam semelhanças entre estes, existem também diferenças cruciais para construir um SoS que os diferenciam de um sistema complexo tradicional, constituído por subsistemas (que podemos chamar aqui de Sistema de Subsistemas — SoSS para diferenciá-lo dos SoS). Por exemplo, na SoSE, ao contrário da engenharia de sistemas tradicional, há dificuldades em se ter um ambiente de testes, pois até o momento não existe um ambiente dinâmico consolidado com o qual sejam

Tabela 1 – Interdependência das características de um SoS. Adaptado de Sauser *et al.* (2008)

Característica	Determina o grau de...	Interdependente com...
Autonomia	Perspectivas dos sistemas são mantidas e permanecem independentes umas das outras	Diversidade
	Erros podem tornar-se correlatos uns com os outros	Emergência
	Portifólio de sistemas é um equilíbrio entre único e geral	Pertencimento
Pertencimento	Conformidade é forçada sobre sistemas necessários	Autonomia
	Responsabilidade é formalizada para pertencimento controlado	Diversidade
	Capacidade permite a distinção entre soluções boas e ruins; sobrevivência do mais bem-adaptado	Emergência
Conectividade	Auto-organização e descentralização	Pertencimento
	Autointeresse e independência	Autonomia
	"Quem", "o quê", "onde", "quando" e "como- permitindo por um lado a coordenação de atividades enquanto resolve diferentes problemas por outro	Diversidade
Diversidade	Preservação de independência é permissível	Autonomia
	Coletivismo se torna uma característica destrutiva (ex: "pensamento de grupo")	Pertencimento
	O legado de novos sistemas não é redundante em relação àqueles que já existem	Conectividade
Emergência	Saber e saber que se tem diferentes habilidades	Diversidade
	Tentar encontrar os sistemas ótimos irá te desviar do caminho (e tentar encontrar os verdadeiros sistemas não irá)	Conectividade

possíveis testes de SoS. Dessa forma, o ambiente de testes e simulações para um SoS acaba sendo o próprio mundo real (BOARDMAN; SAUSER, 2006; DIMARIO, 2006; COLE, 2006; JAMSHIDI, 2011; GOROD *et al.*, 2008). A Figura 8 mostra um comparativo entre um SoSS e um SoS com base nos 5 princípios que definem um SoS. É possível imaginar que alguns destes princípios podem também estar presentes em um SoSS. Dependendo dessas características estarem mais ou menos presentes, podemos classificar o sistema objeto de nossa análise como SoSS ou SoS.

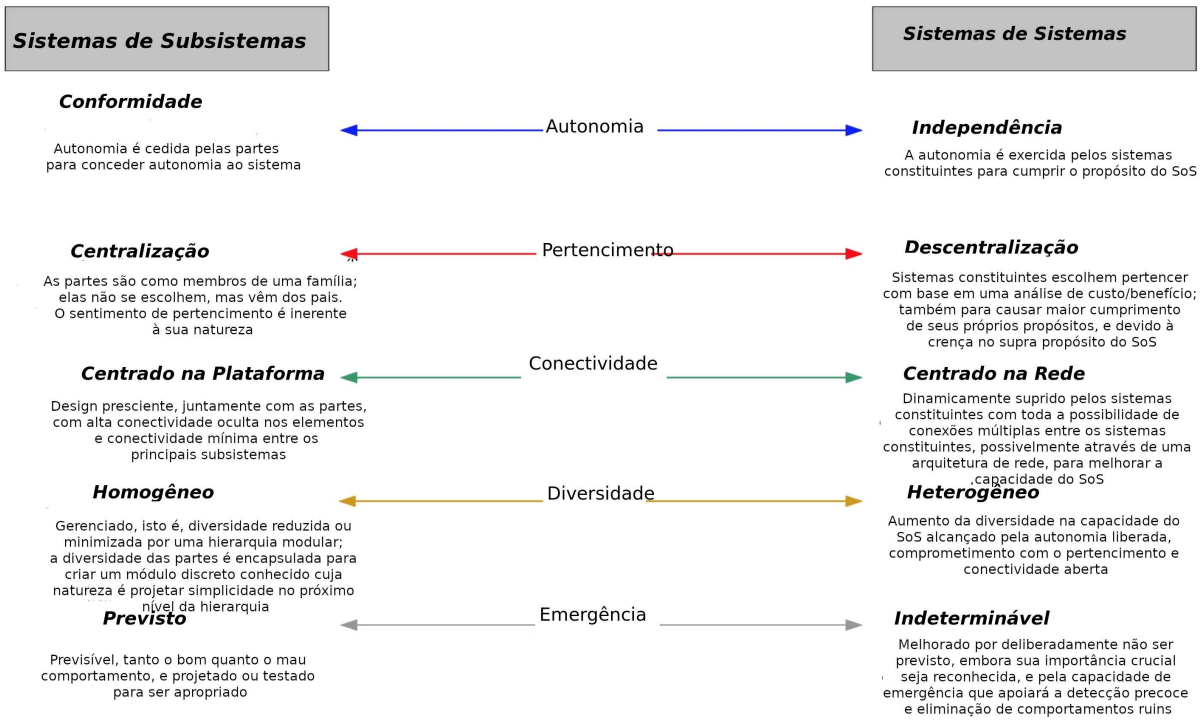


Figura 8 – Diferenças das características de sistemas tradicionais e Sistemas de Sistemas. Adaptado de Gorod *et al.* (2008)

A Figura 8 mostra como SoSS e SoS diferem, considerando cada uma das características levantadas, conforme apresentado por Gorod *et al.* (2008). Um fator importante é a interdependência que ocorre em um SoS, que dá a cada um de seus componentes a capacidade de decidir sobre sua aderência ou não à finalidade essencial de um SoS, condição necessária para que um SoS possa se auto-organizar. Na prática, a presença dos cinco princípios de um SoS pode acontecer com diferentes intensidades, porém de fato para que um sistema possa ser classificado e utilizado como SoS em ambiente dinâmico, este deve, de alguma forma, obedecer à maioria desses princípios. Em outras palavras, não é porque um SoS tem baixa diversidade (ou baixa autonomia, ou baixo pertencimento, por exemplo), que ele deixará de ser SoS, mas em algum grau, esses princípios precisariam estar presentes (GOROD *et al.*, 2008; JAMSHIDI, 2011).

2.3.1 Exemplos de Sistemas de Sistemas

Na sequência, apresentamos alguns exemplos de implementações de Sistemas de Sistemas, relatados na literatura. Essas implementações são baseadas em problemas cotidianos nas áreas de saúde, transporte e sistemas aeronáuticos. Esses exemplos são bons para ilustrar situações concretas onde SoS podem ser interessantes.

- **Sistemas de saúde — Rede Nacional de Informações sobre Saúde (NHIN)**

  Buscando resolver problemas no sistema de saúde americano, o presidente dos EUA, em 2004, ordenou ao seu Secretário de Saúde que iniciasse o desenvolvimento do programa *National Healthcare Information Network* (NHIN), visando criar um sistema nacional de informações em saúde para toda a população até 2014. A primeira das especificações de projeto do NHIN foi finalizada em outubro de 2006 e disponibilizada e revisada pela *America Healthcare Information Community* (AHIC), tendo sido aceita pelo Secretário de Saúde em 2007. Hoje, o NHIN está em evolução constante e fornece em tempo real informações heterogêneas de hospitais, departamentos envolvidos e dados médicos de pacientes. Tem a capacidade de apresentar um Registro Eletrônico de Saúde (*Electronic Health Records*, EHR) a qualquer médico ou hospital para consulta de pacientes. A NHIN conta com uma rede de Organizações Regionais de Informação de Saúde (*Regional Healthcare Information Organizations*, RHIOs) independentes que realizam a comunicação de dados de dezenas de sistemas de informação médica legados, usados em departamentos hospitalares, consultórios médicos e locais de telemedicina em metaformatos especificados pela NHIN de forma confiável, que podem ser usado por qualquer país. É perceptível que o NHIN é uma "rede de redes" que é claramente um complexo SoS que troca informações de saúde entre ele e os RHIOs que ajudam a população dos EUA com segurança e eficiência. Um ambiente de simulação, modelagem e outras ferramentas foram usadas para garantir confiabilidade, planejamento de custo/benefício, configuração, implantação e gerenciamento de sistemas e subsistemas heterogêneos dos RHIOs (SLOANE, 2006; SLOANE *et al.*, 2007; JAMSHIDI, 2011);

- **Sistemas de Transporte — Sistemas de Transporte Nacionais (NTS)**

  Os Sistemas de Transporte Nacionais correspondem a uma coleção de redes composta por sistemas heterogêneos, dentre os quais o Sistema de Transporte Aéreo (*Air Transportation System*, ATS) e seus Sistemas de Espaço Aéreo Nacionais (*National Airspace Systems*, NAS) são um componente. A configuração corrente do NTS não pode ser considerada um SoS, uma vez que seus subsistemas não estão completamente integrados, e sua dimensão global (incluindo-se seus possíveis impactos multimodais e influências políticas, sociais e empresariais) ainda não foi devidamente considerada. DeLaurentis (2005) propôs que uma abordagem de SoS,

aplicada ao NTS, poderia levar a melhores resultados nos contextos tecnológico, sócio-econômico, operacional e político, transformando a arquitetura ATS corrente em um futuro sistema mais eficiente, considerando a complexidade de um sistema desse porte, bem como sua alta demanda, além das diversas incertezas que envolvem um sistema desse tipo. Um sistema de transporte que integrasse não somente o tráfego aéreo, como outros tipos de veículos, como trens, automóveis, caminhões e ônibus, impactaria diretamente todos os setores da sociedade, tanto públicos quanto privados. Keating e Jamshidi em 2008, propuseram pensamentos semelhantes a esse respeito, onde características emergentes poderiam reduzir significativamente o trânsito presente nas ruas e túneis para preservar a vida de usuários, evitando acidentes. Um sistema integrado, monitorando e controlando diferentes tipos de transporte, poderia ser visto como um SoS (LAURENTIS *et al.*, 2004; KEATING, 2008; MORLEY, 2006; JAMSHIDI, 2008; JAMSHIDI, 2011);

- **Sistemas Aeronáuticos — Projeto e-Enabled da Boeing**

  Um bom exemplo de como um SoS pode surgir a partir da integração de diversos sistemas legados que começam a cooperar entre si é o caso do Projeto e-Enabled da Boeing. Este projeto teve como objetivo desenvolver uma estratégia empresarial e uma arquitetura técnica para tornar os aviões da Boeing mais "*network aware*", e, portanto, mais capazes de atender a uma demanda de integração em rede que se mostrava uma tendência nas tecnologias de rede e computação. Logo após o lançamento do projeto, ficou aparente que o projeto deveria ser maior do que simplesmente o avião, e deveria incorporar uma perspectiva que envolvesse toda a companhia aérea. O projeto cresceu para incluir também diversos componentes arquiteturais em terra, relacionados a companhias aéreas, as fábricas da Boeing, além de outros pontos-chave tais como aeroportos, fornecedores e provedores de Internet. A Figura 9 mostra a interação entre os elementos que fazem parte deste SoS.

  De acordo com Wilber, o projeto e-Enabled foi um fator importante para definir uma solução do tipo SoS para o problema de interoperabilidade e comunicação entre vários componentes existentes que compõem os sistemas operacionais das companhias aéreas e de aviação, incluindo operações de voo e manutenção. O objetivo do projeto era encontrar maneiras de alavancar as operações centradas na rede para reduzir os custos de produção, operações e manutenção para os clientes da Boeing e das companhias aéreas (WILBER, 2008; JAMSHIDI, 2011).

  A implantação da arquitetura foi pensada para contribuir principalmente com as aeronaves e demais transportes. No entanto, ele é projetado de forma que os recursos e sistemas possam ser reutilizados e compartilhados em vários outros componentes do sistema. O avião 787, equipado com a capacidade e-Enabled, foi o elemento mais importante para a implementação desta arquitetura. Ele permitiu elementos

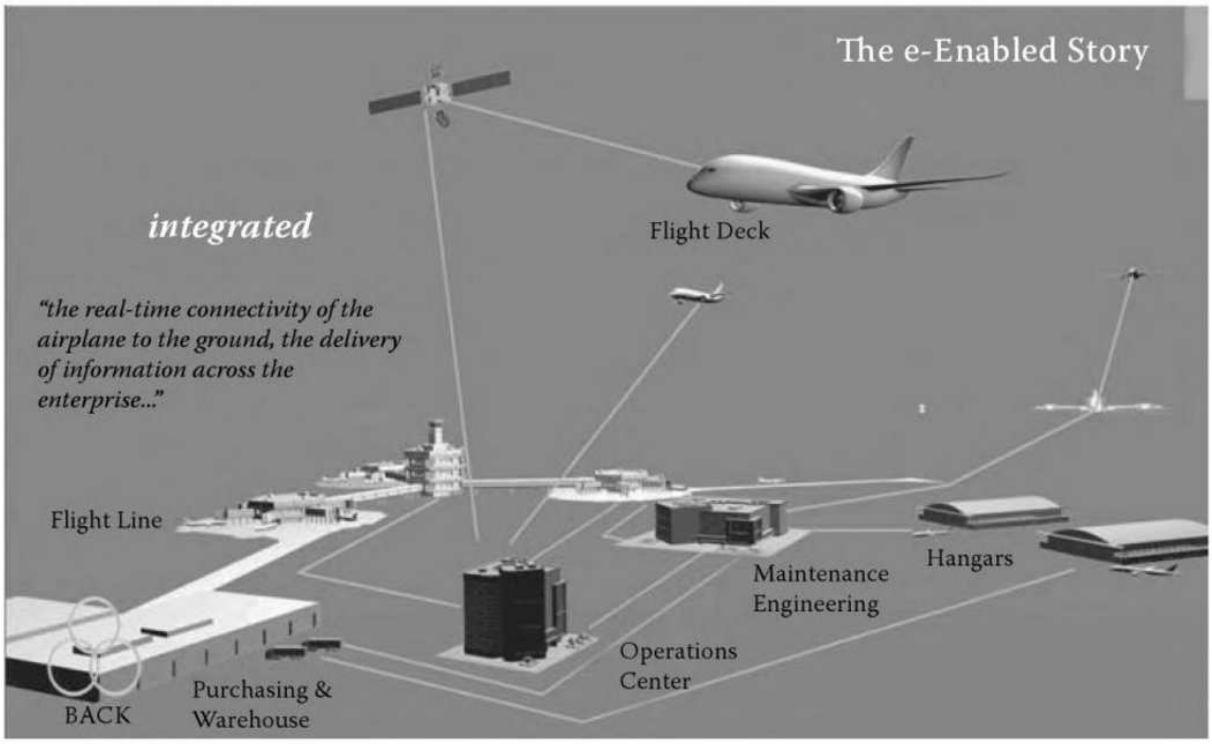


Figura 9 – SoS "E-enabling" para o Boeing 787. Extraído de Jamshidi (2011)

on-board e off-board de rede que garantiram eficiência, flexibilidade e segurança (WILBER, 2008; JAMSHIDI, 2011).

2.3.2 Resiliência

O conceito de resiliência em Sistemas de Sistemas (SoS) é fundamental para garantir a capacidade desses sistemas de se adaptarem a mudanças e incertezas. A resiliência pode ser definido como a capacidade de um sistema de sobreviver e se recuperar dessas mudanças (UDAY; MARAIS, 2014), mantendo seu desempenho face a pertubações. Uma vez que ocorre uma falha, a resiliência é a capacidade inerente de um sistema de sobreviver e se recuperar dessa perturbação. E assim, a resiliência é representada como uma combinação de capacidade de sobrevivência e capacidade de recuperação, conforme mostrado na Figura 10.

Além disso, é importante destacar que a resiliência em SoS não é uma característica estática, mas sim um processo contínuo de adaptação e evolução. Isso significa que os sistemas devem ser capazes de se adaptar a mudanças e perturbações ao longo do tempo.

Por fim, é importante destacar que a resiliência em SoS não é uma característica exclusiva de sistemas tecnológicos, mas também se aplica a sistemas sociais e econômicos, por exemplo.

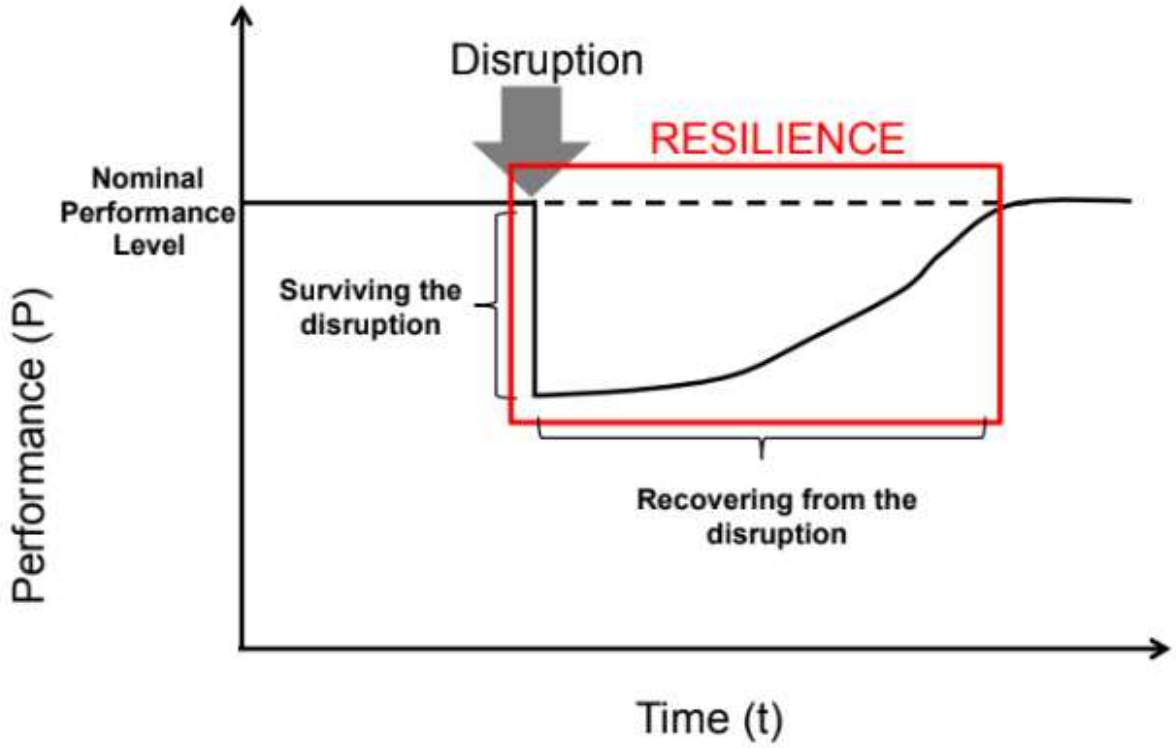


Figura 10 – Ilustração do conceito de Resiliência. Extraído de Uday e Marais (2014)

2.4 Sistemas Ciber-Físicos

Ao longo da história da computação, os sistemas embarcados têm evoluído de diversas maneiras, até chegar aos dias de hoje, quando existem disponíveis diferentes tipos de dispositivos e sistemas que podem se enquadrar sob esta denominação, com um vasto espectro de recursos com diferente poder computacional. Ao longo dos anos 90, parte da comunidade de pesquisa em sistemas embarcados abordou o assim chamado "problema do software embarcado" de forma ligeiramente diferente do tradicional, permitindo o surgimento de uma nova área de tecnologia, cuja ênfase se encontra na integração de computadores, processos físicos e redes de computadores: os Sistemas Ciber-Físicos (CPS, do original em inglês *Cyber-Physical Systems*). De acordo com Lee e Seshia (2011), CPS refere-se a computadores embarcados, comunicando-se por meio de redes e controlando processos físicos, geralmente tendo *loops* de *feedback* onde os processos físicos afetam os cálculos e vice-versa.

O termo "sistemas ciber-físicos" foi criado por Helen Gill da National Science Foundation, nos Estados Unidos em 2006. É comum imaginar que o termo "ciberespaço" está de alguma forma associado ao CPS, mas as raízes do termo CPS são mais antigas e profundas. Tanto os conceitos de "ciberespaço" quanto "sistemas ciber-físicos" são derivados da *cibernética*, e não um derivado do outro (LEE; SESHIA, 2011).

A *cibernética* é uma abordagem transdisciplinar, definida por Norbert Wiener em 1948, para explorar os sistemas regulatórios, suas estruturas, restrições e possibilidades ou, em outras palavras, o estudo dos efeitos do *feedback* na teoria de sistemas, para implementar algum tipo de controle sobre suas variáveis. A palavra cibernética vem do grego *"kybernetike"*, que significa "governança", ou seja, controle e, principalmente, controle por meio de feedback. Mais tarde, o prefixo "ciber" passou a ser utilizado para designar tudo que pudesse se relacionar com as grandes redes integradas de computadores, particularmente, a Internet. O prefixo ciber em sistemas ciber-físicos incorpora ambas as semânticas. Refere-se a: dispositivos físicos conectados à Internet, e que podem ser monitorados e controlados por meio de algum tipo de controlador.

Existem muitas definições de CPS na literatura, mas a definição mais comum entre os colaboradores é que CPS tem a ver com a exposição de partes do ambiente urbano à Internet, onde de alguma forma, sistemas computacionais usam informações coletadas do ambiente por dispositivos sensoriais para desenvolver o controle dos ativos deste ambiente, tendo em vista um certo número de objetivos a serem alcançados e que alguns índices de desempenho devem ser otimizados. Além disso, os CPS são definidos como sistemas integrados que fornecem computação, rede e processos físicos. Outra definição pode ser: um sistema onde o ambiente físico e os componentes de software estão intimamente relacionados, cada um deles operando em diferentes escalas espaciais e temporais, realizando diferentes modalidades de comportamento e interagindo entre si (CONTI *et al.*, 2012; SHA *et al.*, 2009; HORVATH; GERRITSEN, 2012; LEE, 2009; NSF, 2013; KHAITAN; MCCALLEY, 2015).

De acordo com Miclea e Sanislav (2011) e Khaitan e McCalley (2015), os CPS apresentam um conjunto de características fundamentais que são listadas a seguir:

· **Mecanismo de Feedback:** Seus componentes físicos possuem uma capacidade *ciber*, ou seja, são controlados por meio de feedback;

· **Automação:** Loops de controle são fechados utilizando altos níveis de automação;

· **Escalabilidade:** A rede é utilizada, em múltiplas escalas, para integrar seus componentes;

· **Integrabilidade:** Essa integração ocorre em múltiplas escalas temporais e espaciais;

· **Reconfigurável:** O sistema pode ser organizado dinamicamente e reconfigurável em tempo real;

Na área de CPS, existem vários desafios no desenvolvimento de projetos que aproveitem a interação entre recursos cibernéticos e físicos. Os eventos do mundo real precisam

refletir no mundo cibernético e, consequentemente, o mundo cibernético precisa se comunicar com o mundo físico, de modo que as decisões possam ser passadas aos atuadores. Além disso, essa comunicação e integração devem ser realizadas em tempo real e com precisão. Com isso, um CPS precisa coordenar diferentes sistemas (dispositivos computacionais, sensores e atuadores distribuídos) para atuar. Neste caso, sensores e atuadores funcionam como pontes entre os mundos físico e cibernético, adaptados para funcionar sob a variação temporal do contexto físico e cibernético (KHAITAN; MCCALLEY, 2015).

Em meados de 2014, o NIST estabeleceu o CPS Public Working Group (CPS PWG) que visava reunir especialistas para conjecturar sobre os principais aspectos do CPS. Isso reflete o enorme interesse pela área em uma perspectiva de longo prazo. Além disso, na Europa, há uma forte preocupação em aumentar continuamente a conscientização sobre sistemas ciber-físicos, uma vez que essa área engloba uma das tecnologias mais promissoras da atualidade, que vem atraindo largo financiamento no mercado europeu, ultimamente (HÅKANSSON *et al.*, 2015).

O grafo conceitual representado na Figura 11 apresenta uma visão geral da ontologia dos Sistemas Ciber-Físicos (WANG *et al.*, 2015). Esta ilustração foi criada pelo professor Edward A. Lee da UC Berkeley, que sumarizou a taxonomia CPS de S. Shyam Sunder do NIST (*Instituto Nacional de Padrões e Tecnologia*) em uma imagem.

Os sistemas ciber-físicos são inerentemente heterogêneos, pois combinam dinâmica física com processos computacionais. Essa heterogeneidade existe mesmo nos domínios físico e cibernético. O domínio físico pode ser multi-físico, combinando, por exemplo, processos químicos e biológicos, controle de movimento mecânico e operadores humanos. O domínio cibernético pode combinar linguagens de programação, modelos de componentes de software, tecnologias de rede e mecanismos de concorrência. Os sistemas ciber-físicos são intrinsecamente simultâneos. Os subsistemas cibernético e físico coexistem no tempo, mas mesmo dentro desses subsistemas existem processos simultâneos. Modelos de concorrência no mundo físico são muito diferentes de modelos de concorrência em software (por exemplo, intercalação arbitrária de sequências de ações atômicas) e também muito diferentes de modelos de concorrência em redes (por exemplo, ações discretas, assíncronas, parcialmente ordenadas ou com intervalos de clock com tempos controlados) (LEE, 2006). Conciliar esses modelos divergentes de simultaneidade, garantindo a interoperabilidade entre componentes com diferentes modelos de simultaneidade é uma das questões centrais a serem tratadas no CPS.

2.4.1 Aplicações de CPS

Aplicações de sistemas ciber-físicos foram desenvolvidas em vários domínios. A organização da informação, horizontalmente dentro de cada domínio e verticalmente, em

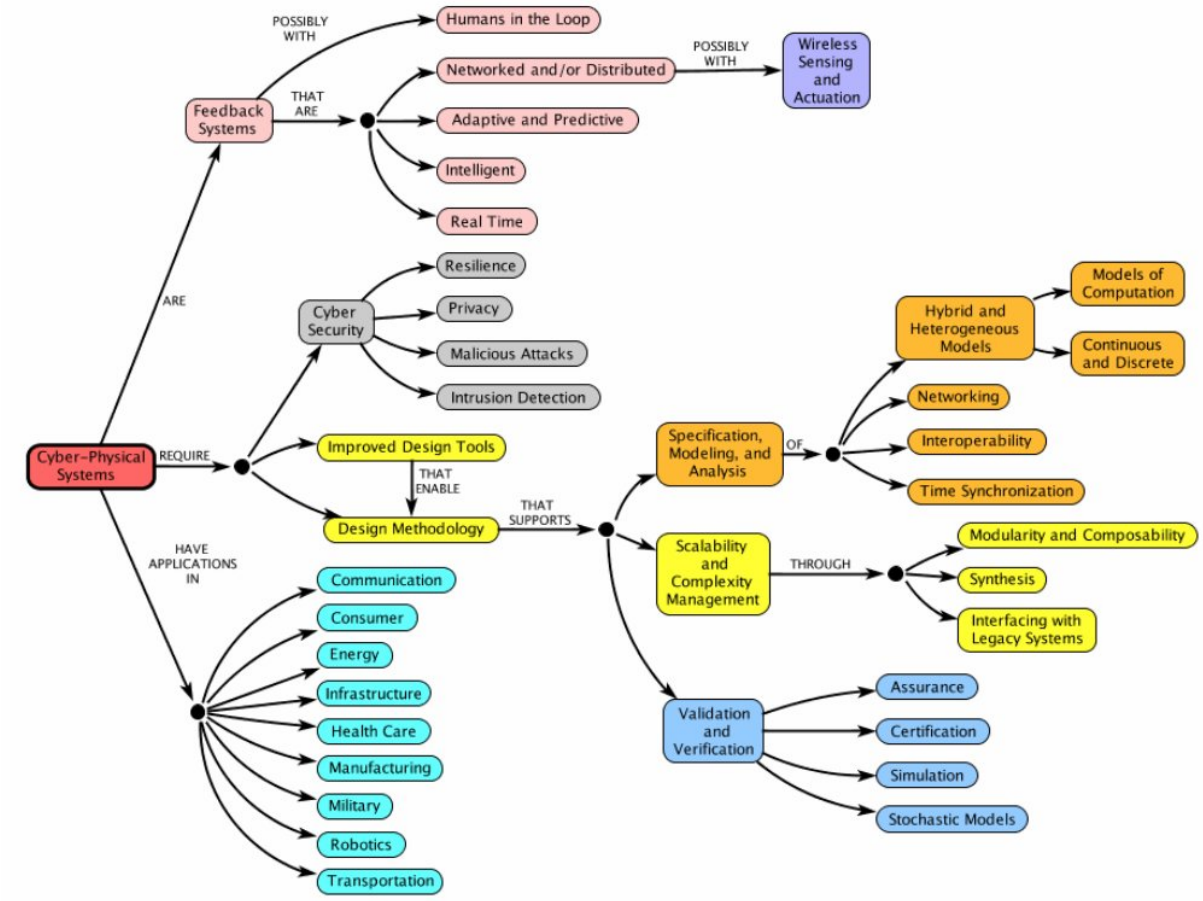


Figura 11 – Grafo conceitual de Sistemas Ciber-Físicos. Extraída de Wang *et al.* (2015)

vários subsistemas, é fundamental para o desenvolvimento de aplicações reutilizáveis e eficientes (BALAJI *et al.*, 2015). A Tabela 2 fornece uma visão geral envolvendo diversos domínios de aplicativos CPS de acordo com seu nível de escala e funcionalidade (GUNES *et al.*, 2014). Nas próximas subseções, detalhamos os seguintes domínios de aplicação, para exemplificar aplicações de CPS: Manufatura Inteligente, Resposta a Situações de Emergência, Transporte Aéreo Não-Tripulado, Infraestrutura Crítica, Saúde e Medicina, Transporte Inteligente e Robótica de Serviços.

- **Manufatura Inteligente**

  Manufatura inteligente refere-se ao uso de tecnologias de software e hardware embarcadas para otimizar a produtividade na indústria ou na prestação de serviços. É um dos principais domínios de aplicação de CPS, devido a fatores como produção em massa, marketing nacional e internacional, crescimento econômico, etc (GUNES *et al.*, 2014). Ao longo dos anos, grandes esforços foram feitos nos Estados Unidos e na Europa para caracterizar o CPS para manufatura inteligente, uma vez que a fabricação enfrenta muitas demandas por alta flexibilidade. Além disso, essas demandas têm sido difíceis de atender ultimamente devido a razões de segurança. Essas razões surgem das interações e cooperações íntimas entre máquinas e humanos na ausência

Tabela 2 – Funcionalidades dos domínios de Sistemas Ciber-Físicos. Adaptado de Gunes *et al.* (2014)

Tipo de Domínio	Escala/Funcionalidade
Manufatura Inteligente	Média Escala, otimizando a produtividade na manufatura de bens ou entrega de serviços;
Resposta a Emergências	Média/Larga Escala; lidando com ameaças à segurança pública e protegendo a natureza e infraestruturas valiosas;
Transporte Aéreo	Larga Escala; operação e gerenciamento de tráfego de sistemas de aeronaves;
Infraestrutura Crítica	Larga Escala; distribuição de suprimentos cotidianos como água, eletricidade, gás e petróleo;
Saúde e Medicina	Média Escala, monitorando as condições de saúde dos pacientes e tomando as ações necessárias;
Transporte Inteligente	Média/Larga Escala, melhorando a segurança, coordenação e serviços de gerenciamento de tráfego em tempo real com compartilhamento de informações;
Robôs para Serviço	Pequena/Média Escala, executando serviços para o bem-estar dos humanos;

de sensores e dispositivos inteligentes suficientes para evitar perigos (GUNES *et al.*, 2014). Segundo MacDougall (2014), nos próximos anos, o CPS melhorará a produtividade e a eficiência dos recursos, a fim de permitir modelos mais flexíveis nos processos de fabricação. A produção de CPS é realmente complexa, mas inovações de sistemas auxiliares, como realidade aumentada e interação multimodal, ajudarão os trabalhadores da fábrica a lidar com essas complexidades;

- **Resposta a Situações de Emergência**

  A resposta a situações de emergência diz respeito a sistemas que lidam com ameaças contra a segurança e o bem-estar público, a fim de proteger a natureza e/ou infraestruturas valiosas. Caracteriza-se por uma forte colaboração entre profissionais de gestão de emergências, autoridades e moradores das comunidades. O uso de mecanismos eficazes de alerta, resposta e recuperação são necessários em futuros sistemas de gerenciamento de emergência. Nesse sentido, os sistemas ciber-físicos podem fornecer uma resposta rápida a emergências, por meio de um grande número de nós sensores, em regiões com casos de desastres naturais ou provocados pelo homem (GUNES *et al.*, 2014). De acordo com Stankovic *et al.* (2005), esta resposta rápida requer que os nós avaliem coletivamente a situação e informem rapidamente a autoridade central, mesmo em ambientes dinâmicos. Portanto, robustez, utilização efetiva de recursos, capacidade de adaptação e pontualidade entram em jogo na resposta a emergências;

- **Transporte Aéreo Não Tripulado**

  Transporte aéreo não tripulado refere-se a sistemas de aviação civis ou militares, tele-operados ou autônomos e seu gerenciamento de tráfego. Espera-se que os veículos aéreos inteligentes se tornem cada vez mais populares, não apenas em serviços militares, mas também em atividades civis, como vigilância, pulverização agrícola, fotografia, arqueologia e muitos outros campos (NILSSON, 2013; CAVOUKIAN, 2012). Um veículo aéreo não-tripulado, mais conhecido como *drone*, é um exemplo de veículo aéreo inteligente que se tornou muito popular ultimamente. Espera-se que a densidade desse tipo de tráfego aéreo aumente, exigindo um conjunto hierárquico de ferramentas de gerenciamento em rede para maximizar a eficiência, mantendo a segurança no solo e no ar. Além disso, questões como consciência física, robustez e exploração cooperativa da superfície planetária compartilham uma série de limitações importantes e desafios de pesquisa (ATKINS, 2006). Neste sentido, prevê-se que os CPS tenham um impacto profundo na aviação e na Gestão do Tráfego Aéreo do futuro (GUNES *et al.*, 2014);

- **Infraestrutura**

  Infraestrutura refere-se a propriedades valiosas do setor público necessárias para a sobrevivência ou o bem-estar das nações. Uma das aplicações mais promissoras no domínio da infraestrutura são os *Smart Grids* (termo em inglês para Redes Inteligentes), um tipo de rede de transmissão de energia na qual o monitoramento e atuação inteligente sobre a rede permite um maior controle. Estes incorporam usinas industriais de armazenamento de energia, instalações de transmissão, recursos de energia renovável e distribuição de energia, além de instalações de gerenciamento em casas/edifícios inteligentes (GUNES *et al.*, 2014). Um *Smart Grid* fornece monitoramento, distribuição e planejamento de carga em tempo real no nível da concessionária; um equilíbrio de oferta e demanda no nível do dispositivo; integração dos recursos energéticos existentes na rede; consciência da grade em grande escala e capacidade de alternar entre níveis altos e baixos. De acordo com Mo *et al.* (2012), um desafio crucial a ser enfrentado trata das demandas de segurança do CPS em relação à continuidade do fornecimento de energia e precisão da precificação dinâmica;

- **Saúde e Medicina**

  Cuidados de saúde e medicina referem-se às questões que abordam múltiplos aspectos da fisiologia do paciente. Atenção especial é dedicada a aplicações médicas em pesquisa CPS, como oportunidades em assistência remota de atendimento domiciliar, vida assistida, sala de cirurgia inteligente, dispositivos médicos inteligentes e assim por diante. Os desafios tecnológicos atuais no projeto do Sistema Médico Ciberfísico abrangem o desenvolvimento confiável baseado em software para fornecer

novas funcionalidades, maior conectividade de dispositivos médicos equipados com interfaces de rede e demanda por monitoramento contínuo de pacientes (GUNES *et al.*, 2014);

- **Transporte Inteligente**

  Transporte inteligente refere-se a detecção, comunicação, cálculo e tecnologias de controle em sistemas de transporte. Visa melhorar a segurança, a coordenação e os serviços na gestão do tráfego com o compartilhamento de informação em tempo real (GUNES *et al.*, 2014). Um dos principais desafios no projeto e operação de ITS é a troca de informações, sendo uma tarefa difícil em sistemas altamente distribuídos. Do ponto de vista técnico, existem dificuldades em integrar informações usando padrões compatíveis e conectar vários sistemas (um problema geralmente conhecido como Problema da Interoperabilidade Semântica (HEILER, 1995; WONG *et al.*, 2005; VETERE; LENZERINI, 2005), especialmente devido à complexidade e abundância dos fluxos de informações, além da heterogeneidade de hardware e dados envolvidos no contexto das TIs (GREGOR *et al.*, 2016). Outro desafio está no âmbito da área de engenharia chamada de Pesquisa Operacional: como rotear e gerenciar o tráfego de pessoas que se deslocam em um cenário urbano, minimizando o tempo e os custos de viagem. Graças à integração do CPS em infraestruturas, veículos e estradas, os sistemas de transporte inteligentes podem obter assistência ao motorista, prevenção ou notificação de colisões, melhorias no tempo de viagem, reduções no congestionamento e controle avançado sobre infraestrutura e veículos para economia de energia;

- **Robótica de Serviços**

  Robótica de serviços refere-se a sistemas onde robôs inteligentes são usados para realizar serviços para o bem-estar dos seres humanos. Os robôs podem ser totalmente autônomos, semi-autônomos ou controlados remotamente por operadores humanos. Os domínios de aplicação mais promissores são: defesa (por exemplo, eliminação de explosivos, vigilância em áreas proibidas, etc.), monitoramento e controle ambiental, vida assistida, logística e outros (GUNES *et al.*, 2014). Do ponto de vista do CPS, a integração de humanos e robôs inteligentes possibilitará uma melhor cooperação, colaboração e organização para superar tarefas complexas.

No que diz respeito à modelagem e projeto de CPSs existem algumas questões em aberto, como, por exemplo, a necessidade de um framework unificador para modelar comportamentos computacionais e que permita fácil interface e consistência com o ambiente físico. Essa estrutura deve incluir artefatos de software capazes de permear as várias camadas do CPS e disparar eventos em tempo real. Assim, sua topologia deve ser

controlada dinamicamente em tempo real. Além disso, Khaitan e McCalley (2015) apontam a necessidade de conjuntos de dados do mundo real para testar e validar as ideias de pesquisa propostas. Além disso, Lee (2009) enfatiza a necessidade de novas linguagens de programação que consigam modelar a natureza dinâmica e assíncrona do CPS, em diferentes escalas temporais e espaciais. Os CPSs devem usar abstrações apropriadas que capturem informações temporais e espaciais para modelar o mundo real de maneira intuitiva, considerando segurança, restrições de energia, recursos e robustez (SHA *et al.*, 2009). Além disso, modelos futuros devem ser compostos por métodos que verifiquem e validem tanto componentes de software quanto hardware e todo o sistema, para garantir alto grau de dependência entre eles e controlar topologias reconfiguráveis operando em tempo real. Khaitan e McCalley (2015) também enfatizam a necessidade de computação de alto desempenho e algoritmos sofisticados para executar simulações e modelos em grande escala.

2.5 Gerenciamento Cognitivo

O conceito de Gerenciamento Cognitivo — e o próprio termo — surgiu nos últimos anos a partir dos conceitos previamente estabelecidos de Rádio Cognitivo (MITOLA, 2000) e Redes Cognitivas (THOMAS *et al.*, 2005). O conceito de *Rádio Cognitivo* surgiu quando o gerenciamento do espectro de rádio para transmissão e recepção de informações passou a ser monitorado e utilizado, dependendo de uma verificação de seu uso efetivo ou não por diferentes transmissores. Ao invés de se reservar previamente certas faixas de frequência, que poderiam ter seu uso pouco utilizado por diferentes transmissores, surgiu a ideia de que as faixas de frequência poderiam ser monitoradas e alocadas, dependendo do uso efetivo dessas faixas por diferentes usuários. O conceito de Redes Cognitivas, abstraiu essa questão do processo de cognição do uso das faixas de frequência, para o uso de redes de comunicações em geral. A ideia era monitorar o uso da rede, e fazer a alocação de recursos de maneira dinâmica, dependendo do uso efetivo dos recursos à disposição, utilizando o processo da cognição para compreender melhor, e obter conhecimento, sobre como estava sendo feito o uso efetivo dos recursos (o espectro de frequência, no caso do Rádio Cognitivo, e dos dispositivos conectados em rede, no caso das Redes Cognitivas). Esses conceitos desempenharam um papel importante nas ideias de Internet das Coisas e das Cidades Inteligentes, sendo crucial para garantir Qualidade de Serviço, Confiabilidade e bom gerenciamento de recursos em um cenário em que uma enorme quantidade de dispositivos deve estar totalmente conectada, principalmente através da Internet.

Generalizando os conceitos de Rádio Cognitiva e Redes Cognitivas para diferentes dispositivos interconectados na Internet das Coisas, desenvolveu-se uma visão mais abstrata dessas ideias no conceito de Gerenciamento Cognitivo, onde se antevê o uso de

diferentes objetos, conectados entre si em uma situação de Internet das Coisas, e onde seria desejável que esses recursos colocados à disposição, e que poderiam interagir entre si formando coalizões de objetos cooperando entre si, pudessem fazer emergir Sistemas de Sistemas. No Gerenciamento Cognitivo, tais objetos podem se conhecer mutuamente, e explorar os serviços que estão sendo disponibilizados, de forma que diferentes sistemas distribuídos possam formar coalizões e operar de maneira completamente autônoma, utilizando processamento cognitivo para essa finalidade. Dessa forma, atuando como uma maneira de conectar usuários e aplicativos a serviços ou Objetos do Mundo Real (RWO, do inglês *Real World Object*), ao mesmo tempo em que oculta de níveis superiores as características específicas dos componentes, o Gerenciamento Cognitivo torna-se uma ferramenta de fundamental importância para a implementação real da IoT em aplicações de cidade inteligente.

Embora ainda não exista uma versão definitiva para o conceito de Gerente Cognitivo, algumas estruturas são propostas na literatura e podem ser usadas para entender melhor os conceitos principais e o estado da arte sobre Gerenciamento Cognitivo.

A principal proposta para a estrutura de um Gerente Cognitivo foi introduzida por Kelaidonis *et al.* (2012), que propõe um modelo hierárquico com múltiplos níveis de abstração, e estuda o problema desde os níveis mais próximos dos dispositivos e objetos do mundo real (RWO, do inglês *Real World Object*), passando por diferentes níveis de serviço através de composições de representações virtuais desses RWOs, bem como os processos internos para construí-los. Os processos fundamentais desse quadro são a criação dinâmica de Objetos Virtuais (VO, do inglês *Virtual Object*) e sua composição em estruturas mais complexas nos assim chamados Objetos Virtuais Compostos (CVO, do inglês *Composite Virtual Object*). Além disso, o modelo de Kelaidonis *et al.* (2012) prevê a instanciação de CVOs baseada no conhecimento e a autorregeneração de CVOs. As Figuras 12 e 13 ilustram respectivamente a estrutura de Gerenciamento Cognitivo proposta por Kelaidonis *et al.* (2012) e uma visão de alto nível de uma possível aplicação desse *framework*, aplicado no contexto de cidades inteligentes.

Os principais elementos do modelo multi-nível de Kelaidonis *et al.* (2012) são melhor detalhados a seguir:

- **Nível de VO**

  Neste nível existem representações virtuais de objetos do mundo real. Os autores consideram dois tipos básicos de RWOs: habilitadas por tecnologias de informação e comunicação (TICs, como sensores e atuadores) e não habilitadas por TICs. Nesse contexto, os VOs se conectam aos RWOs e atuam como um “plug and play”, fornecendo uma interface (genérica) de alto nível para objetos, abstraindo a complexidade da infraestrutura IoT subjacente (em termos de tecnologia para interconexão). As

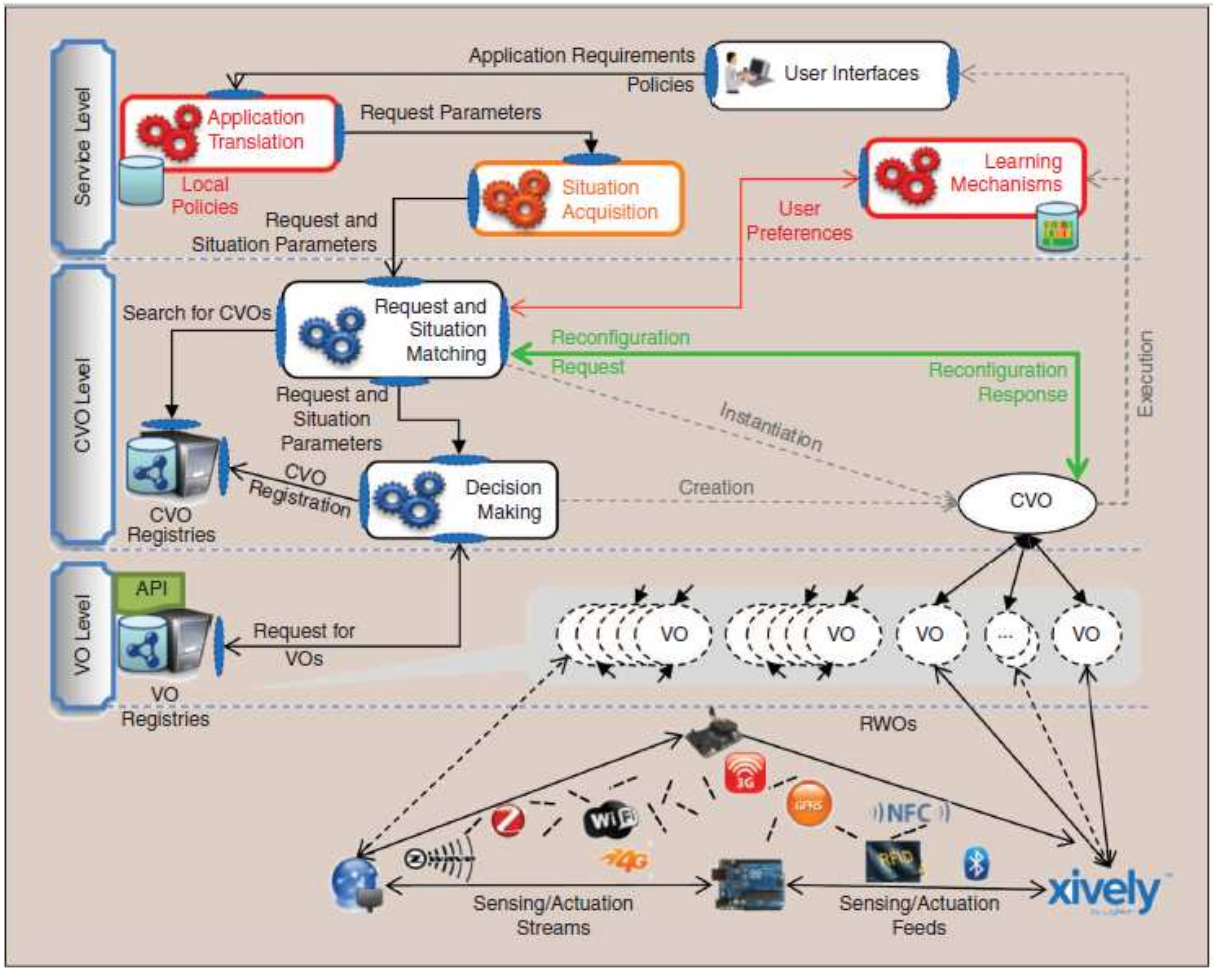


Figura 12 – Um modelo ilustrativo de um framework para Gerenciamento Cognitivo. Retirado de Foteinos *et al.* (2013)

informações referentes a cada VO são armazenadas nos Registros de VO, que podem ser recuperados por consulta a esses registros;

- **Nível CVO**

  Um CVO pode ser criado dinamicamente para oferecer um determinado serviço ou aplicação, que pode ser realizado através da composição de um ou mais VOs ou CVOs. Para além das funcionalidades disponíveis semelhantes àquelas registradas a nível de VO, existem mais duas estruturas vitais a este nível: correspondência de pedidos e situações e tomada de decisão. O primeiro constrói conhecimento e experiência relacionados a CVOs criados anteriormente para usá-los mais tarde para chegar a decisões confiáveis mais rapidamente. Isso é feito procurando por CVOs já existentes que atendam aos requisitos de serviço solicitados;

- **Nível de serviço**

  Este nível permite que usuários e partes interessadas definam características de um

serviço requerido por meio de interfaces apropriadas. Esses requisitos consistem em funções, políticas e parâmetros para o cumprimento de um determinado serviço. Neste nível existem mais dois componentes: aquisição da situação, para fornecer os parâmetros relevantes da situação, e de tradução do aplicativo, para inferir funções e políticas a partir da solicitação do usuário. Aqui, os mecanismos de aprendizagem são explorados para aprender as preferências dos usuários;

- **Características cognitivas**

  Os recursos cognitivos podem ser encontrados em todos os três níveis: otimização de exploração, aprendizado de máquina e técnicas de reconhecimento de padrões. Em particular, o nível VO usa otimização e aprendizado de máquina para selecionar links ideais entre VOs e CVOs e prever links problemáticos, o nível CVO usa reconhecimento de padrões para propor CVOs que podem ser reutilizados e o nível de serviço usa raciocínio semântico para fazer traduções para outro nível e técnicas de aprendizagem para adquirir preferências de uso (FOTEINOS *et al.*, 2013).

Outros grupos de pesquisa também se referem ao Gerenciamento Cognitivo, utilizando uma terminologia um pouco diferente mas, em suma, apresentando ideias muito similares.

Por exemplo, Jiang *et al.* (2014) apresentam uma arquitetura de rede para uma Internet das Coisas Cognitiva (ou IoT Cognitiva) envolvendo três planos: um Plano de Protocolo (Sistema IoT e suas camadas, neste caso), um plano de gerenciamento cognitivo (CM) e um plano de ajuste, responsável pelo ajuste das ações com base nas estratégias geradas pelo plano de CM. A Figura 14 ilustra este conceito. Outra questão importante, destacada por Sasidharan *et al.* (2014) é o recurso de autorregeneração (*self-healing*), indispensável em um contexto amplo como uma Cidade Inteligente, pois em muitos casos, uma intervenção corretiva humana direta, em caso de um mal funcionamento de partes do sistema, se torna inviável, à medida que a rede cresce.

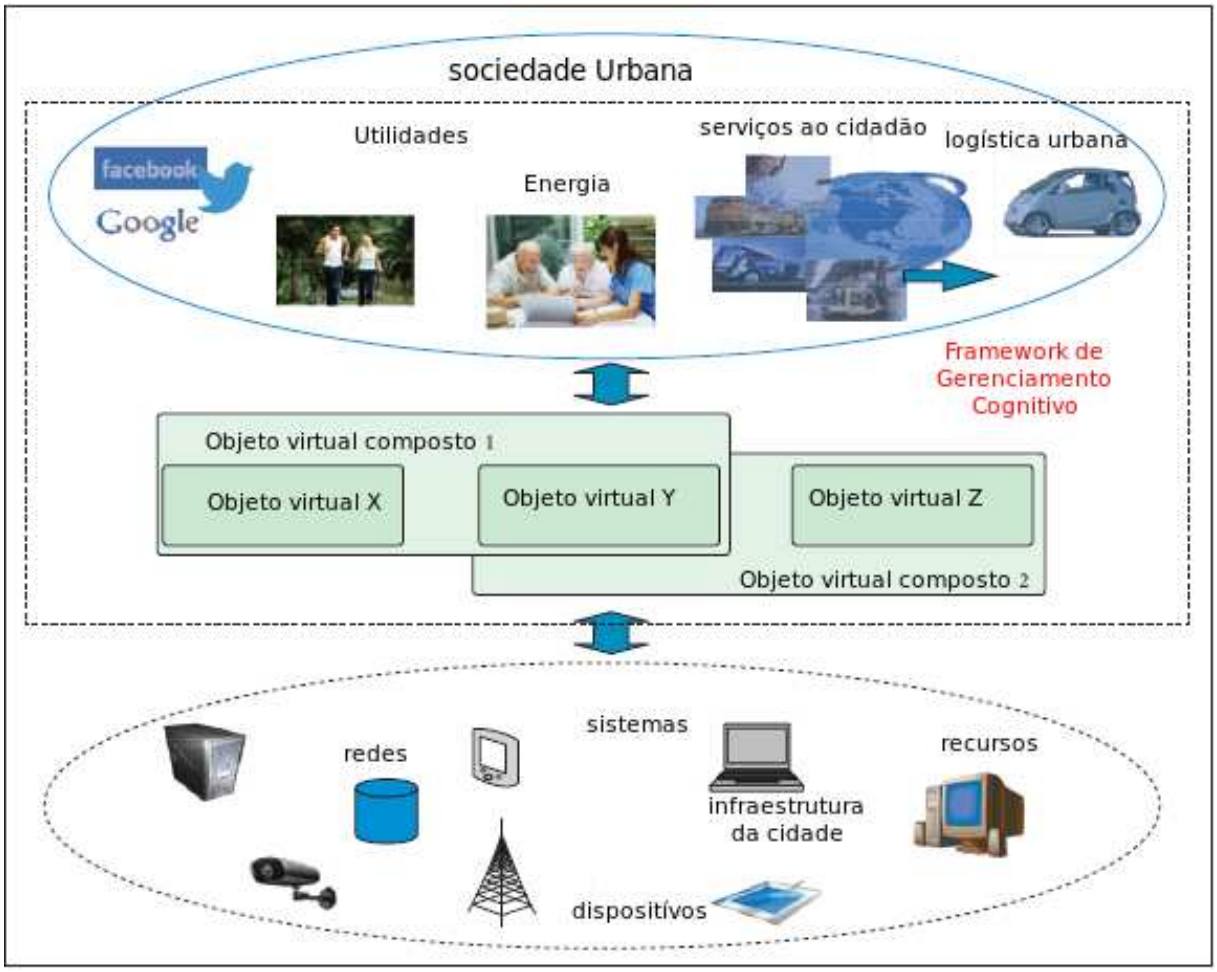


Figura 13 – Uma visão de alto nível de uma aplicação do framework discutido aplicado em cidades inteligentes. Adaptado de Vlacheas *et al.* (2013)

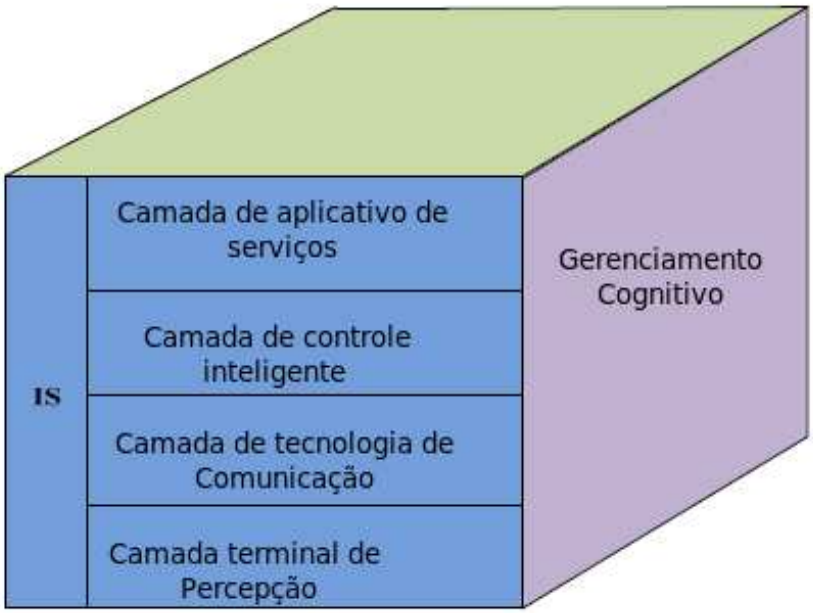


Figura 14 – Arquitetura de Gerenciamento Cognitivo para a Internet das Coisas. Adaptado de Jiang *et al.* (2014)

2.6 Resumo do Capítulo

Neste capítulo conduzimos uma breve revisão sobre alguns dos fundamentos mais importantes relacionados a este trabalho, trazendo referências e definições que serão importantes para seu entendimento geral.

O entendimento dos temas apresentados no presente capítulo é crucial para apreciar a complexidade e a inovação propostas posteriormente no Capítulo 5. Por exemplo, sem compreender o papel dos Sistemas Ciber-Físicos e da IoT Cognitiva, seria difícil avaliar a capacidade dos *Cognitive Twins* de interagir com um ambiente físico de forma inteligente.

A revisão de conceitos como *Sistemas de Sistemas* e *Gerenciamento Cognitivo* prepara o terreno para introduzir a ideia de *Cognitive Twins* construídos a partir de sistemas que formam coalizões entre si. A discussão sobre *Arquiteturas Cognitivas* e *IoT* fornece o contexto necessário para entender como os *Cognitive Twins* podem ser implementados e utilizados em cenários reais, abrindo possibilidades para aplicações inovadoras em diversos domínios.

A proposta que será detalhada no Capítulo 5 descreve a especificação de um *Cognitive Twin* evolutivo que integra os conceitos de *Arquiteturas Cognitivas*, *IoT*, *SoS*, *CPS*, e *Gerenciamento Cognitivo* em um sistema capaz de aprender e se adaptar autonomamente. O processo envolve a orquestração de dispositivos simples, utilizando técnicas de *Aprendizado de Máquina* e *Estratégias Evolutivas* para otimizar a topologia de conexão entre diferentes tipos de *Nodes*.

3 *Cognitive Twin*

3.1 Definições

Nos últimos anos, dentre as novas tecnologias surgindo em meio aos desafios de um mundo digital hiperconectado, há um interesse crescente na criação e operação de versões digitais de entidades físicas, os *Digital Twins* (DT, Gêmeos Digitais, em português). Introduzido originalmente por Grieves (2014), dentro do contexto da manufatura, como uma "Representação Virtual" dos produtos produzidos em uma linha de manufatura real, controlada digitalmente. Contendo um espaço "real", um espaço virtual e um fluxo de dados e informações entre ambos, o conceito evoluiu ao longo do tempo, conforme o entendimento sobre o tema ganhou forma. Embora não haja consenso sobre uma definição geral e canônica de *Digital Twins*, uma possível boa definição é aquela apresentada por Lim *et al.* (2019), como uma "réplica virtual de alta fidelidade de um ativo físico, com comunicação bidirecional em tempo real para fins de simulação e auxílio à decisão, além de recursos para aprimoramento do serviço do produto". Nesse *survey* pode-se ainda encontrar diversas outras definições que apareceram em trabalhos relevantes sobre o tema (LIM *et al.*, 2019).

Outra importante definição para *Digital Twin* pode ser encontrada em Arup (2019), onde tal entidade é definida como "uma combinação de um modelo computacional e um sistema do mundo real, projetada para monitorar, controlar e otimizar suas funcionalidades. Por meio de dados e *feedback*, tanto simulados quanto reais, um *Digital Twin* pode desenvolver capacidades para autonomia, aprender e raciocinar sobre seu ambiente". Partindo desta definição, Zhang *et al.* (2020) apresentam uma hierarquia que classifica os *Digital Twins*, desde os tipos mais simples e descritivos até os mais complexos. Esses últimos são classificados como *Cognitive Digital Twins* (Gêmeos Digitais Cognitivos, em português) e podem replicar, em algum nível, os processos cognitivos humanos e atuar com mínima ou nenhuma intervenção humana. Para os autores, esses processos cognitivos devem envolver, obrigatoriamente, autoconsciência (do inglês *self-awareness*, não confundir com *consciousness*) para a melhoria dos sistemas de percepção de um agente, envolvendo aquisição dinâmica de conhecimento e seu uso de modo a facilitar autoadaptação e capacidades de gerenciamento e tomada de decisão nos mundos digital e físico, de maneira info-simbiótica. Os autores ainda defendem que apenas com a incorporação do fator "cognição", todo o potencial disruptivo desta tecnologia será atingido. Por "cognição", os autores consideram as capacidades e processos mentais e de autoconsciência que alavancam aquisição dinâmica de conhecimento, gestão e sua sinergia nos mundos digital e físico, abrangendo dimensões que se relacionam com estímulos, tempo, interação,

objetivos e mudança de contexto/situação. A Figura 15 ilustra a conceitualização de um *Cognitive Digital Twin*, conforme apresentada por Zhang *et al.* (2020).

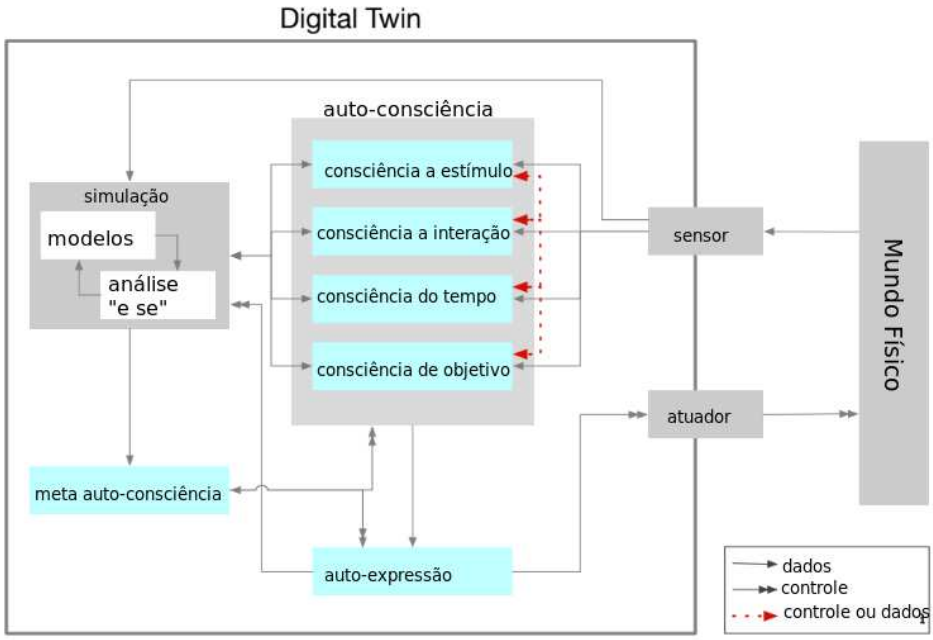


Figura 15 – Conceitualização de um *Cognitive Twin* com capacidades de autoconsciência completa, adaptado de Zhang *et al.* (2020)

Entre outras definições, Abburu *et al.* (2020b) apresentam três níveis de *Digital Twins* — *Digital Twins*, *Hybrid Digital Twins* e *Cognitive Digital Twins* — sendo um *Cognitive Twin* definido como um sistema híbrido de autoaprendizagem e proatividade, que otimiza suas capacidades cognitivas ao longo do tempo com base em dados coletados e experiência adquirida. Também deve conseguir combinar técnicas de análise de IA e conhecimento especializado para controlar o comportamento do sistema e resolver problemas emergentes. Os autores ainda apresentam uma descrição de um *toolbox* para a criação de *Cognitive Twins* focados em processos industriais. Na sessão 3.2 elaboraremos um pouco mais sobre os três níveis aqui mencionados.

Posteriormente, o mesmo grupo de pesquisa (ABBURU *et al.*, 2020a) definiu que o principal objetivo da camada cognitiva apresentada na hierarquia por eles proposta (apresentada na Figura 16) era permitir a resolução de um problema ao introduzir novos conhecimentos responsáveis pelo fornecimento de *insights* para o processo de criação e aprendizado do modelo, ou seja, a introdução de restrições de interpretação em cima dos dados coletados originalmente. Para eles, a cognição (no sentido computacional) atua sustentada por modelos existentes, que podem ser definidos utilizando IA, de modo a estender a inteligência e fornecer entendimento aprofundado e estratégias de *Reasoning*. Os autores sintetizam esse processo geral em quatro passos:

- Inserção de novos conhecimentos (relevantes ao problema);

- Aprendizado de modelos mais precisos ao aplicar novos conhecimentos;
- Melhoria do entendimento situacional (redução de incertezas) ao aplicar novos modelos;
- Planejamento de ações para resolução do problema, baseado no entendimento situacional aprimorado.

Mais especificamente, Abburu *et al.* (2020a) descrevem esses passos da seguinte maneira:

- Extração e aquisição de conhecimento, construído a partir de fontes de dados existentes (por exemplo, conteúdo não estruturado e semiestruturado) e de especialistas, respectivamente. O objetivo é coletar conhecimento, que está relacionado com incertezas em dados e modelos. Uma vez que o processo está relacionado com o apoio à compreensão humana, é importante que o processo seja conduzido por estruturas de conhecimento bem definidas (como Grafos de Conhecimento) que fornecem uma descrição geral do domínio;
- Aprendizagem, que engloba a aplicação de novos conhecimentos sobre os dados, modelos e métodos existentes, visando aprender modelos mais precisos (de conjuntos de dados existentes). São três as atividades principais:
  - Tornar conjuntos de dados existentes livres de anomalias, permitindo seu uso na aprendizagem de modelos mais precisos;
  - Melhorar os métodos de aprendizagem usados, introduzindo algumas restrições orientadas pelo conhecimento no processo de aprendizagem;
  - Adicionar novos métodos que podem complementar os existentes (no contexto das incertezas mencionadas acima).
- Compreensão, que está relacionada com a aplicação de novos modelos a dados em tempo real para obter uma melhor interpretação das situações de interesse (por exemplo, detecção de problema/anomalia). (Aqui os autores supõem que, assim como na cognição semelhante à humana, esse processo pode ser iterativo, ou seja, compreender um processo pode gerar dados, que podem ser usado para melhorar o processo de aprendizagem, como em aprendizado por reforço);
- Planejamento, para definir ações ótimas baseando-se na compreensão do comportamento do sistema.

Fernández *et al.* (2019) elaboram em seu trabalho uma linha evolutiva partindo de *Digital Twin* até o conceito por eles introduzido (e enfatizado, apesar de não conter

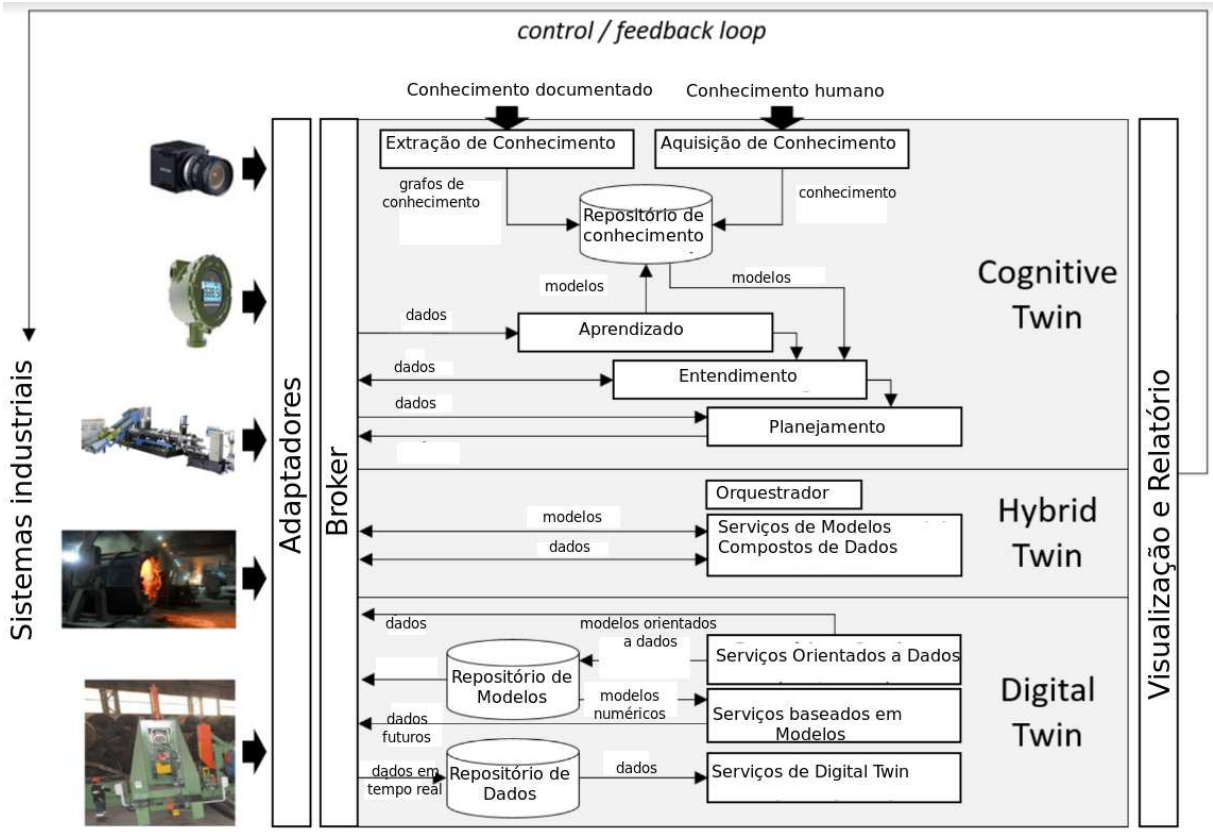


Figura 16 – Arquitetura Conceitual para *Digital/Cognitive Twins* Adaptada do apresentado em Abburu *et al.* (2020a) e Abburu *et al.* (2020b)

ideias exatamente novas) de *Associative Cognitive Digital Twin*. Segundo os autores, um DT refere-se basicamente a uma réplica digital em tempo real de um recurso físico. Em seguida, os mesmos apresentam uma definição própria para DT como uma "descrição, representação, visualização ou modelo digital viável de uma entidade física ou abstrata (objeto, sistema, estrutura, recurso, processo, serviço etc) para um dado propósito". Os autores ainda observam que uma entidade pode ter multiplos DTs (visualizações) e descrevem um DT através da função:

$$DT = Modelo(Entidade, Propósito)$$

Ainda sobre DTs, os autores definem suas dimensões internas como sendo: complexidade, nível de conhecimento, nível de abstração, granularidade, cardinalidade, tipos de descrição, ciclo de vida, propósito etc.

Na sequência, os autores definem um *Cognitive Digital Twin* (C-DT) como "uma representação e ampliação (*augmentation*, no original em inglês) digital de uma entidade ao longo de seu ciclo de vida para otimizar as atividades cognitivas. Pode ser considerado um especialista digital ou copiloto, que pode aprender e evoluir, integrando diferentes fontes de informação para o propósito proposto". Um C-DT pode ser descrito pela seguinte

descrição funcional (FERNÁNDEZ *et al.*, 2019):

$$C\text{-}DT = Modelo(Entidade, Propósito\ Cognitivo)$$

A estrutura de um C-DT emularia parcialmente a estrutura dos modelos mentais humanos correspondentes, estruturas muito eficientes, otimizadas e adaptadas, por milhões de anos de evolução.

Por fim, Fernández *et al.* (2019) apresentam o conceito de *Associative Cognitive Digital Twin* (AC-DT) como "uma descrição aumentada contextual que inclui explicitamente os relacionamentos associados relevantes, conexões ou canais de informação da entidade considerada com as outras entidades, para a aplicação e finalidade consideradas". Um AC-DT pode ser descrito pela seguinte descrição funcional:

$$AC\text{-}DT = Modelo(Entidade, Relações\ Associadas, Propósito\ Cognitivo)$$

Um AC-DT poderia utilizar diversos canais de informação como interfaces humano-máquina e informações de realidade aumentada, informações contextuais, dados relacionados ao problema, dados históricos, conhecimento associado, interface de voz, gerenciamento, instruções de orientação e controle, legislação, verificação de regulamentos, objetos binários grandes etc. Como exemplo, os autores trazem um AC-DT de um controlador de velocidade de um trem, que poderia incluir os dados de referência da rota ferroviária e um sistema autônomo de consciência situacional em tempo real (isto é, o controlador atua diretamente sobre o trem, enquanto o mesmo está em funcionamento) para analisar os dados correspondentes. Poderia ter canais-conexões-relacionamentos com sinalização ferroviária, sistemas automáticos de proteção de trens, variáveis de estado dos trens, normas de condução de segurança, câmeras de vídeo, GPS etc. Um objeto para esse AC-DT poderia disparar alarmes para o operador do trem, gerar comandos controlados remotamente e gerenciar a velocidade do trem. Os principais blocos de construção de uma tecnologia AC-DT são, segundo os autores (FERNÁNDEZ *et al.*, 2019), modelos digitais cognitivos, bancos de dados em tempo real, relacionamentos hierárquicos e análises associadas, simulação e ferramentas preditivas, para melhor entender o comportamento e o desempenho dos sistemas ou serviços e para a tomada de decisões. A Figura 17 sumariza o *framework* proposto pelos autores para a construção de AC-DTs.

Visando aperfeiçoar o conceito de *Cognitive Twin*, Eirinakis *et al.* (2020) apresentam o conceito de *Enhanced Cognitive Twin* (ECT). Partem do conceito de um *Cognitive Twin* como um DT dotado de cognição, sendo essa cognição emergente de uma camada de processamento em tempo real (mesmo entendimento anterior do termo) onde observações, conhecimento e experiência interoperam para permitir um entendimento e controle

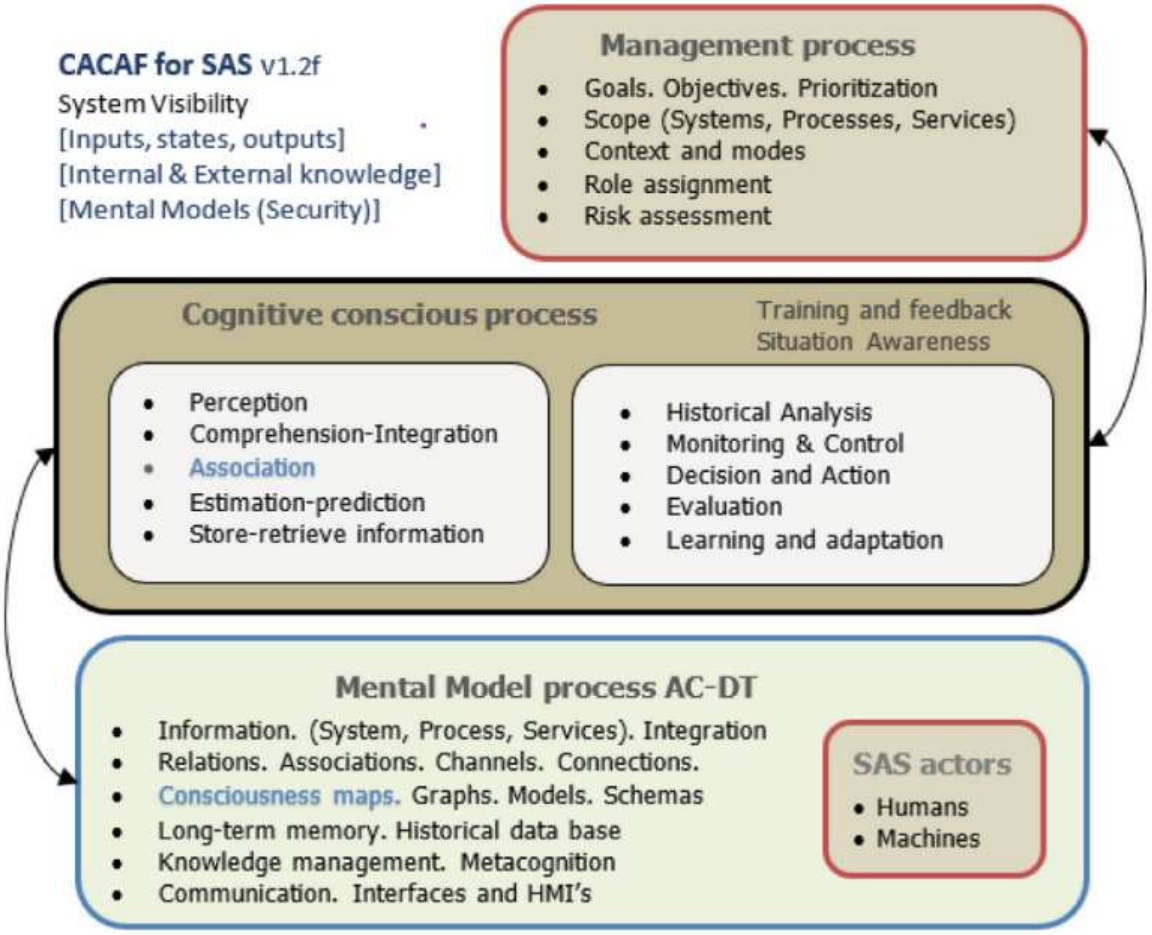


Figura 17 – Framework para construção de *Associative Cognitive Digital Twins* (AC-DT). Reproduzida de Fernández *et al.* (2019)

do comportamento de uma entidade física. A grande diferença entre um CT e um ECT é que neste último, parte-se do princípio de que o objeto físico sendo virtualizado pode ter seu comportamento alterado e/ou modificado ao longo do tempo. Dessa forma, em um ECT, o próprio comportamento do objeto físico é modelado e continuamente re-estimado, por meio de métricas que categorizam o estado ou função desse objeto físico, além de algoritmos de otimização que buscam atualizar constantemente esse modelo. Dessa forma, um ECT possui, além das características usuais de um CT, habilidades de decidir as melhores ações sobre a entidade física, mesmo que seu comportamento mude ao longo do tempo. A Figura 18 ilustra o conceito de *Enhanced Cognitive Twin*.

Em um outro trabalho da área, Lu *et al.* (2020) apresentam um *Cognitive Twin* centrado em Grafos de Conhecimento. Neste artigo, os autores explicitamente trazem suas próprias considerações sobre as definições de DT e CT. Segundo os mesmos, um DT pode ser definido como uma duplicata digital de entidades, com comunicação bidirecional em tempo real habilitada entre os espaços cibernéticos e físicos. Indo além, desenvolvem o

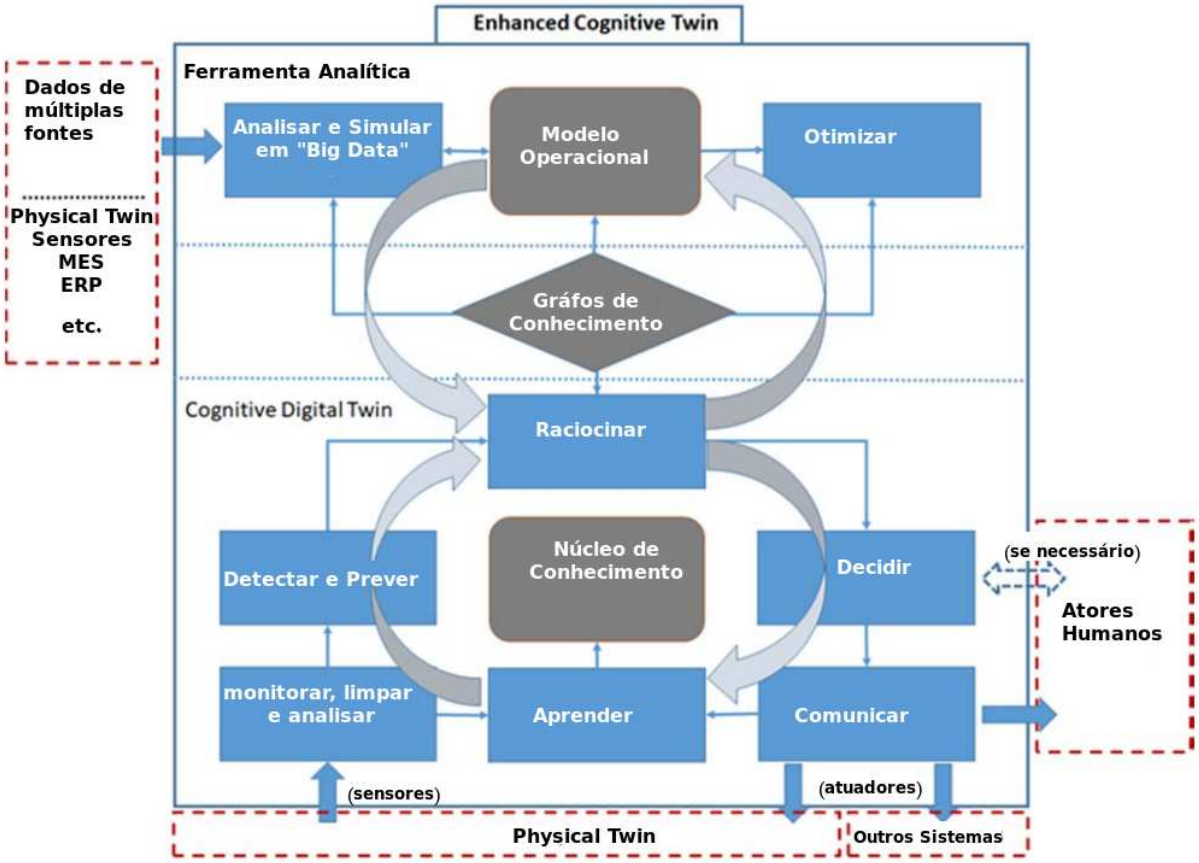


Figura 18 – Arquitetura Conceitual para um *Enhanced Cognitive Twin*. Adaptado de Eirinakis *et al.* (2020)

início de uma formalização para seu modelo:

$$DT_{sys} = PE\{Sys\} \cup VE\{\Sigma Model(Ms, Mp, Mth, Ml, Mt, Mm)\} \cup Comm\{\Sigma Data(EntitySt, EntityDe, Dtype, Datacontent)\}$$

onde DT_{sys} se refere a um DT do sistema Sys; $PE\{Sys\}$ refere-se à entidade física de Sys; $VE\{\Sigma Model(Ms, Mp, Mth, Ml, Mt, Mm)\}$ refere-se a uma coleção de modelos relacionados a Sys. Cada modelo inclui vários itens:

- Ms (Estrutura do Modelo): topologia do modelo, entradas, saídas e parâmetros;
- Mp (Propósito do Modelo): as visões de modelagem, "por que o modelo é necessário?"
- Mth (Teoria de Modelagem): a base matemática da modelagem, por exemplo, sistema de equações diferencial-algébrico;

- Ml (Linguagem de Modelagem): qualquer linguagem que expressa informação ou conhecimento, ou sistemas em uma estrutura definida por um conjunto consistente de regras;
- Mt (Ferramenta de Modelagem): ferramentas para implementação de modelos;
- Mm (Método de Modelagem): um conjunto de conceitos para explicar "como desenvolver modelos usando uma determinada linguagem em uma ferramenta de modelagem para representar os formalismos?", por exemplo. modelagem de elementos finitos e modelagem de equações estruturais.

$Comm\{\Sigma Data(EntitySt, EntityDe, Dtype, Datacontent)\}$ refere-se aos fluxos de dados e informações entre entidades físicas e entidades virtuais. Cada fluxo inclui vários itens:

- $EntitySt$ (Entidades de Início): início do fluxo de dados e informações;
- $EntityDe$ (Entidades de Destino): destino do fluxo de dados e informações;
- $Dtype$ (Tipo de dados): tipo de dados, como dados em tempo real e dados offline;
- $Datacontent$ (Conteúdo dos dados): os dados usados neste fluxo de dados.

De modo análogo, os autores apresentam um esboço de formalização para *Cognitive Twin*:

$$\begin{aligned} CT_{sys} = PE\{Sys\} \cup VE\{&\Sigma Model(Ms_t, Mp_t, Mth_t, Ml_t, Mt_t, Mm_t), \\ &Ontopology(entities, relationships)\} \cup \\ &Comm\{\Sigma Data(EntitySt, EntityDe, Dtype, Datacontent)\}, \\ &t = 1, 2, 3, ..., \textit{marcações de tempo num ciclo de vida} \end{aligned}$$

Onde CT_{sys} se refere a um CT do sistema Sys; $PE\{Sys\}$ refere-se à entidade física de Sys; $VE\{\Sigma Model(Ms_t, Mp_t, Mth_t, Ml_t, Mt_t, Mm_t), Ontopology(entities, relationships)\}$ referem-se a uma coleção de modelos relacionados a Sys. Diferente dos DTs, cada modelo no CT é adicionado com um *timestamp* no ciclo de vida. Exceto para os itens em DTs, $Ontopology(entities, relationships)$ refere-se à ontologia para representar a topologia entre os Modelos.

- As *entities* (entidades) referem-se a todas as informações relacionadas aos modelos, como composições;
- As *relationships* (relações) referem-se a todas as inter-relações das entidades.

A Figura 19 ilustra as diferenças e semelhanças entre *Digital Twins* e *Cognitive Twins* segundo Lu *et al.* (2020).

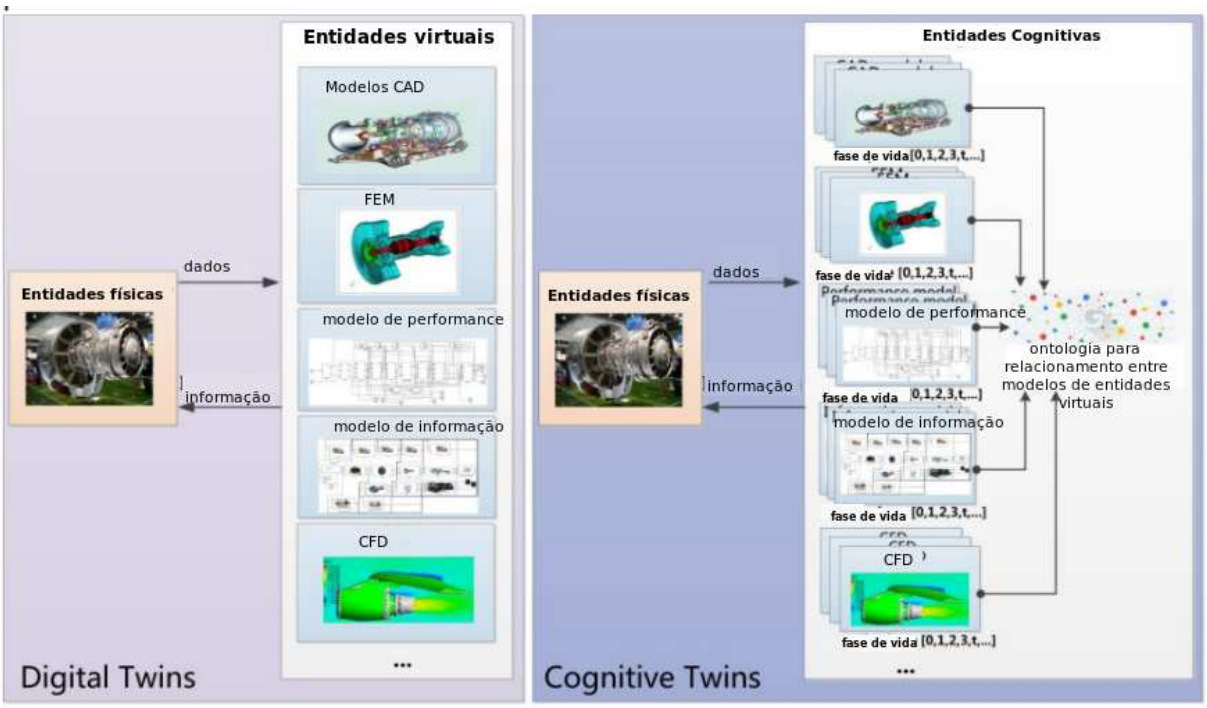


Figura 19 – Diferenças e semelhanças entre *Digital Twins* e *Cognitive Twins*. Adaptado de Lu *et al.* (2020)

Argumentando que muito de nossa vida social existe na (e através da) Internet e considerando-se que, com a ascensão dos dispositivos IoT, os *Cognitive Twins* poderiam de fato interagir com o mundo mais diretamente, Somers *et al.* (2020) trazem uma visão mais centrada no comportamento do agente. No artigo citado, é utilizado um conceito de *Cognitive Twin* como um "reflexo digital do usuário, destinado a tomar decisões e realizar tarefas em nome deste", para "destacar o papel fundamental que os mecanismos cognitivos desempenham na modelagem e na tomada de decisões humana, no espaço digital IoT". Lá, eles aplicam esse conceito como forma de modelar processos cognitivos subjacentes às decisões do usuário. Além disso, nessa mesma direção, Du *et al.* (2020) apresentam um modelo de *Digital Twin* pessoal de cognição orientada por informações (Cog-DT), que definem como uma "réplica digital do processo cognitivo de uma pessoa em relação ao processamento de informações", modelando, monitorando e predizendo o comportamento da pessoa através do processamento de diversos tipos de informação.

A tabela 3 sumariza as principais definições encontradas na literatura para *Cognitive Twin*. Aqui neste trabalho, com base nas definições encontradas e dentro do escopo do que será apresentado, assumiremos a seguinte definição (de nossa própria autoria), baseada nas definições anteriores:

Um *Cognitive Twin* é uma réplica digital dos processos dinâmicos e cognitivos — ou apenas cognitivos — de um sistema físico inteligente, geralmente voltada à representação parcial de uma pessoa. Estes processos cognitivos referem-se àqueles identificados nas teorias cognitivas, tais como percepção, memória, comportamento, adaptação, planejamento, aprendizado, *Reasoning* (Raciocínio) etc. A classificação de um agente como *Cognitive Twin* refere-se não apenas à duplicação do comportamento observável do agente virtual, em relação ao original, mas também da possibilidade de aprofundada investigação dos processos internos dessa réplica.

É importante salientar que neste trabalho optamos pela utilização do termo *Cognitive Twin* em detrimento de *Digital Twin*. Essa escolha é fundamentada na expansão significativa do conceito tradicional de *Digital Twin* (DT) para abranger não apenas a replicação digital de processos físicos e sistemas, mas também a integração e a simulação dos processos cognitivos humanos, permitindo uma interação autônoma com o ambiente e a tomada de decisões baseadas em conhecimento dinâmico e adaptativo.

A ideia aqui é transparecer que *Cognitive Twins* são projetados para replicar não apenas os aspectos físicos de uma entidade, mas principalmente os processos cognitivos humanos. Isso envolve a simulação de funções mentais como percepção, memória, aprendizado, e raciocínio, permitindo à réplica digital "pensar" e "compreender" seu ambiente de forma semelhante a um humano.

A integração desses processos cognitivos confere aos *Cognitive Twins* uma autonomia significativa, permitindo-lhes tomar decisões e executar ações com base no conhecimento adquirido através da interação com o ambiente. Essa capacidade de autoaprendizagem e adaptação é fundamental para operações complexas e ambientes dinâmicos, onde as condições podem mudar rapidamente.

Tabela 3 – Definições encontradas na Literatura para *Cognitive Twin*

Trabalho	Cognitive Twin
(ZHANG *et al.*, 2020)	Combinação de modelo computacional e sistema do mundo real que pode replicar processos cognitivos humanos e executar ações conscientes autonomamente
(ABBURU *et al.*, 2020b)	Um sistema híbrido, de autoaprendizagem e proativo que otimiza suas capacidades cognitivas
(FERNÁNDEZ *et al.*, 2019)	Uma representação e ampliação digital de uma entidade ao longo de seu ciclo de vida para otimizar as atividades cognitivas
(EIRINAKIS *et al.*, 2020)	DT dotado de cognição, sendo essa cognição emergente de uma camada de processamento em tempo real onde observações, conhecimento e experiência interoperam
(LU *et al.*, 2020)	Duplicata digital de uma entidade com comunicação bilateral entre espaço físico e cibernético. Atua ainda através de domínios diferentes com suas interrelações
(SOMERS *et al.*, 2020)	Reflexo digital do usuário, destinado a tomar decisões e realizar tarefas em nome deste usuário
(DU *et al.*, 2020)	Réplica digital do processo cognitivo de uma pessoa em relação ao processamento de informações

3.2 Taxonomias de Digital Twins

Conforme mencionado anteriormente neste capítulo, Zhang *et al.* (2020) apresentam uma proposta de classificação de *Digital Twins* de acordo com suas capacidades cognitivas. Dependendo de como o *Digital Twin* lida com os dados e o grau de inteligência que o mesmo apresenta, ele pode ser de um dos seguintes tipos:

- *Digital Twin* Descritivo: provê informações acerca do mundo como ele é, mas não desenvolve muito sobre "como ele poderia ser";
- *Digital Twin* Preditivo: tipicamente realiza uma análise sobre os dados, como em técnicas estatísticas, Mineração de Dados e Aprendizado de Máquinas, para extrapolar sobre o que pode acontecer no futuro. Entidades pertencentes a esta classe produzem boa extrapolação apenas sobre aquilo que pode ser inferido a partir do histórico ou dos dados de treinamento;
- *Digital Twin* Prescritivo: entidades desta classe podem valer-se de simulações e análises contrafactuais (do tipo "e se"), provendo uma maior explicação acerca dos mecanismos de comportamento do sistema. A exploração de múltiplos cenários contra-

factuais pode fornecer "insights" acerca de boas soluções para problemas e otimizar ações em eventos futuros ou reduzir riscos;

- *Cognitive Digital Twin*: a mais alta classificação de inteligência, essas entidades conseguiriam replicar alguns dos processos de cognição humana e executar ações conscientes autonomamente, com mínima ou nenhuma intervenção humana.

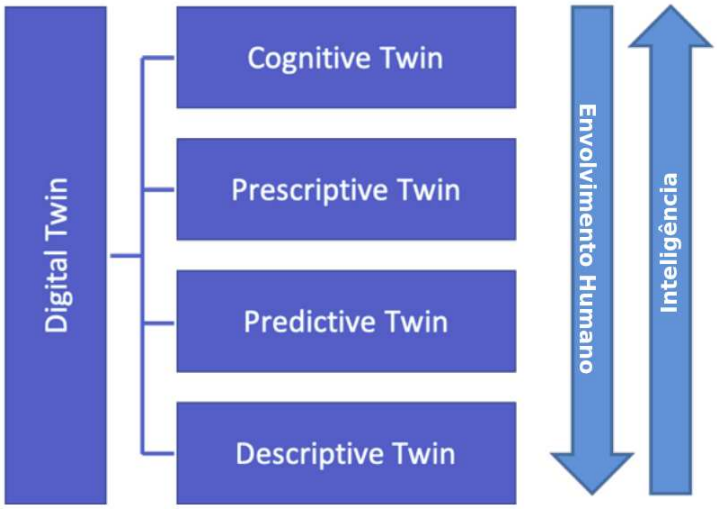


Figura 20 – Modelo de classificação de *Digital Twins* adaptado de Zhang *et al.* (2020)

A Figura 20 ilustra a hierarquia proposta por Zhang *et al.* (2020). O autor desta tese defende que *Digital Twins* construídos seguindo modelos cognitivos bem estabelecidos (ou, mais diretamente, Arquiteturas Cognitivas) enquadram-se na mais alta categoria, embora o conceito de "consciência" leve a armadilhas argumentativas perigosas e deva ser deixada de lado em favor do termo "intenção".

Aprofundando sobre os tipos de *Cognitive Digital Twins*, Zhang *et al.* (2020) apresentam cinco níveis de autoconsciência:

- ***Digital Twins* Conscientes a Estímulos**: Representando a base para todos os outros tipos listados, este *Digital Twin* apresenta essencialmente uma natureza reativa, conseguindo demonstrar que ao detectar e reagir a estímulos e eventos no ambiente que monitora, pode influenciar o estado e o comportamento do sistema observado. Não envolve simulações nem análises contra-factuais ("e se") pois ambas valem-se de um fator temporal. Como exemplo, pode-se considerar um *Digital Twin* que demonstre como um sistema reage e adapta-se a eventos que requerem otimização de rotas e políticas, com objetivos predefinidos;
- ***Digital Twins* Conscientes a Interações**: Entidades que pertencem a esta categoria operam sobre o conhecimento adquirido a partir das interações com o mundo digital. Tais interações podem relacionar-se com dados históricos ou oriundos de

simulações de entidades reais interagindo com o mundo físico. Além disso, *Digital Twins* Conscientes a Interações podem ser guiados por dados vindos do mundo físico. Quando o conhecimento vindo do mundo físico é processado por sua contraparte digital, ocorre um enriquecimento do conhecimento devido à posterior simulação de situações, planejamento e tomada de decisão mais bem informada, que pode ser realimentada para o sistema real;

- ***Digital Twins* Conscientes ao Tempo**: Estes *Digital Twins* utilizam dados relacionados ao desempenho do sistema quando o mesmo responde autonomamente a estímulos. Suportam ainda planejamento preditivo e proativo de modo automático (ou semi-automático) baseando-se em dados históricos. Devido a essa capacidade, tais sistemas podem avaliar periodicamente se uma dada estratégia irá degradar seu desempenho no futuro e, em caso positivo, uma nova decisão pode ser gerada com base em análise contrafactual e posta em funcionamento antes da perda de desempenho. Aqui ocorre ainda uma "info-simbiose" na qual o *Twin* monitora feedbacks variantes no tempo e atualiza sua base de conhecimento;

- ***Digital Twins* Conscientes a Objetivos**: Nesta categoria, os *Digital Twins* tem conhecimento explicito sobre seus objetivos e estados desejados. A combinação de objetivos definidos no design do sistema e de objetivos ajustados em tempo de execução podem facilitar a resposta para adversidades que ocorram no mundo físico. Caso um determinado objetivo seja impossível de ser satisfeito, tais objetivos poderiam ser relaxados (durante a execução) e uma solução sub-ótima permitida;

- ***Digital Twins* Meta-autoconscientes**: *Digital Twins* nesta categoria são conscientes dos demais níveis e adaptam-se a como eles são materializados e/ou decidem os *trade-offs* entre os diferentes níveis de autoconsciência. Podem, por exemplo, ajustar a frequência de predição e replanejamento, caso percebam que o ambiente não muda tão drasticamente ou mesmo mudar o algoritmo de planejamento em resposta a estímulos.

A tabela 4, adaptada de Zhang *et al.* (2020), sumariza as informações importantes trazidas no trabalho citado.

Em outra proposta de classificação, Abburu *et al.* (2020b) apresentam uma categorização mais simples, ilustrada na Figura 21 com três níveis de abstração:

- *Digital Twin* (DT): Uma réplica digital de um sistema físico que captura características e comportamentos de tal sistema. Seu propósito é permitir medidas, simulações e experimentações com a réplica para que assim adquira-se conhecimento acerca da contraparte física;

Tabela 4 – Atributos relativos à autoconsciência. Adaptado de (ZHANG *et al.*, 2020)

Atributos	Awareness				
	Estímulos	Interações	Tempo	Objetivos	Meta
Parâmetros	Estímulos, eventos e estados (por exemplo, carga, performance, vida útil da bateria, etc.)	Fonte de dados, topologia de coordenação fisica (por exemplo, capturado usando modelos de objeto, análise de rede)	Dados históricos que relacionam-se com eventos, estados etc.	Objetivos individuais ou coletivos	Performance dos outros níveis de consciência
Técnicas	Regras de condição-ação, algorítimos baseados em limiares	Coordenação de Nós fazendo "apostas" por tarefas, compartilhamento de conhecimento e negociação	Técnicas Estatísticas, classificadores de Machine Learning	otimização baseada em utilidade, aprendizado por reforço	"Balde" de Modelos, algorítimo de otimização do bandido
Propósitos	Monitoramento de eventos, atualização de parâmetros, mitigação de discrepâncias	Tomada de decisão coordenada	Predição, adaptação baseada no histórico	Maximização de utilidade de decisão	Coordenação entre os níveis de consciência
Reativo ou Proativo	Reativo	Ambos	Ambos	Ambos	Ambos
Simulações e análises contrafactuais	Não	Opcional	Sim	Opcional	Opcional

- *Digital Twin* Híbrido (HT): Uma extensão do modelo anterior, entidades nesta categoria conseguiriam reconhecer, prever e comunicar comportamentos sub-ótimos — mas previsíveis — antes que tais comportamentos ocorram. Modelos híbridos combinam sensores de diversas fontes com análises de Inteligência Artificial para atingir capacidades preditivas maiores que as anteriores enquanto otimiza e controla o sistema físico;

- *Cognitive Digital Twin* (CT): Tais modelos incorporam uma série de recursos cognitivos que permitirão perceber comportamentos complexos e não-previstos e raciocinar acerca de estratégias de otimização, levando a sistemas que continuamente evoluem a sua própria estrutura digital tal qual seu comportamento. Um *Cognitive Twin* seria, portanto, um híbrido, dotado de aprendizado e proativo, que otimiza suas próprias capacidades cognitivas ao longo do tempo baseado nos dados e experiência adquiridos.

Novamente, o autor deste trabalho defende que os conceitos que serão aqui mos-

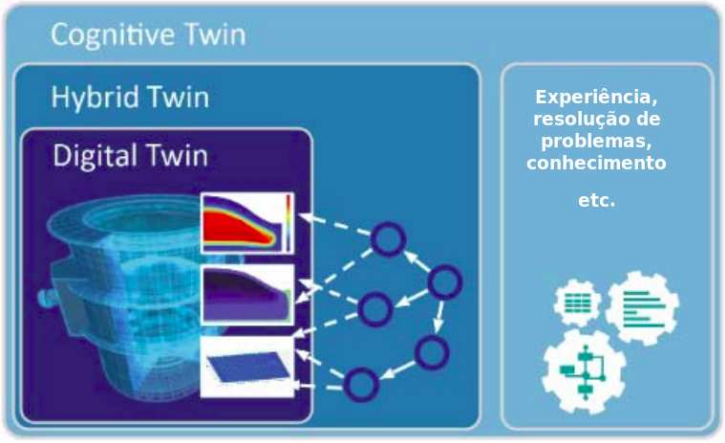


Figura 21 – Modelo de classificação de *Digital Twins* adaptado de Abburu *et al.* (2020b)

trados enquadram-se na mais alta categoria também desta hierarquia.

3.3 Aplicações

Uma série de aplicações aparece na literatura para uso de *Cognitive Twins*, sejam tais aplicações de fato implementadas ou apenas propostas. Abburu *et al.* (2020b) enumeram possíveis casos de uso para a ferramenta proposta, sendo elas:

1. Otimização de operações de um centro de tratamento de gás na produção de alumínio;
2. Minimização de riscos de saúde e segurança e maximização de rendimento de metais na produção de silício;
3. Monitoramento de recursos na produção de aço e produtos relacionados;
4. Monitoramento em tempo real de produtos concluídos, visando eficiência operacional;
5. Melhoramento da eficiência em um trocador de calor;

Posteriormente, o mesmo grupo de pesquisa (ABBURU *et al.*, 2020a) aplicou sua ferramenta (COGNITWIN) em um problema real acerca do controle preditivo em uma indústria de processamento de aço.

Seguindo os conceitos por eles apresentados, Somers, Oltramari, e Lebiere (SOMERS *et al.*, 2021; SOMERS *et al.*, 2020) usaram a arquitetura cognitiva ACT-R (ANDERSON *et al.*, 2004; ANDERSON, 2009) — seguindo uma abordagem híbrida de dados e baseada em conhecimento — para implementar um *Cognitive Twin* como um assistente

pessoal que mimetiza o processo humano de tomada de decisões, para aprender o comportamento do usuário com base em dados passados, relativos à rede social deste mesmo usuário. Em seguida, como prova de conceito, o assistente utiliza o conhecimento adquirido para selecionar convidados para uma festa. Em Somers *et al.* (2020), os autores argumentam que o planejamento de uma festa é uma atividade melhor tratada seguindo uma abordagem híbrida, onde aspectos como os passos de planejamento do evento, categorias dietéticas etc, são melhor tratados ao nível simbólico, enquanto preferências dietéticas, quem são os possíveis convidados, agendas, tamanho da festa e outras categorias que apresentam regularidades estatísticas podem ser diretamente aprendidas dos dados. Antes de modelar os *Cognitive Twins*, uma série de dados sintéticos foram gerados representando não apenas as agendas dos indivíduos como também suas interações.

Como aplicação para sua ferramenta proposta, o Cog-DT, em uma plataforma de Realidade Virtual, Du *et al.* (2020) utilizaram um treinamento multimodal com dados vindos de sensores de neuroimagens, comportamento fisiológico (foco do olhar, frequência de movimento ocular, fluxo de calor, pressão arterial etc) e ergonômico (padrão de postura ao andar) para modelar a reação de um indivíduo acerca de diferentes estímulos, mapeando a Carga Cognitiva baseada na quantidade e tipo de informação e minimizando essa carga. O agente foi então utilizado para investigar como transmitir informações de maneira efetiva ao usuário, evitando sobrecarga cognitiva, considerando o formato da informação (números, formas, texto etc), além de quando e como a informação deve ser apresentada ao usuário (que utiliza um *headset*).

Rozanec *et al.* (2020) utilizaram um *Cognitive Digital Twin* em um cenário de manufatura para, através da sugestão de ações para um usuário, auxiliar na melhoria de Indicadores de Desempenho (KPIs, *Key Performance Indicators* em inglês) — detectando possíveis anomalias e prevendo seus impactos na produção — e planejando alguns aspectos da própria produção, como reagendamento de planos já existentes (ROŽANEC *et al.*, 2021).

3.4 Resumo do Capítulo

Neste capítulo discutiu-se acerca das definições, diferentes modelos de taxonomia e aplicações de estudos recentes em *Cognitive Twin*. Foi ainda fornecida uma definição própria sobre o conceito.

Os temas discutidos neste capítulo conectam-se com o que será discutido no Capítulo 4. Esta conexão reside na evolução dos Sistemas Cognitivos em direção a uma maior complexidade, interconectividade e autonomia. Os *Cognitive Twins* representam uma outra visão em relação aos modelos de *Digital Twins*, introduzindo capacidades cognitivas que permitem uma interação mais sofisticada com o ambiente e a tomada de decisões

autônoma. O *Distributed Cognitive Toolkit* (DCT, detalhado no capítulo seguinte), por sua vez, fornece uma plataforma tecnológica para implementar e distribuir esses modelos cognitivos avançados em uma variedade de dispositivos, ampliando seu alcance e eficácia. Ambos os conceitos refletem um movimento em direção a sistemas cada vez mais inteligentes e autônomos, capazes de aprender, adaptar-se e operar de forma independente em ambientes complexos e dinâmicos. A implementação prática de *Cognitive Twins* pode ser grandemente facilitada e enriquecida pelo uso de ferramentas como o DCT, que oferecem a flexibilidade, escalabilidade e interoperabilidade necessárias para desenvolver, testar e implantar esses sistemas em ambientes reais.

4 O DCT (Distributed Cognitive Toolkit)

4.1 O CST (*Cognitive Systems Toolkit*) e sua Arquitetura de Referência

O CST (*Cognitive Systems Toolkit*) é um projeto de longo prazo sendo desenvolvido em nosso grupo de pesquisa, no DCA-FEEC-UNICAMP, visando construir um *toolkit* de propósito geral destinado a ser usado na construção de Arquiteturas Cognitivas, fornecendo várias soluções de modelagem, podendo ser integrado em diferentes configurações, utilizando Java como linguagem de implementação (PARAENSE *et al.*, 2016). A concepção do CST envolve ideias oriundas de diversas arquiteturas cognitivas, tais como Clarion (SUN, 2015) e LIDA (FRANKLIN *et al.*, 2014). A ideia principal por trás do CST era ser uma espécie de "canivete suíço" de modelos cognitivos, que poderiam ser organizados de diferentes maneiras conforme o projetista, a fim de testar modelos e teorias cognitivas na formação de novas arquiteturas. O CST foi projetado para impor restrições mínimas ao usuário (basicamente necessita-se seguir um pequeno conjunto de conceitos simples), que discutiremos brevemente nesta seção.

A Figura 22 ilustra os principais conceitos do *Core* do CST, que permeiam todo o desenvolvimento do *toolkit*. Esse *Core* corresponde a um conjunto de princípios arquiteturais, utilizados de maneira generalizada em qualquer Arquitetura Cognitiva desenvolvida com o auxílio do CST. De uma certa maneira, atender essas demandas do *Core* são a única restrição imposta em modelos construídos com o CST, sendo assim os blocos básicos a partir dos quais estruturas mais complexas podem ser construídas utilizando o *toolkit*.

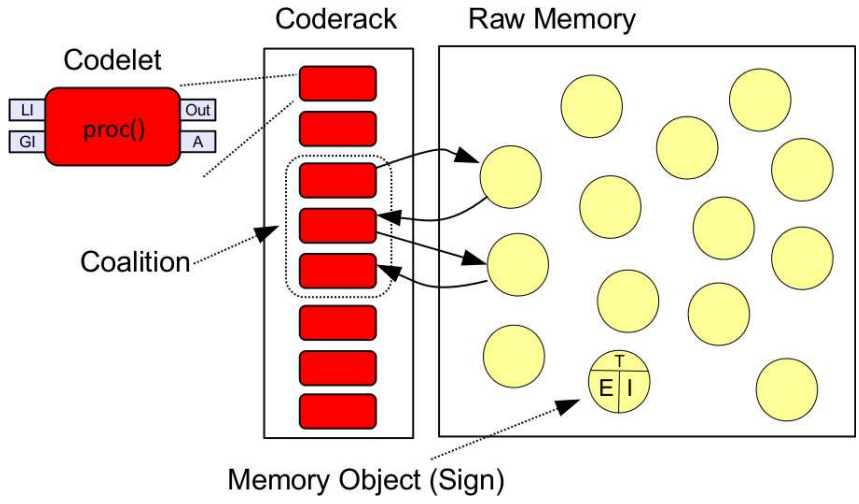


Figura 22 – O Core do CST — adaptado de Paraense *et al.* (2016)

Os dois conceitos centrais do *Core* do CST são Codelets e Memórias. Codelets, como originalmente definidos por Hofstadter e Mitchell (1994) em sua arquitetura *Copycat* e posteriormente usados por Franklin *et al.* (1998) na arquitetura LIDA (FRANKLIN *et al.*, 2014), são micro-agentes, pedaços de processos não bloqueantes paralelos e assíncronos que, em conjunto com as Memórias, são responsáveis pela construção de todos os processos cognitivos, ou funções, em uma Arquitetura Cognitiva. Em termos materiais, um Codelet é um trecho de código que executa de maneira contínua, cíclica e em sua própria *thread*, sendo responsável por uma parte de um sistema que opera em paralelo.

Conforme mencionado anteriormente, um dos princípios fundamentais do *core* do CST é que qualquer arquitetura cognitiva construída utilizando-se o CST deve ser orientada a Codelets, uma vez que todas as funções ou capacidades cognitivas devem ser implementadas na forma de Codelets ou grupos de Codelets interagindo entre si. Isso significa que de um ponto de vista conceitual, qualquer sistema implementado com o CST é conceitualmente um sistema paralelo e assíncrono, onde cada componente é modelado por um Codelet. Os Codelets do CST são implementados de maneira similar aos da Arquitetura Cognitiva LIDA, sendo comparáveis aos processadores de propósito específico descritos na Teoria de Workspace Global de Baars (BAARS, 1988; BAARS; FRANKLIN, 2007).

O segundo conceito central na infraestrutura do CST é a noção de Memória. Uma Memória é uma estrutura genérica capaz de armazenar informações, de maneira geral, de qualquer tipo de dados, que pode ser utilizado como entrada ou saída por Codelets. Codelets e Memórias podem interagir, criando Coalizões, que podem ser estáticas ou dinâmicas, sendo usadas para implementar as funções cognitivas atribuídas à arquitetura como, por exemplo, Percepção ou Comportamento. A Figura 22 ilustra os conceitos de Codelet, Memória e Coalizão.

Além de apresentar o toolkit CST, Paraense *et al.* (2016) também descrevem uma Arquitetura de Referência, para a construção de arquiteturas cognitivas genéricas, chamada de Arquitetura de Referência CST. Nessa arquitetura, os Codelets são organizados em grupos, onde cada grupo organiza os *codelets* responsáveis por implementar um modelo cognitivo de uma função cognitiva determinada. A Figura 23 ilustra a referida arquitetura.

O CST foi usado para implementar diferentes arquiteturas cognitivas como, por exemplo, em uma aplicação de consciência de máquina, usada para controle de tráfego (PARAENSE, 2016), na construção da Arquitetura Cognitiva MECA (GUDWIN *et al.*, 2017) e ao usar MECA para controle de tráfego (GUDWIN *et al.*, 2018) e robótica de transporte (GUDWIN *et al.*, 2020).

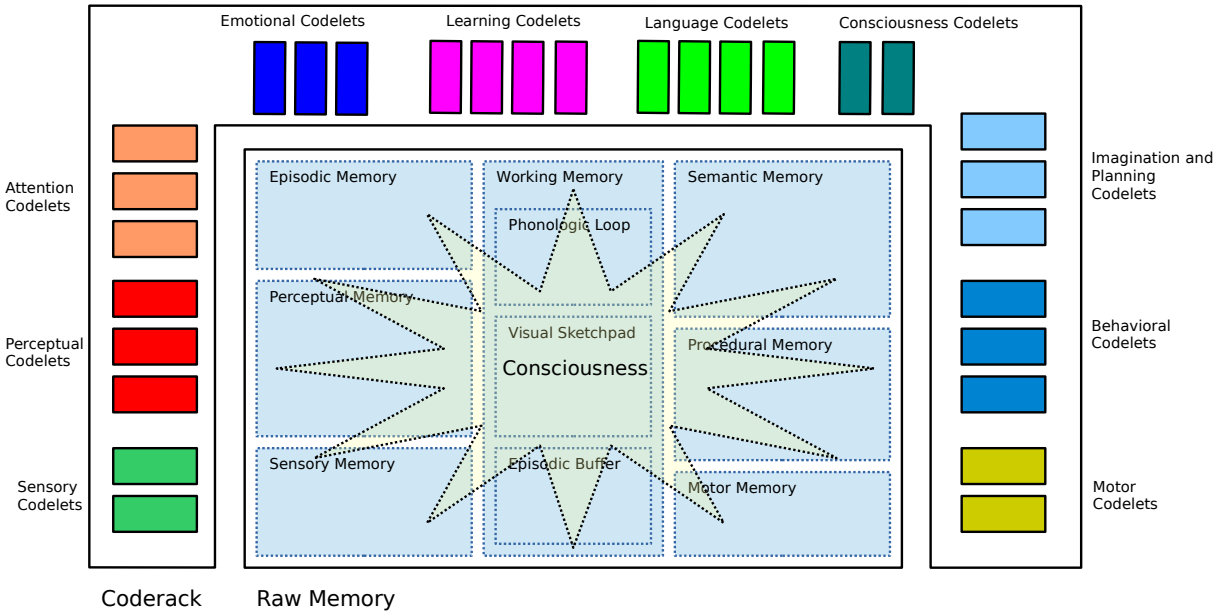


Figura 23 – Visão Geral da arquitetura do CST — adaptado de Paraense *et al.* (2016)

4.2 Uma Implementação Distribuída

O CST foi originalmente concebido para a construção de Arquiteturas Cognitivas implementadas em Java, executando sobre dispositivo único. Embora isso possa ser aceitável para uma grande variedade de Sistemas Cognitivos, há situações em que isso pode ser muito restritivo. Este é o caso se o Sistema Cognitivo pretendido se encaixar em uma das seguintes situações:

- Uma Arquitetura Cognitiva destinada a ser distribuída entre vários dispositivos;
- Sem suporte Java disponível;
- Necessitar do uso de alguma ferramenta específica não disponível (ou com limitações) em Java, como por exemplo, as mais modernas bibliotecas de Redes Neurais e *Deep Learning*.

Enquanto para a primeira situação, uma versão aprimorada do CST pode contornar o problema, a segunda pode ser um problema, especialmente ao lidar com sistemas com baixo poder computacional, como micro-controladores, onde pode ocorrer de nenhuma JVM (*Java Virtual Machine*) estar disponível. Além disso, a exigência de Java pode trazer dificuldades adicionais para integração com tecnologias de ponta como *Deep Learning*, onde apesar de existirem interfaces Java, estas geralmente não incluem todas as funcionalidades fornecidas para outras linguagens, como, por exemplo, Python.

Tais questões nos levaram à proposição de uma ferramenta, funcionalmente similar ao CST, mas distribuída e multi-linguagem: o DCT, que foi desenvolvida como parte

deste trabalho. O DCT, acrônimo para *Distributed Cognitive Toolkit*, é um toolkit básico para ajudar no desenvolvimento de Sistemas Cognitivos de forma distribuída e independente de linguagem (GIBAUT; GUDWIN, 2020). Um Agente Cognitivo criado com DCT deve conseguir executar em múltiplos dispositivos físicos (desktops, Raspberries ou Arduínos) ou virtuais (como contêineres Docker). É uma reimplementação das ideias vistas pela primeira vez no documento principal do já citado CST (PARAENSE *et al.*, 2016), focada em aproveitar ao máximo as vantagens de ser verdadeiramente distribuída e dos possíveis benefícios que se pode ter no uso de cada linguagem de programação. Como esperado, também segue algumas linhas teóricas, como ser orientada a Codelets, de modo análogo ao CST.

O DCT concebe um Sistema Cognitivo distribuído como um sistema multiagentes, onde um protocolo padrão é utilizado para a comunicação entre esses microagentes. Seguindo as ideias do CST, cada agente em tal sistema é construído usando Codelets e Memórias. Muitas Arquiteturas Cognitivas como, por exemplo, MECA (GUDWIN *et al.*, 2017), LIDA (FRANKLIN *et al.*, 2014) e outras podem ser vistas desta maneira. É de grande importância notar que a estrutura e as funcionalidades de um sistema multiagentes são altamente compatíveis com conceitos de Computação Distribuída e podem ser confortavelmente mapeados para a Internet das Coisas: Codelets Sensoriais podem ser simplesmente dispositivos sensores do mundo real e Codelets Motores podem ser simplesmente relés ou atuadores, enquanto Codelets mais complexos podem ser incorporados em microcontroladores ou até mesmo contêineres de software na nuvem. Note ainda que, dependendo da maneira como for implementado, um Sistema Cognitivo construído com o DCT pode também se adequar aos conceitos de Sistemas de Sistemas e Sistemas Ciber-Físicos.

O primeiro protótipo do DCT — e o software atual baseado nele — foi escrito em *Shell script* e *Python*, e pode ser implantado em contêineres (como Docker) ou em diferentes dispositivos, incluindo Raspberries e Arduinos. A Figura 24 ilustra a ideia: se as condições de entrada/saída forem satisfeitas, não há necessidade de executar todo o sistema em um único computador.

4.2.1 Visão Geral da Arquitetura do DCT

Do ponto de vista arquitetônico, o DCT é composto por um conjunto de *Nodes* que se comunicam entre si e integram, na totalidade, um sistema funcional com capacidades cognitivas. Teoricamente, diferentes subconjuntos da mesma coleção de *Nodes* podem até atuar como sistemas diferentes. Aqui, o termo *Node* representa uma entidade (lógica ou física), que funciona como armazenamento para grupos de Codelets e/ou Memórias e é responsável pelo seu funcionamento e ciclos de vida. A Figura 24 ilustra de maneira informal um conjunto de *Nodes* compondo uma arquitetura cognitiva e dando uma ideia

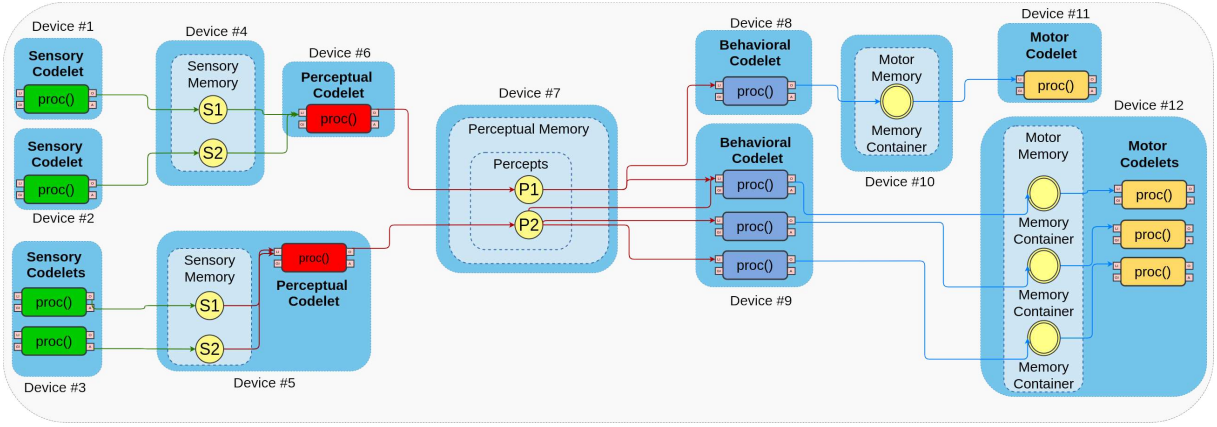


Figura 24 – Ilustração de um sistema multi-dispositivo orientado a Codelets como visto em Gibaut e Gudwin (2020). Observe que, dependendo da capacidade computacional do dispositivo, ele pode executar um único Codelet ou vários

dessa não-homogeneidade nas configurações dos dispositivos. Na subseção 4.2.4 desenvolveremos uma definição mais completa para o conceito de *Node*.

Para se comunicarem, os *Nodes* seguem um protocolo bem definido, que regula os possíveis modos de interação entre *Nodes*. Seguindo os princípios básicos do CST, Codelets só podem interagir diretamente com Memórias, ou seja, um Codelet representa um bloco de unidade computacional, aplicando algum processo nos dados, mas todos os dados devem vir de Memórias, e da mesma forma novos dados gerados devem ser armazenados em Memórias. Apesar de Codelets poderem, provisoriamente, criar estruturas de dados necessárias para seus cálculos, eles não devem armazenar esses dados em si próprios. Devem transferir esses dados para Memórias. Assim, todo armazenamento de dados deve ser realizado por Memórias, que podem ser de diferentes tecnologias. Para esta comunicação, a DCT utiliza, canonicamente, mensagens formatadas em *JSON*. Isso permite o uso de uma boa variedade de tecnologias. A implementação de referência desenvolvida nesse trabalho inclui comunicação por meio de *sockets* e bancos de dados simples como *MongoDB* e *Redis*. Posteriormente, novos mecanismos podem ser acrescentados. Seguindo essas diretivas, um usuário do DCT pode optar por qualquer linguagem ou tecnologia disponível no dispositivo onde se deseja que o Node opere.

Formalmente, podemos conceber um Sistema Cognitivo, sob a perspectiva do DCT, da seguinte forma:

Definição 4.1 *Sistema Cognitivo DCT*

Seja N um conjunto de Nodes*, onde um* Node *é uma entidade (lógica ou física) que encapsula um ou mais Codelets e/ou* Memories *destinados a serem executados sob a supervisão de um único* Node Master[1] *em um sistema operacional.*

[1] Vide seção 4.2.4 para uma definição do que é um Node Master

Para cada Node*, existe uma* Interface $I = \{M, S\}$*, onde* M *é um subconjunto de* Memories *implementado em um* Node*, que serão acessados de outros* Nodes *externos e* S *é um servidor que escuta um URI (Uniform Resource Identifier). Este servidor* S *deve escutar as requisições e responder com mensagens formatadas* JSON*.*

Um Sistema Cognitivo criado com DCT é definido pela interação entre os elementos de N *seguindo algum Modelo Cognitivo implementado por meio de Codelets (e.g. MECA ou LIDA).*

A estrutura das mensagens JSON utilizadas como resposta do servidor estão detalhadas no Apêndice A, na subseção A.3.3.

4.2.2 Estrutura de um Codelet DCT

De uma forma bastante genérica e abrangente, um Codelet DCT é composto, fundamentalmente, por um arquivo de código executável[2], e seus arquivos de configuração. O arquivo executável pode ser construído, em princípio, em qualquer linguagem de programação, desde que siga as diretrizes especificadas mais adiante. Nos exemplos desenvolvidos neste trabalho, os Codelets são implementados na linguagem *Python*. Os arquivos de configuração fornecem informações adicionais para a execução do arquivo de código. A Figura 25 ilustra esta estrutura.

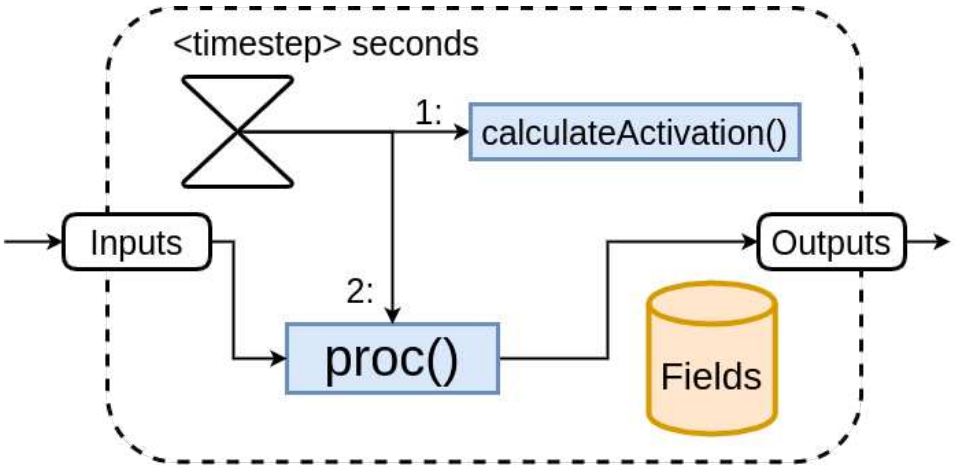


Figura 25 – O conceito de um Codelet DCT. Extraído de Gibaut e Gudwin (2020)

De modo mais específico, os arquivos que caracterizam um Codelet são:

[2] Ou seja, que pode ser carregado e executado, quando necessário. No caso do uso de linguagens compiladas, isso não inclui um arquivo de código fonte. No caso de linguagens interpretadas, em que o próprio código fonte pode ser executado pelo interpretador, entendemos que um arquivo de código fonte pode ser visto como um código executável. Na sequência nos referiremos simplesmente a um arquivo executável

- Um arquivo executável com o código do `Codelet`, definindo necessariamente as funções `calculateActivation()` e `proc()`[3]. Esse arquivo pode ser um script ou programa compilado, que deve permanecer rodando até ser comandado (pelo *Node Master*) a parar. Onde este software executará depende inteiramente da entidade que o comportará: diretamente no hardware ou escalonado por outra aplicação, como no caso do Docker;

- Um arquivo de configuração do Codelet (*fields.json*). Este arquivo contém algumas informações sobre o comportamento do Codelet, como suas entradas e deve ser possível alterá-lo dinamicamente;

- Arquivos de configuração e/ou inicialização adicionais (e.g. arquivos *.ini*), que possam ser necessários por um motivo específico do problema.

Cada Codelet deve estar em um diretório próprio, contendo todos os arquivos que o compõem. Isso evita conflitos de nomenclatura entre arquivos correspondentes a Codelets diferentes.

Seguindo convenções similares às do CST, o programa com o código executável do Codelet deve ter duas funções essenciais: uma delas, com o nome `calculateActivation` e outra com o nome `proc`. A primeira é usada para calcular o nível de ativação do Codelet, que tipicamente expressa a relevância atual da informação disponibilizada pelo Codelet. Esse nível de ativação pode ser usado de diferentes maneiras, dentro da arquitetura cognitiva sendo construída, mas usualmente para ser comparado com um valor de limiar, que pode ser utilizado para decidir se a função principal (`proc`) deve ou não ser executada no instante atual. Na implementação de referência em Python, a classe Codelet possui uma implementação padrão para `calculateActivation` que pode ser sobreposta por uma versão própria do usuário, caso assim se deseje. A segunda função (`proc`) executa a funcionalidade principal do Codelet e é a função mais importante a ser definida pelo usuário. Esta função representa o código de procedimento que será chamado periodicamente, a cada passo de tempo. Essa função deve ser não bloqueante, o que significa que deve ser possível executar outros processos ao lado de um Codelet. Note que, como o `proc` está relacionado à utilidade principal do Codelet, o programador/usuário deve necessariamente escrever esse código para cada tipo diferente de Codelet, pois a versão canônica da função é meramente uma rotina vazia.

O arquivo de configuração (*fields.json*) contém uma estrutura que define parâmetros importantes no início da execução de um Codelet. Note que fontes externas podem alterar este arquivo e o Codelet pode ser programado para atualizar periodicamente suas

[3] A definição do comportamento dessas funções é apresentada na sequência, no texto dos próximos parágrafos

informações ao ler este arquivo novamente. A estrutura interna do arquivo de configuração está descrita em detalhes no apêndice A na seção A.3.1.

O funcionamento básico de um Codelet se dá da seguinte forma: a cada passo de tempo (definido no arquivo *fields.json*) o método/função `calculateActivation` é chamado e executado, definindo se o Codelet deve ser ativado ou não. Caso o nível de ativação seja maior ou igual ao *threshold* (também definido no arquivo *fields.json*), o método/função `proc` é executado, seguindo as instruções implementadas pelo usuário que, de maneira padronizada, envolve capturar dados de *inputs* (se houver), processá-los e disponibilizá-los em *outputs* (se for o caso) sendo, tanto *inputs* como *outputs*, estruturas de Memória. Em casos especiais de Codelets sensores, podem não haver *inputs* e em Codelets atuadores, podem não haver *outputs* na forma de Memórias, pois nesses casos os Codelets estão interagindo com entidades externas ou diretamente com dispositivos sensores e atuadores.

4.2.3 A Estrutura da Memória DCT

A outra estrutura fundamental, a Memória, é um termo genérico para a estrutura de dados que contém as informações que Codelets consomem e/ou escrevem. Além disso, contém meta-informações, como por exemplo, seu nome, URL, tipo, *timestamp* e um valor real (que expressa uma possível avaliação do uso da informação), que podem ser utilizadas de modo auxiliar no funcionamento do sistema. Como dito anteriormente, esta informação é padronizada como uma estrutura *JSON*. A estrutura interna de uma Memória que atenda as especificações do DCT encontra-se descrita no apêndice A em A.3.3.

A atual implementação do DCT suporta Memórias implementadas em bancos de dados *MongoDB* e *Redis* bem como escrita direta em arquivos JSON.

4.2.4 Nó DCT

Na filosofia do DCT, uma arquitetura cognitiva pode ser distribuída em diferentes máquinas, executando em paralelo. Nesse contexto, um *Node* é uma abstração para um subsistema virtual que contém um número arbitrário de Codelets e/ou Memórias, que integram parte desta arquitetura cognitiva. A escolha de quais Codelets e quais Memórias devem fazer parte do *Node* depende tipicamente da capacidade computacional das máquinas envolvidas, que tanto podem ser computadores de maior porte, como microcontroladores com pouca memória e pouco poder computacional. Tipicamente, espera-se que cada dispositivo físico (computador, *smartphone* ou dispositivo IoT) execute um único *Node*. Entretanto, nada impede que mais de um *Node* sejam executados em um mesmo equipamento. Um *Node* nada mais é do que um subsistema que é desenhado para funcionar como um componente de um sistema maior. Cada *Node* é gerenciado e supervisionado por um *Node Master*, um processo exclusivo e único para cada *Node*, e que é responsável por

carregar o código necessário e manter o *Node* operacional. Diferentes instâncias do *Node Master*, com diferentes arquivos de configuração podem ser executados em uma mesma máquina, sendo que cada instância será responsável por gerenciar um *Node* diferente. O *Node Master* é responsável por iniciar, finalizar, adicionar e remover Codelets e/ou Memórias, que estão executando sob sua supervisão. Além disso, deve verificar periodicamente a integridade de seu sistema, reexecutando processos inativos e ouvir solicitações externas, como solicitações de informações ou até mesmo solicitações de desligamento.

Para compor uma arquitetura cognitiva, os *Nodes* precisam se comunicar entre si, por meio da interligação entre seus Codelets e Memórias. Da mesma forma, internamente ao *Node*, Codelets devem se interconectar com Memórias, segundo as imposições da arquitetura cognitiva. A topologia dessa interconexão é definida em um arquivo de configurações, utilizado pelo *Node Master* para gerenciar o *Node*. Esse arquivo de configurações tem o nome `param.ini` e sua estrutura interna está descrita no apêndice A seção A.3.2.

O *Node Master* possui ainda uma *Interface* para comunicação externa, por meio da qual fontes externas conseguem se comunicar com os Codelets e Memórias e verificar seu estado operacional. Para isso, essa *Interface* implementa um pequeno servidor por meio de *sockets* abertos.

4.2.5 Autorregeneração (*Self-Healing*)

Entre as muitas possibilidades permitidas por um sistema distribuído, uma capacidade que é muito importante para garantir sua integridade operacional é a autorregeneração. O DCT, em sua versão atual, apresenta algumas habilidades de autorregeneração. Sendo um projeto em desenvolvimento, alguns recursos adicionais estão previstos para ser incorporados em versões futuras.

Existem quatro tipos de falhas que um sistema cognitivo construído com DCT pode estar sujeito:

- Um Codelet ou Memória morre devido a alguma situação anormal que leva a uma falha;
- Um Codelet destinado a parar é reiniciado erroneamente;
- Um Codelet ou Memória é atacado e fica inativo;
- Um *Node* inteiro é atacado e fica inacessível por outros *Nodes*.

O DCT, em sua versão atual, pode lidar com as três primeiras situações mostradas acima. A quarta situação ainda não pode ser contornada, mas já se prevê diferentes

maneiras pelas quais pretende-se contornar essa situação, em uma versão futura do sistema. Para garantir o atendimento às três primeiras situações, o *Node Master* verifica periodicamente a integridade de seu processo em execução. Para isso, o *Node Master* tem acesso ao arquivo *param.ini* com uma lista de todos os Codelets e *Memories* contidos no dispositivo e uma lista de tudo que deve ser executado. Se acontecer de algo que deveria estar em execução não estiver neste estado, ou algo que deveria ter sido eliminado ainda estiver em execução, o *Node Master* deve corrigir o erro executando novamente ou matando o processo apropriado.

Para evitar um "desastre de autorregeneração" (algo semelhante a um "câncer") onde o sistema tornar-se-ia impossível de ser encerrado de maneira legítima, existem dois mecanismos implementados. Um sinal — chamado de "aviso de suicídio" — pode ser enviado a um *Node* por qualquer entidade (geralmente outro *Node* ou um usuário) e, assim que este sinal é recebido, o respectivo *Node Master* deve encerrar todos os seus processos e encerrar a si mesmo. Alternativamente, um limite pode ser estabelecido ("death_threshold") e um sistema de votação pode ser usado, onde o *Node Master* só encerra suas atividades se um certo número de votos for alcançado.

Uma solução para o quarto problema ainda está em desenvolvimento, mas a ideia central é capacitar *Nodes* específicos com poder de gerenciamento para adicionar ou remover outros *Nodes* da rede. Esses *Nodes* gerenciadores, usando informações sobre a taxa esperada de atualização de informações das estruturas de todos os *Nodes* sob sua supervisão e do tempo decorrido desde o último dado recebido, podem disparar sub-rotinas de recuperação. Essas sub-rotinas podem reiniciar completamente o *Node* na própria máquina original onde houve a falha, se isso for possível, ou redistribuir a carga da tarefa, reiniciando o *Node* em outra máquina, caso a máquina afetada esteja inoperante.

Observe que esse recurso de autorregeneração não é obrigatório para todos os sistemas criados com DCT.

4.2.6 Suporte Padrão DCT em Python

Até o momento, o DCT foi implementado para a linguagem de programação Python, e no futuro, serão desenvolvidas implementações do DCT para outras linguagens de programação, como Java, C, C++, C#, PHP, Ruby, Perl, JavaScript, etc. As funções implementadas no pacote DCT em Python estão listadas no apêndice A, seção A.4.1.

Todo *Node* DCT deve implementar um servidor que atenda requisições REST para comunicar-se com os demais elementos dos sistemas e subsistemas aos quais está inserido. A implementação atual, em Python, utiliza a biblioteca Flask. Esse servidor é responsável por receber as requisições do cliente e retornar as informações solicitadas. As requisições implementadas encontram-se descritas no apêndice A, seção A.4.2.

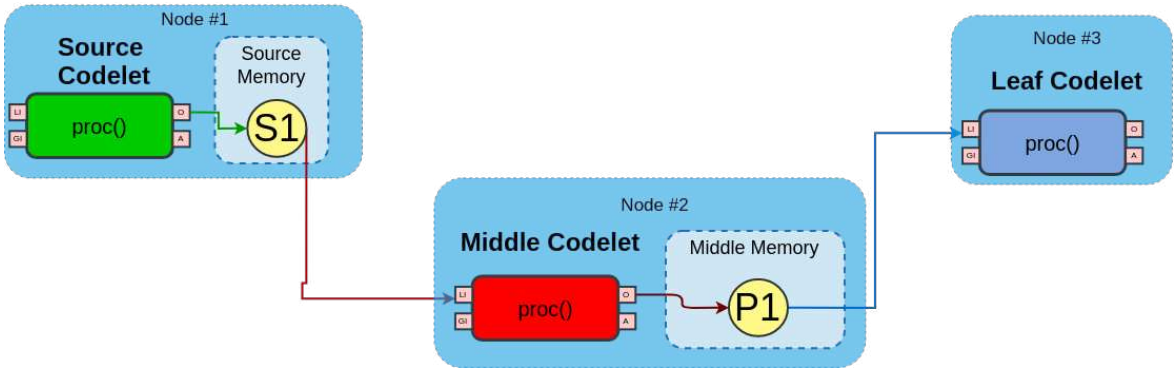


Figura 26 – Estruturas dos *Nodes* utilizadas da aplicação de escalabilidade e exemplos de conexão. Note que as Memórias foram arbitrariamente definidas como saída

Uma descrição passo-a-passo de como criar e executar um aplicativo DCT pode ser encontrada no Apêndice A, na seção A.1.

4.2.7 Escalabilidade de Aplicações com o DCT

Para demonstrar a escalabilidade dos sistemas construídos com o DCT, foi desenvolvido um exemplo de aplicação, como prova de conceito, constituído por um grande número de *Nodes* simplificados, apenas com o intuito de estudar o processo de carregamento, execução e integração entre *Nodes*, de maneira incremental (GIBAUT; GUDWIN, 2020), em uma arquitetura distribuída. Nesse exemplo, uma arquitetura de topologia arbitrária que executa *Nodes* em contêineres Docker distribuída em uma rede local é progressivamente ampliada com mais *Nodes* de modo automático durante a execução do sistema.

Nesta aplicação, cada *Node* possuía apenas um Codelet e uma Memória. Usamos três tipos de Codelets fictícios:

- **Source Codelets** são Codelets que geram informações, emulando sensores. Aqui, cada um deles gerou um sinal senoidal contínuo e escreveu nas respectivas *Memórias* de saída;
- **Leaf Codelets** são os elementos finais de uma cadeia de informações, podendo representar atuadores, cuja finalidade é receber um comando e interagir com o ambiente. No exemplo, eles apenas imprimem as informações recebidas;
- **Middle Codelets** estão entre os dois acima e poderiam representar um processo computacional mais complexo. No exemplo apresentado, eles apenas copiam informações das entradas para as saídas.

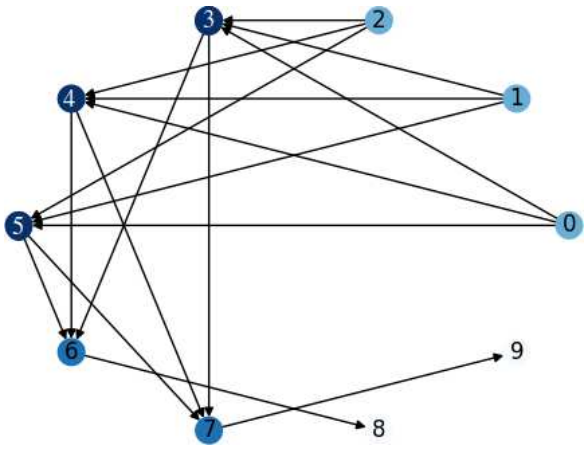
Todos os Codelets rodavam em contêineres Docker e se comunicavam através do protocolo *TCP* e arquivos *JSON* como Memórias. A Figura 26 ilustra as estruturas e um

exemplo de conexão entre elas.

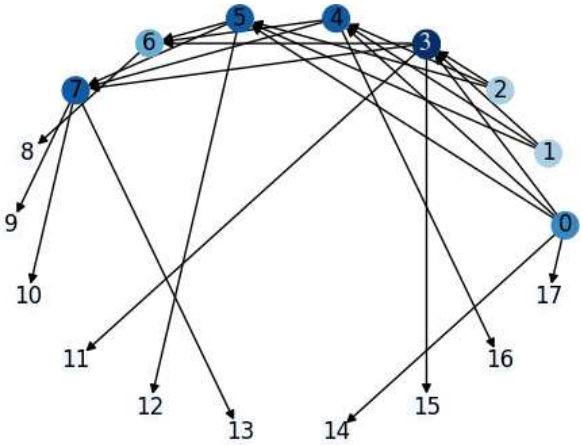
Inicialmente foram executados os dez primeiros *Nodes* simultaneamente, formando um "núcleo" ao qual outros *Nodes* foram posteriormente adicionados. Em seguida, no mesmo computador, outros oito *Nodes* foram executados, conectando-se aos elementos já existentes de forma aleatória, com distribuição uniforme, tomando cuidado em estabelecer ligações apenas entre **Source Codelets** e **Middle Codelets** ou entre **Middle Codelets** e **Leaf Codelets**. Na sequência, outros cinco *Nodes* foram executados em um segundo computador e, finalmente, mais dez *Nodes* executados em um terceiro computador, sempre de forma aleatória e distribuição uniforme.

A Figura 27 ilustra o processo: cada uma das quatro sub-figuras mostra um estado do experimento apresentado em (GIBAUT; GUDWIN, 2020). Esses gráficos são direcionados, onde a borda mostra a direção do fluxo de informações (ou seja: o *Node* que escreve aponta para o que lê). Além disso, a cor dos nós no gráfico reflete o número de conexões (quanto mais conexões, mais escura é a matiz do tom de azul).

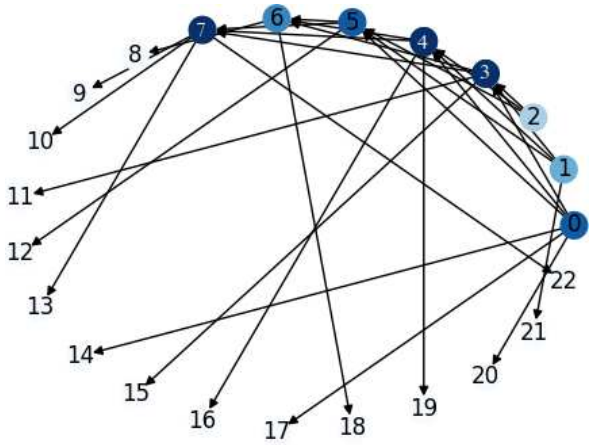
Esta aplicação exemplo mostrou que o gargalo da comunicação entre *Nodes* é definido pelo tempo de serialização do módulo *json* do *Python* e pelo método de armazenamento em Memória escolhido (arquivo plano *JSON*, neste caso), implicando em latência na leitura/gravação do banco de dados ou a latência da rede. No artigo em questão o intervalo de tempo mínimo sustentado pelo sistema sem apresentar problemas foi de cerca de 500ms, porém desde então foram realizadas melhorias no software que diminuem esse valor.



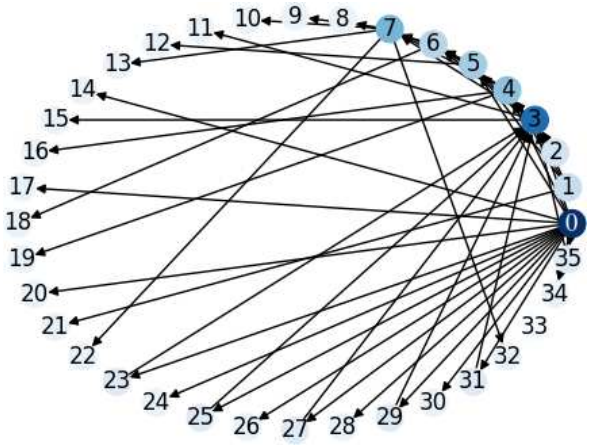
(a) Sistema-exemplo em seu estado inicial, executando em um único computador



(b) Sistema-exemplo após a primeira adição de novos *Nodes*



(c) Sistema-exemplo executando em dois computadores, após nova adição de *Nodes*



(d) Configuração final do Sistema-exemplo, executando ao longo de três computadores

Figura 27 – Sistema-exemplo *dummy* crescendo progressivamente com sucessivas adições de novos *Nodes* em tempo de execução ao longo de diferentes dispositivos. Retirado de Gibaut e Gudwin (2020)

4.3 Resumo do Capítulo

Este capítulo foi dedicado à apresentação e análise detalhada do *Distributed Cognitive Toolkit* (DCT), uma ferramenta desenvolvida para facilitar a construção de Arquiteturas Cognitivas Distribuídas. Este capítulo descreve os aspectos técnicos, a implementação distribuída do DCT e suas capacidades, incluindo auto-regeneração e suporte padrão em Python. O objetivo principal do DCT é fornecer um framework de desenvolvimento flexível e robusto para pesquisadores e desenvolvedores interessados em Sistemas Cognitivos e suas aplicações.

Inicialmente, o capítulo introduz o conceito do *Cognitive Systems Toolkit* (CST) e sua arquitetura de referência, destacando a importância de um *framework* padronizado para o desenvolvimento de sistemas cognitivos. O CST serve como a base teórica e con-

ceitual sobre a qual o DCT é construído, incorporando práticas de design modulares e escaláveis.

O capítulo ainda detalha a estrutura arquitetônica do DCT, incluindo os principais componentes, como Codelets, a estrutura da Memória, *Nodes*, e as capacidades de autorregeneração. Cada componente é explicado em termos de sua função dentro do sistema, seu papel na promoção da distribuição e descentralização do processamento cognitivo, e sua contribuição para a resiliência e eficiência do sistema como um todo.

O capítulo também aborda o suporte padrão do DCT em Python, enfatizando a acessibilidade e a facilidade de uso da ferramenta. A escolha do Python como linguagem de implementação alinha-se com sua popularidade na comunidade científica e sua rica biblioteca de pacotes para computação científica e Inteligência Artificial.

Finalmente, o capítulo conclui com uma discussão sobre a escalabilidade de aplicações usando o DCT, demonstrando como a arquitetura distribuída e os princípios de design modular facilitam a expansão e adaptação de sistemas cognitivos para atender a requisitos variados e crescentes. Este foco na escalabilidade não apenas amplia o potencial de aplicação do DCT mas também sublinha a visão do *toolkit* como uma plataforma para futuras inovações em sistemas cognitivos distribuídos e *Cognitive Twins*.

5 Especificação de um *Cognitive Twin* Evolutivo Usando Sistemas Distribuídos

5.1 Um *Cognitive Twin* Evolutivo

Depois de apresentar alguns conceitos-chave e a ferramenta para construção de arquiteturas cognitivas distribuídas desenvolvida no contexto desta tese, nosso próximo passo é discutir a técnica que desenvolvemos para construir *Cognitive Twins*, baseada em dados coletados da observação de um agente (aqui chamado de "agente primordial"), em sua interação com o ambiente. A técnica envolve a combinação, utilizando um processo evolutivo, de uma vasta quantidade de dispositivos simples, orquestrados para compor um sistema integrado, capaz de mimetizar o comportamento do agente observado, construindo dessa forma um *Cognitive Twin* do agente primordial. Na filosofia original do CST, as partes funcionais de uma arquitetura cognitiva são os Codelets, que podem ser agrupados de acordo com suas funções cognitivas. No DCT, onde tanto Codelets quanto Memórias podem estar distribuídos em *Nodes*, nos referiremos a *Nodes* que assumirão funções cognitivas distintas. Dessa forma, podemos ter *Nodes* Sensoriais, *Nodes* Perceptuais, *Nodes* Comportamentais e *Nodes* Motores, de acordo com as funções cognitivas de sensoreamento, percepção, geração de comportamento ou atuação motora. Apesar de ser possível imaginarmos *Nodes* responsáveis por outras funções cognitivas, na técnica aqui desenvolvida nos limitaremos somente a essas 4 funções cognitivas.

Sensores e atuadores podem ser vistos como *Nodes* Sensoriais e Motores, ou seja, dispositivos simples que realizam uma tarefa muito específica. A complexidade do processo de mimetização do comportamento do agente primordial está na descoberta de quais são as informações sensoriais relevantes para cada tomada de decisão, e seu uso na determinação de ações correlatas às do agente primordial em sua interação com o ambiente. Seguindo essa perspectiva, postulamos que a conexão entre sensores e atuadores é gerada por combinações evolutivas de elementos que se agrupam para gerar informação útil e elementos que usam essa informação para controlar os atuadores. A informação útil é gerada por meio de *Nodes* Perceptuais, que de maneira gradativa constroem abstrações de mais alto nível, que podem ser posteriormente utilizadas por *Nodes* Comportamentais, para gerar dados motores. Embora seja relativamente simples, esse ciclo cognitivo cuja informação segue sequencialmente e de maneira assíncrona por sensores, perceptos, comportamentos e ações motoras encontra fundamento em processos cognitivos do tipo 1, em teorias de Processos Duais (OSMAN, 2004), sendo também semelhante ao *System 1* encontrado na arquitetura cognitiva MECA (GUDWIN *et al.*, 2017). Esse tipo de sis-

tema está associado à parte da cognição responsável por processos de "baixo nível", mais ligados a reações instintivas, emoções, necessidades mais básicas e outros processos que não exigem planejamento ou capacidades mais complexas.

De acordo com as teorias de Processos Duais, o Sistema 1 é caracterizado como o modo de pensamento rápido, intuitivo e automático. Este sistema opera sem esforço consciente, é responsável por nossas reações e impressões iniciais, e pode ser influenciado por viéses e heurísticas. É o sistema que está em ação quando você realiza uma tarefa simples que não requer muita concentração ou atenção consciente, como reconhecer o rosto de um conhecido em uma multidão ou responder a uma simples questão de matemática como 2 + 2, que tem seu resultado decorado. O Sistema 1 é essencial para a nossa sobrevivência diária, permitindo-nos realizar múltiplas tarefas simultaneamente sem sobrecarregar nossa capacidade de processamento consciente. Ele nos ajuda a tomar decisões rápidas em situações que exigem respostas imediatas ou em ambientes familiares onde as heurísticas podem ser aplicadas com sucesso.

Apesar de não ser utilizado neste trabalho, é importante apresentar também o conceito de Sistema 2. O Sistema 2 é caracterizado por requerer atenção consciente e esforço deliberativo para operar e baseia-se na lógica e na razão para analisar informações e tomar decisões. Pode ainda ser controlado voluntariamente, permitindo suprimir ou modificar respostas automáticas fornecidas pelo Sistema 1. No entanto, tem uma capacidade limitada, o que significa que só pode lidar com um número limitado de tarefas de cada vez, pode se tornar sobrecarregado ou fatigado.

Na construção desse modelo, temos duas restrições principais: primeiro, visamos usar dispositivos simples e de baixo poder computacional e assumimos que não temos uma quantidade infinita de dispositivos (com uma variedade infinita de respostas de entrada-saída) para buscar a melhor combinação. Abordamos essas restrições tendo dispositivos que podem aprender (ou de alguma forma se adaptar) e recorrendo a uma heurística que melhora essa busca. Também assumimos que seria impraticável ter esses dispositivos totalmente conectados, pois essa sobrecarga de comunicação pode prejudicar o desempenho e/ou ser infactível nos dispositivos físicos.

Assim, propomos encontrar uma configuração otimizada por meio de treinamento explícito utilizando métodos de Aprendizado de Máquina inseridos nos Codelets e uma Estratégia Evolutiva (CASTRO, 2006) para encontrar uma conexão adequada entre eles. Discutiremos isso em detalhes nas próximas subseções.

5.2 Especificação da Estrutura da Arquitetura Cognitiva para os *Cognitive Twins*

A arquitetura cognitiva para o *Cognitive Twin* construída por nossa metodologia envolve 4 tipos diferentes de *Nodes*, todos eles muito simples e cada um dos tipos referente a uma das quatro funções cognitivas: *Nodes* Sensoriais, *Nodes* Perceptuais, *Nodes* Comportamentais e *Nodes* Motores. Uma visão geral dos *Nodes* utilizados na construção do nosso *Cognitive Twin* pode ser vista nas Figuras 28 a 31.

Os *Nodes* Sensoriais se conectam somente aos *Nodes* Perceptuais. Da mesma forma, os *Nodes* Perceptuais têm como entradas somente *Nodes* Sensoriais, e como saídas, somente *Nodes* Comportamentais. Por fim, os *Nodes* Comportamentais, têm como entrada os *Nodes* Perceptuais e como saídas os *Nodes* Motores. A topologia de conexão entre *Nodes* Sensoriais, Perceptuais, Comportamentais e Motores é determinada por um algoritmo evolutivo que é detalhado na seção 5.3.

Como cada *Node* possui somente um único Codelet e uma única Memória, na conexão entre dois *Nodes* não é necessário indicar quais são os Codelets de um *Node* que se conectam com quais Memórias do outro *Node*, pois essa conexão é trivial.

Apesar da técnica aqui descrita poder ser implementada sobre quaisquer tipos de sistemas distribuídos, em nossos experimentos, para agilizar o processo de experimentação, utilizamos apenas dispositivos virtuais, executando *Nodes* com Codelets codificados em Python, em contêineres Docker. Isso nos permitiu manipular melhor algumas estruturas, como enviar ou solicitar dados do programa mestre de e para cada um dos Codelets e criar ou destruir esses dispositivos virtuais conforme fôssemos precisando deles. Mesmo assim, mantivemos as estruturas internas simples para fazer um bom paralelo com dispositivos de baixo poder computacional.

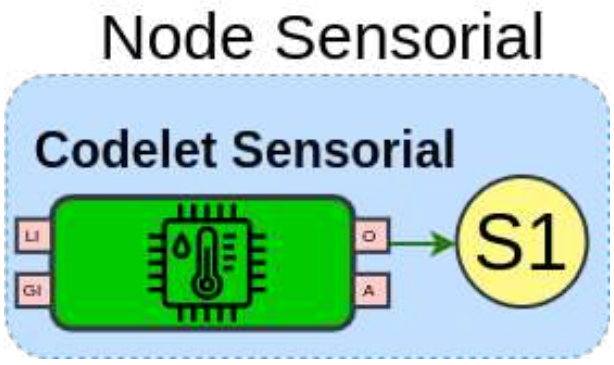


Figura 28 – *Node* Sensorial utilizado para construção do Agente Cognitivo proposto neste trabalho. Este *Node* simples apenas faz leituras do ambiente simulado em que o agente primordial está inserido

5.2.1 Nodes Sensoriais

Seguindo a teoria apresentada por Gudwin *et al.* (2017), os Codelets Sensoriais correspondem à porta de entrada de um sistema cognitivo, capturando dados de sensores reais, físicos ou virtuais. É por meio desses elementos que os dados brutos oriundos de propriedades do ambiente são introduzidos no sistema, capturados por meio de sensores.

Por exemplo, se considerarmos um olho humano como um sensor, os dados brutos seriam a luz que entra nas pupilas. Essas propriedades físicas são então traduzidas em números, que representam a intensidade relativa de atributos específicos, como luminosidade e matizes de cores de um quadro em particular.

Neste trabalho, os Codelets Sensoriais são estruturas virtuais que fazem um paralelo direto com sensores físicos simples: por meio de requisições para um servidor que representa o ambiente, tais entidades coletam informações como temperatura, umidade, luminosidade, presença etc.

É importante aqui fazer uma distinção muito importante. As informações sensoriais utilizadas pelo *Cognitive Twin* não têm como ser as mesmas do agente primordial sendo observado. Isso ocorre porque normalmente os sensores são dispositivos localizados no espaço. A menos que o próprio agente primordial seja capaz de enviar uma representação de seus dados sensoriais para o *Cognitive Twin* (o que seria possível, em tese, caso o agente primordial sendo observado fosse também um agente computacional), os dados sensoriais utilizados pelo *Cognitive Twin* não são uma representação exata dos sensoriais reais que o agente primordial utiliza em seu processo cognitivo. No caso de agentes primordiais humanos, claramente isso não é possível. Dessa forma, assumimos que a informação sensorial utilizada pelo *Cognitive Twin* é somente uma representação da informação sensorial original do agente primordial. Em alguns casos, poderá ser até uma informação sensorial diferente, de um mesmo objeto ou grandeza física, capturada de um

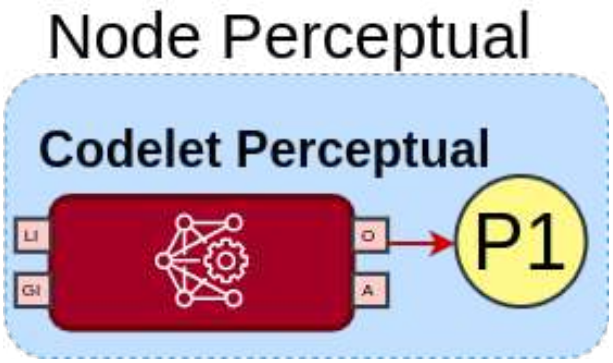


Figura 29 – *Node* Perceptual utilizado para construção do Agente Cognitivo proposto neste trabalho. Este *Node* possui como função principal uma Árvore de Decisão para classificação

outro ângulo, perspectiva ou posição.

É fundamental ainda lembrar que os dados brutos coletados pelos Codelets Sensoriais são apenas a primeira etapa no processamento de informações em um sistema. É preciso haver uma série de outras etapas envolvidas no processo de análise e interpretação desses dados, como, por exemplo: filtragem, agregação, análise de padrões, tomada de decisões, entre outras.

Os Codelets Sensoriais são peças fundamentais em um sistema, sendo responsáveis por introduzir dados brutos que serão processados e interpretados ao longo de várias etapas do processo de análise.

Os *Nodes* Sensoriais utilizados neste trabalho são dispositivos virtuais compostos por um Codelet Sensorial e uma Memória, conforme ilustrado na Figura 28.

5.2.2 Nodes Perceptuais

As estruturas subsequentes no fluxo de informações são *Nodes* Perceptuais, compostos por um Codelet Perceptual e uma Memória, conforme ilustrado na Figura 29. Esses elementos são responsáveis por reunir as informações brutas coletadas pelos *Nodes* Sensoriais e transformá-las em estruturas chamadas Perceptos (GUDWIN *et al.*, 2017). Para entender melhor, voltemos ao exemplo do olho humano, onde as saídas dos Codelets Perceptuais — escritas na Memória Perceptual, interna ao *Node* — seriam informações como *profundidade*, *objetos*, propriedades relacionais (como distância de algo) e assim por diante. É importante salientar que os *Nodes* Perceptuais são responsáveis por representar a forma como um agente experiencia o mundo, já que as informações que ele pode extrair dos dados dependem muito do funcionamento dessas estruturas.

Neste trabalho, a estrutura interna de um Codelet Perceptual é representada por um Classificador de Árvore de Decisão (KOTSIANTIS, 2013), conforme detalhado mais adiante na subseção 5.3.6, no qual as entradas são dados sensoriais e a saída é

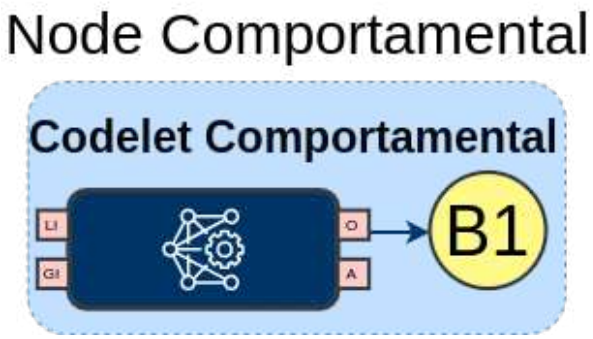


Figura 30 – *Node* Comportamental utilizado para construção do Agente Cognitivo proposto neste trabalho. Este *Node* possui como função principal uma Árvore de Decisão para classificação

um valor inteiro que representa uma ”impressão”, um *token*, um símbolo codificado que sintetiza informações acerca do conjunto de dados apresentados. Esse processo, de certa forma, representa um *encoding* dos dados brutos, possibilitando uma melhor análise e interpretação dos mesmos. É importante ressaltar que a utilização de um Classificador de Árvore de Decisão permite uma maior precisão na transformação dos dados brutos em informações mais úteis e compreensíveis, além de frequentemente apresentarem melhores resultados em dados tabulares, quando comparados a redes neurais (GRINSZTAJN *et al.*, 2022).

Em termos práticos, os *Nodes* Perceptuais funcionam como ”encoders”, mapeando combinações de valores vindos dos sensores (que compõem a entrada de cada Codelet) em números inteiros únicos.

Nodes Perceptuais possuem um papel importante no processo de tomada de decisão do agente. Isso ocorre porque as informações geradas por essas estruturas são essenciais para a compreensão do ambiente no qual o agente está inserido. A partir dessas informações, o agente pode tomar decisões mais acertadas e precisas, uma vez que ele é capaz de compreender melhor o ambiente em que está inserido.

5.2.3 Nodes Comportamentais

Os *Nodes* Comportamentais são compostos por um Codelet Comportamental e uma Memória, conforme ilustrado na Figura 30. Essas estruturas são de extrema importância, pois são responsáveis pela ativação de um ou mais protocolos que controlam o comportamento do agente com base em informações previamente estruturadas. O objetivo principal dessas estruturas é controlar o que o agente deve fazer, ou seja, controlar um ou mais *Nodes* Motores, isto é, estes *Nodes* propõem ações a serem tomadas pelo agente,

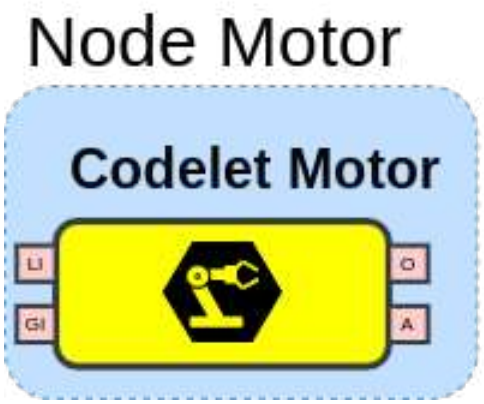


Figura 31 – *Node* Motor utilizado para construção do Agente Cognitivo proposto neste trabalho. Este *Node* possui como função principal um tratamento de regras simples e é responsável por enviar comandos diretamente para os atuadores do ambiente simulado que o agente primordial está inserido

uma vez ativados. Para isso, a Ativação é geralmente codificada em um valor real de 0 a 1 que representa uma "força do sinal", ou seja, uma forma de medir a importância desse comportamento para a situação atual. É importante observar que esse protocolo pode ser qualquer coisa, desde uma simples heurística até um método completo de Aprendizado de Máquina e a entrada do Codelet Comportamental pode incluir não apenas Percepção, mas também outras informações, como as provenientes de um subsistema Motivacional ou mesmo Emocional, seguindo a teoria apresentada em (GUDWIN *et al.*, 2017).

Vale ressaltar que um único *Node* Motor pode ter vários *Nodes* Comportamentais como entrada e, portanto, esses comportamentos competem efetivamente para prevalecer e o *Node* cujo Codelet tem maior ativação tem seus comandos aceitos. Essa competição é fundamental para garantir que o agente tome as decisões que melhor se adéquem a uma determinada situação. É possível ainda que um Codelet Comportamental transmita informações para múltiplos Codelets Motores, configurando, por exemplo, comportamentos mais complexos.

Além disso, é importante destacar que os Codelets Comportamentais podem ser programados para se adaptarem a novas situações e mudanças no ambiente. Isso significa que eles podem aprender com base em experiências anteriores e ajustar seu comportamento segundo as informações mais recentes que recebem. Esse tipo de adaptabilidade é fundamental para garantir que o agente possa se adaptar a novas situações e ambientes de maneira eficaz.

Neste trabalho, os Codelets Comportamentais também utilizam um Classificador de Árvore de Decisão como método para decidir os dados — '1' ou '0' — a enviar para seus Codelets Motores com base na Percepção do agente. Os *Nodes* Comportamentais desempenham um papel central na construção de agentes inteligentes capazes de tomar decisões e agir de maneira autônoma em ambientes complexos e em constante mudança. Eles fornecem a base para a tomada de decisões e o controle de comportamento dos agentes, permitindo que eles se adaptem e aprendam com base em experiências anteriores.

5.2.4 *Nodes* Motores

Os *Nodes* Motores são compostos unicamente por um Codelet Motor, conforme ilustrado na Figura 31. Como já mencionado anteriormente, este componente representa um paralelo direto com um atuador, que pode ser físico ou não. Ele é responsável por responder à entrada recebida e enviar um comando para a entidade correspondente em nosso ambiente virtual. É importante destacar que esse processo pode ser uma associação direta, utilizando a expressão "se isso, então" ou algum método mais sofisticado. Neste trabalho, optou-se por uma heurística que utiliza uma associação direta — como uma regra — para entrada e saída de cada Codelet. Esta escolha esta pautada em três vantagens

presentes em sistemas simbólicos que são importantes aqui:

- Requisitos Computacionais Menores: Sistemas simbólicos geralmente requerem menos recursos computacionais para serem executados em comparação com modelos de ML, que podem ser pesados e exigir capacidades de processamento e armazenamento significativas. Isso é particularmente vantajoso ao se considerar o uso de dispositivos com poder computacional limitado.
- Eficiência em Ambientes com Poucos Dados: Sistemas simbólicos não dependem de grandes volumes de dados para treinamento, ao contrário de muitos modelos de ML. Eles podem ser particularmente úteis em domínios onde os dados são escassos, caros ou difíceis de coletar.
- Determinismo e Previsibilidade: A execução de regras segue um caminho bem definido, tornando o comportamento do sistema previsível e determinístico. Isso é crucial em aplicações onde a consistência e a confiabilidade das respostas são fundamentais.

Isso conclui nossa visão geral das estruturas elementares utilizadas na arquitetura cognitiva de nosso *Cognitive Twin*. A seguir, detalharemos o processo geral de construção da arquitetura.

5.3 O Processo Geral de Construção do *Cognitive Twin*

O processo geral de construção da arquitetura para nosso *Cognitive Twin* envolve basicamente a determinação da topologia de conexão entre os diferentes tipos de *Nodes* e os ajustes das funções internas de cada Codelet visando reproduzir o comportamento do agente primordial. Esse é um processo de otimização offline de duas etapas, sendo a primeira uma otimização da topologia de conexão entre os *Nodes* e a segunda um processo de treinamento convencional dos modelos de Aprendizado de Máquina internos aos *Codelets* em cada *Node*. As conexões entre os *Nodes* serão definidas por meio de uma Estratégia Evolutiva e, dadas as configurações das conexões de cada indivíduo de uma determinada geração, métodos supervisionados de treinamento serão utilizados com o intuito de minimizar o erro entre as saídas esperadas e as obtidas.

Neste trabalho, as estruturas internas dos Codelets Sensoriais não são alteradas, simulando dispositivos muito simplistas, como um termômetro digital. No entanto, é possível que no futuro sejam feitas melhorias nos Codelets Sensoriais, para torná-los mais eficientes e precisos na coleta de dados brutos.

Nas subseções seguintes, apresentamos detalhes de cada uma destas etapas.

5.3.1 O Problema a Ser Resolvido

Antes de prosseguir, é importante relembrarmos o problema que o método apresentado propõe-se a resolver: o objetivo é copiar o comportamento aproximado do agente primordial, por exemplo, um humano ou criatura virtual. Esse comportamento copiado considera apenas a relação entre observações do ambiente onde o agente primordial se encontra (e que, supõe-se, seriam também percebidos pelo agente por meio de seus sensores) e de suas ações, ou seja, apenas observações externas a este agente, não possuindo, portanto, informações acerca de seu estado mental.

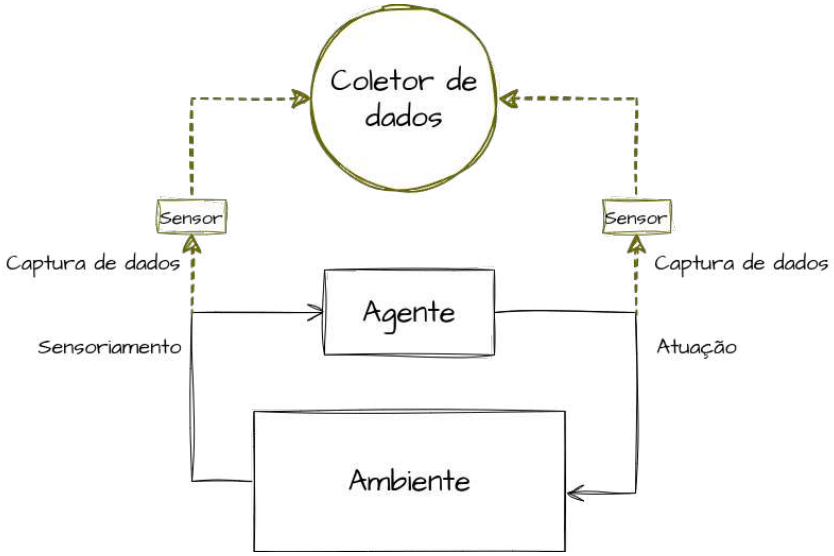


Figura 32 – Interação Agente-Ambiente com coleta de dados de interação. Um agente — por definição — sensoria e atua sobre um ambiente no qual está inserido. O Coletor captura dados do ambiente e do agente para posterior contrução do *Cognitive Twin*

A Figura 32 apresenta, além da ilustração clássica que relaciona um agente e seu ambiente, uma entidade adicional — o Coletor de Dados — que coleta observações tanto sobre o agente (e.g., sua presença em um determinado cômodo) como do ambiente (temperatura, iluminação etc), além das ações tomadas por este agente (acender uma lampada, ligar o ar-condicionado etc). A ideia central é, após a coleta completa de dados, treinar outra entidade de modo *offline* — o *Cognitive Twin* — para comportar-se como o agente primordial, isto é, frente as mesmas condições, este deve atuar de maneira similar. Ao final do treinamento o agente primordial será substituído pelo *Cognitive Twin* (Figura 33) e uma rodada de testes deve ser realizada a fim de verificar a qualidade da entidade construída.

5.3.2 Visão Geral do *Cognitive Twin*

A Figura 34 ilustra a topologia geral da arquitetura cognitiva para o *Cognitive Twin*, que visamos obter ao final do processo de construção. A arquitetura em questão

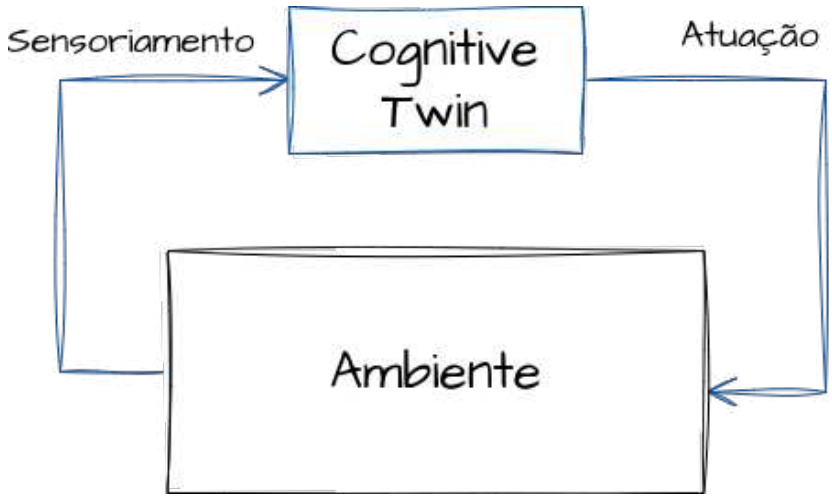


Figura 33 – Interação *Cognitive Twin* -Ambiente. Note não haver — nem deveria — diferenças em relação à interação do agente primordial e seu respectivo ambiente

é composta por 4 blocos de *Nodes* e o fluxo de informação segue sequencialmente do bloco de *Nodes* Sensoriais até o bloco de *Nodes* Motores, passando pelos blocos de *Nodes* Perceptuais e Comportamentais, respectivamente. Importante salientar que o número de *Nodes* em cada bloco depende de condições de contorno da aplicação (para os Sensoriais e Motores) e do processo de otimização (para os Perceptuais e Comportamentais).

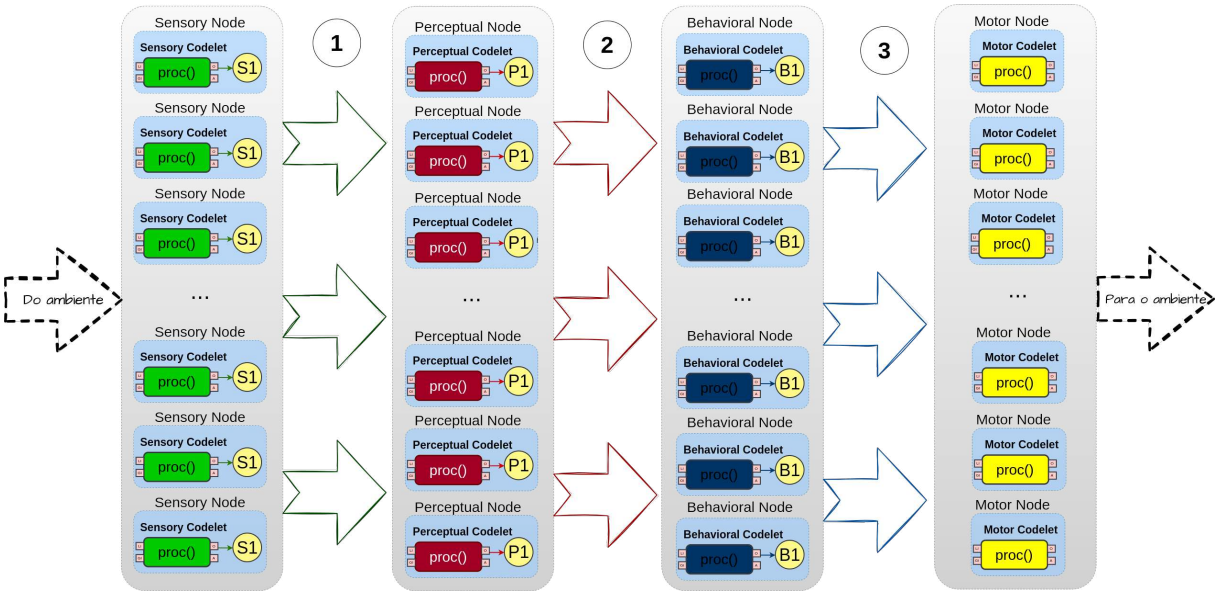


Figura 34 – Visão geral do *Cognitive Twin*. Cada um dos blocos representa um agrupamento de um tipo de *Node*

O modelo apresentado possui características próprias de uma arquitetura de Subsunção Dinâmica pois, em suas últimas etapas, o comportamento dominante é definido por um sinal *e* ("eval") dinamicamente gerado pelos próprios *Nodes* em questão, caso um *Node* Motor possua mais de um *Node* Comportamental como entrada.

Arquitetura de Subsunção é um nome genérico para uma família de arquiteturas computacionais utilizadas em controle inteligente, desenvolvida por Brooks (1986) na década de 80. Brooks propôs uma estratégia paralela onde novos comportamentos poderiam ser desenvolvidos de forma isolada e posteriormente integrados na arquitetura geral. Todos os comportamentos competiam entre si para ter acesso aos atuadores. Na Arquitetura de Subsunção clássica, os nós de supressão têm nós dominantes e não dominantes fixos. Isso significa que, uma vez que um comportamento esteja em um nível superior, ele sempre terá prioridade na configuração de seu comportamento. Podem existir situações onde a alteração desses níveis de prioridade é mais interessante do que possuir uma estrutura rígida. Para lidar com essa dificuldade, alguns autores (HAMADI *et al.*, 2010; HECKEL; YOUNGBLOOD, 2010; NAKASHIMA; NODA, 1998) propuseram um esquema de subsunção dinâmica, no qual não há entrada dominante fixa em um nó de supressão, mas essa dominância pode ser alterada dinamicamente no tempo, de acordo com situações específicas. A Figura 35 ilustra um exemplo de Arquitetura de Subsunção Dinâmica no qual o atuador recebe dos comportamentos um sinal de controle x_i e um sinal de avaliação e_i. O atuador então, seguindo um critério pré-estabelecido, seleciona o sinal de controle x_i cujo e_i melhor satisfaz as condições impostas. Esse critério pode ser desde um simples "selecionar o máximo valor" até heurísticas mais complexas.

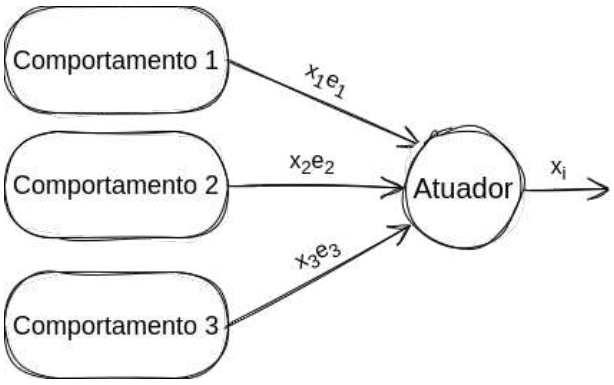


Figura 35 – Exemplo de Arquitetura de Subsunção Dinâmica. O atuador processa o sinal de controle x_i do comportamento cujo sinal de avaliação e_i atende a um critério pré-estabelecido

Ainda sobre a arquitetura do *Cognitive Twin*, faz-se necessário discorrer sobre as conexões entre os blocos e os tipos de dados que transitam por elas. Todas as conexões presentes entre os *Nodes* da arquitetura são definidas por heurísticas vindas de um "Controle de Processo", responsável pelo processo de otimização, que definiremos na subseção seguinte. As conexões entre o primeiro e o segundo bloco — marcada na Figura 34 com o rótulo numérico "1" — são as conexões entre os *Nodes* Sensoriais e os *Nodes* Perceptuais. Cada *Node* Perceptual possui uma quantidade fixa e configuração própria de *Nodes* Sensoriais como entrada. Como neste trabalho cada *Node* Sensorial captura um valor inteiro (int), as conexões "1" transmitem esse tipo de dado. Como cada *Node* Perceptual recebe

múltiplos inteiros, a entrada de seu `proc` é efetivamente um vetor de inteiros.

As conexões entre o segundo e o terceiro bloco — marcada na Figura 34 com o rótulo numérico "2" — são as conexões entre os *Nodes* Perceptuais e os *Nodes* Comportamentais. Nesta etapa, todos os *Nodes* do segundo bloco são entradas para os *Nodes* do terceiro bloco. As conexões "2" transmitem valores inteiros (int) e, analogamente à etapa anterior, a entrada dos `proc` dos *Nodes* Comportamentais são vetores de inteiros.

As conexões entre o terceiro e o quarto bloco — marcada na Figura 34 com o rótulo numérico "3" — são as conexões entre os *Nodes* Comportamentais e os *Nodes* Motores. Nesta etapa, os *Nodes* Motores podem possuir um ou mais *Nodes* Comportamentais como entrada, seguindo critérios estabelecidos pelo "Controle de Processo" (vide subseção 5.3.3). As conexões "3" transmitem valores lógicos booleanos (bool) e a entrada dos `proc` dos *Nodes* Motores são vetores de um ou mais valores booleanos.

5.3.3 Controle de Processo e Heurísticas de Decisão

Diversos aspectos da arquitetura mostrada na subseção anterior são específicos da aplicação, definidos por heurísticas ou arbitrariamente, como o número de *Nodes* Perceptuais e Comportamentais. Apesar desses meta-parâmetros serem ajustáveis tal como quaisquer processos do gênero, talvez a definição de uma meta entidade que controla esses detalhes facilite a compreensão do leitor. Assim, pode-se considerar que qualquer definição, ajuste ou escolha que não seja diretamente ligada a outro processo já existente — como a Estratégia Evolutiva — é executado pelo Controle de Processos. A heurística da escolha das conexões entre os *Nodes* Sensoriais e os *Nodes* Perceptuais, por exemplo, pode ser encarada como pertencente a este Controle.

5.3.3.1 Sobre as Conexões Entre os *Nodes*

O número de *Nodes* ao longo de todo o processo de otimização é fixo. Para os *Nodes* Sensoriais e Motores essa afirmação é mais simples de compreender, visto que existem restrições associadas à aplicação. Para os demais *Nodes*, esses valores são arbitrados e, neste trabalho, foram selecionados de modo a não sobrecarregar a máquina onde os experimentos foram realizados, sendo 35 Perceptuais e 35 Comportamentais. As informações de quaisquer modelos internos são zeradas e as conexões são refeitas para cada Indivíduo durante o processo de avaliação de *fitness*. Mais detalhes e definições sobre Estratégia Evolutiva serão abordados na subseção 5.3.4.

Cada um dos *Nodes* Perceptuais possui conexões de entrada fixas (não são reconfiguráveis) e escolhidas aleatoriamente da seguinte maneira:

1. um número k é escolhido seguindo uma distribuição uniforme entre $n\_sensores/2$

e $n\_sensores$;

2. k sensores são escolhidos seguindo uma distribuição uniforme sem reposição dentre todos os sensores disponíveis.

Assim, é bastante improvável que dois *Nodes* Perceptuais coletem informações do mesmo conjunto de *Nodes* Sensoriais, podendo ser tratados como únicos.

O restante das conexões são reconfiguráveis de acordo com cada Indivíduo e seguem conforme já mencionado anteriormente: todos os *Nodes* Comportamentais possuem todos os *Nodes* Perceptuais ativos como entrada e cada *Node* Motor possui apenas um *Node* Comportamental como entrada, sendo os demais comportamentos eliminados do Indivíduo.

5.3.4 Detalhes da Estratégia Evolutiva

Estratégias Evolutivas (EEs) constituem uma classe de algoritmos de otimização inspirados nos princípios da evolução biológica (Algoritmos Evolutivos), caracterizados pela seleção natural e pela reprodução com variação (CASTRO, 2006; BEYER; ARNOLD, 2001). Esses algoritmos são aplicados para resolver problemas complexos de otimização, onde as soluções ideais são desconhecidas ou difíceis de serem alcançadas através de métodos convencionais. As EEs operam por meio da simulação de um processo evolutivo, iniciando com uma população de soluções candidatas (indivíduos) para um determinado problema. Cada indivíduo na população é avaliado com base em uma função objetivo (*fitness*), que quantifica a qualidade da solução proposta. Através da aplicação de operadores genéticos, como mutação e recombinação, novas gerações de soluções são criadas. A seleção é então aplicada para escolher os indivíduos mais aptos da geração atual para serem pais da próxima geração, promovendo gradualmente a melhoria das soluções ao longo das gerações.

As Estratégias Evolutivas foram formalmente introduzidas na década de 1960, como uma metodologia para otimização de problemas de engenharia (SCHWEFEL, 1965). Desde então, elas evoluíram e se diversificaram, encontrando aplicações em uma ampla gama de campos, incluindo otimização de projetos de engenharia, inteligência artificial, economia e biologia computacional.

Para aplicar uma Estratégia Evolutiva precisamos elaborar algumas definições. Primeiro, precisamos definir nossa codificação do Indivíduo. Aqui, nosso Indivíduo é um vetor binário com o comprimento do número total de Codelets Perceptuais mais o número total de Codelets Comportamentais, onde cada índice representa um Codelet específico. Nessa definição, um '1' representa que o Codelet correspondente está ativo na composição do Agente e um '0' significa um Codelet não conectado. É importante explicitar que o

número total de Codelets do reservatório é fixo e que cada Codelet Perceptual tem um conjunto próprio de Codelets Sensoriais conetados, algo que será tratado mais a frente. Esta decisão do número fixo foi tomada de modo a evitar agentes desnecessariamente grandes e de modo a simular um número fixo de dispositivos disponíveis. Um exemplo de Indivíduo é mostrado na Figura 36:

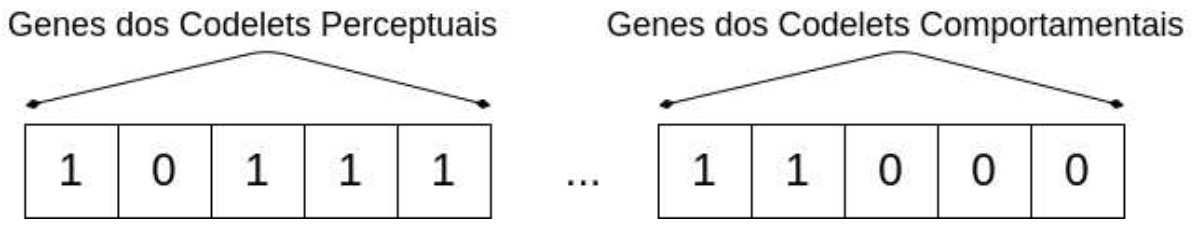


Figura 36 – Exemplo de indivíduo para nosso processo de Estratégia Evolutiva

Em segundo lugar, precisamos definir um método de mutação. Para este trabalho adotamos uma probabilidade simples de '*bit-flip*' como mutação, significando que cada indivíduo tem uma probabilidade *mut_p* de ser mutante e cada um de seus genes tem uma probabilidade *ind_m* de mudar seu estado. Portanto, se um '0' se tornar um '1', isso significa que devemos considerar o Codelet correspondente ao montar a topologia do agente.

Como adotamos a saída dos Codelets Motores como exclusivamente binária, podemos escolher o método de avaliação de *fitness* como uma Distância de Hamming entre as saídas esperadas do sistema e as saídas reais avaliadas em relação a um conjunto de teste, após treinamento, que será melhor tratado na subseção 5.3.5. Com esta escolha, quanto menor esse valor, chamado de *score*, melhor o *fitness* individual.

Em seguida, escolhemos como método de seleção os cinco melhores indivíduos para a próxima geração e o melhor para ser clonado. Fizemos isso para ter alguma possibilidade de recuperação em situações de mínimos locais.

Além disso, optamos por não ter nenhum processo de *crossover*. Essa escolha foi decisão de projeto, pois previámos que não faria alterações significativas e exigia um processo adicional que poderia tornar cada iteração mais longa.

5.3.5 O Processo de Treinamento

O principal componente do processo de otimização, conforme já mencionado, é uma simples — mas eficiente — Estratégia Evolutiva. Este processo é representado graficamente na Figura 37. Mas nosso método *evaluation* requer mais atenção. É nesta parte que tentamos realizar um mapeamento de entrada-saída.

Inicialmente, com base no *Indivíduo*, reconfiguramos as conexões dos Codelets corretamente, incluindo a limpeza de todo dado armazenado nas Memórias e decidindo

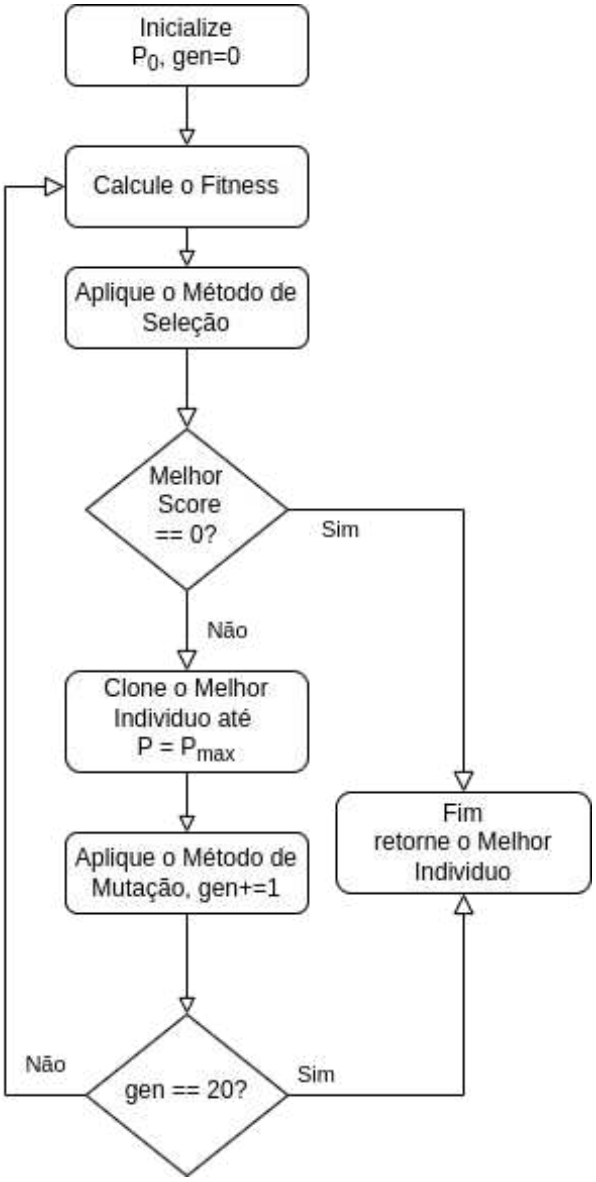


Figura 37 – Representação UML da Estratégia Evolutiva da abordagem proposta

qual Codelet Comportamental alimentará cada Codelet Motor. Fazemos isso escrevendo um novo arquivo *fields.json* e enviando-o para cada Codelet, e forçando um valor vazio para as Memórias relevantes, como o *motor-memory*, em cada Codelet Motor.

Na segunda parte, em posse de um conjunto de treinamento, enviamos as entradas deste conjunto para cada Codelet Perceptual relevante (aqueles com um correspondente '1' na codificação do *Indivíduo*) e ativamos um sinal de "treinamento", através de uma Memória presente em cada um deles com este único propósito. Este processo de treinamento visa criar um valor numérico único (um símbolo) de sua própria situação acerca dos sensores observados, assim, uma saída de um Codelet Perceptual é um inteiro correspondente a uma posição em um vetor com observações únicas. Por exemplo, se "[[0, 1, 0], [0, 1, 1], [0, 1, 0]]" representa um conjunto de entradas de treinamento, suas respectivas saídas poderiam ser algo como "[[0], [1], [0]]". Lembrando que cada Codelet Perceptual

tem suas próprias conexões de sensores, então suas respostas diferem umas das outras.

Antes de realizar a etapa de avaliação dos Indivíduos da população, é necessário realizar a parte do treinamento supervisionado. Nesta etapa precisamos treinar os Codelets Comportamentais adequadamente considerando a resposta de entrada-saída de cada Codelet Perceptual e respectivo Codelet Motor que o mesmo alimenta. Fazemos isso em duas etapas: agregando respostas Codelets Perceptuais para cada entrada de treinamento e mapeando essas respostas para uma entrada de um Codelet Motor que geraria a saída desejada. Esses dois conjuntos (respostas *Perceptual* e entrada *Motor*) representam o treinamento que temos em cada Codelet Comportamental.

Neste trabalho, o método de treinamento utilizado para as Árvores de Decisão foi a divisão utilizando o índice Gini de diversidade como critério de redução de impureza (BREIMAN, 2017). Isso será melhor comentado na subseção 5.3.6.

A etapa de agregação é feita solicitando o modelo já treinado de Codelets Perceptuais e suas respectivas máscaras de entrada (representando quais sensores os alimentam) e montando uma saída Perceptual conjunta. Apesar desse assunto não ser tratado neste trabalho, essa abordagem também pode ser útil para a tarefa sensível a dados, pois o sistema que coleta/envia as informações não precisa ser o mesmo que centraliza a Estratégia Evolutiva.

Então, precisamos definir como cada Codelet Motor responde. Esta parte é sensível, pois um Codelet Comportamental pode ser atribuído a dois (ou mais) Codelet Motores com comportamentos fundamentalmente diferentes. Isso pode impossibilitar o treinamento correto do modelo. Após a coleta de dados, enviamos cada informação de entrada-saída para o respectivo Codelet Comportamental.

Em seguida, estamos prontos para avaliar o desempenho do sistema. Um a um, enviamos o exemplo correspondente no conjunto de teste como entrada do agente e esperamos até que a informação se propague obtendo, como resposta do sistema, um vetor com todos os Codelets Motores. Esse "tempo de espera" está diretamente relacionado a sobrecargas de comunicação e simultaneidade de processos, se estiver usando um único computador.

Por fim, calculamos a distância de Hamming entre os valores esperados (saída do teste) e a saída real. Isso representará o *fitness* do *Indivíduo*.

5.3.6 Algoritmo de Árvores de Decisão

O algoritmo de aprendizado da Árvore de Decisão aprende recursivamente a Árvore da seguinte maneira:

1. Atribua todas as instâncias de treinamento à raiz da Árvore. Defina o nó atual como

o Nó Raiz;

2. Para cada atributo

   a Particione todas as instâncias de dados no nó pelo valor do atributo;

   b Calcule a taxa de ganho de informação ou redução de impureza do particionamento conforme o critério escolhido.

3. Identifique a característica resultante na maior taxa de ganho de informação (ou redução de impureza). Defina esse recurso como o critério de divisão no nó atual;

4. Se a melhor taxa de ganho de informação (ou redução de impureza) for 0, marque o nó atual como uma Folha e retorne;

5. Particione todas as instâncias segundo o valor do atributo do melhor recurso.

6. Para cada nó Filho:

   a Se o nó Filho for "puro" (tem instâncias de apenas uma classe), marque-o como uma Folha e retorne;

   b Se não, defina o nó Filho como o nó atual e volte para a etapa 2.

No caso específico de uma Árvore utilizada para regressão, esse particionamento descrito acima continua até que um número pré-definido de indivíduos esteja presente. O processo de inferência ocorre ao percorrer a Árvore até uma folha, sendo o valor previsto a média das instâncias de treinamento presentes nesta folha (ou uma classe rotulada, no caso de problemas de classificação) (BREIMAN, 2017). Neste trabalho, utilizamos a versão para classificação.

A impureza de Gini mede a frequência com que um elemento escolhido aleatoriamente de um conjunto seria incorretamente rotulado caso fosse rotulado aleatoriamente e independentemente segundo a distribuição de rótulos no conjunto. Atinge seu mínimo (zero) quando todos os casos no nó se enquadram em uma única categoria de destino. Ele pode ser expresso pela fórmula:

$$\mathrm{I}_G(p) = \sum_{i=1}^{J} \left(p_i \sum_{k \neq i} p_k \right) = \sum_{i=1}^{J} p_i(1 - p_i) = \sum_{i=1}^{J} (p_i - p_i^2) = \sum_{i=1}^{J} p_i - \sum_{i=1}^{J} p_i^2 = 1 - \sum_{i=1}^{J} p_i^2.$$

onde J representa o número total de classes (folhas) e p_i representa a frequência relativa de uma classe i do total J.

Nas Árvores de decisão, pode-se escolher a redução deste critério de impureza como forma de dividir o conjunto de elementos a cada nó.

No capítulo seguinte falaremos sobre experimentos e seus respectivos resultados.

5.4 Resumo do Capítulo

Neste capítulo falamos sobre o método proposto, incluindo a abordagem escolhida para treinamento de cada componente, bem como a meta-heurística escolhida para definição da topologia do agente.

6 Experimentos

Este capítulo aborda a aplicação prática e a validação das teorias e abordagens discutidas nos capítulos anteriores por meio de uma série de experimentos cuidadosamente projetados. Estes experimentos visam ilustrar a viabilidade, eficácia e resiliência de Sistemas Cognitivos Distribuídos construídos com o DCT em cenários complexos e dinâmicos, destacando o potencial e as capacidades dos *Cognitive Twins* evolutivos. Cada experimento tem uma intenção clara, desenhada para testar diferentes aspectos do framework DCT e demonstrar a aplicabilidade deste em contextos reais e desafiadores.

O primeiro experimento visa investigar a resiliência de sistemas DCT, focando em sua capacidade de manter operações críticas frente a falhas e interrupções. Este experimento simula cenários onde componentes do sistema são submetidos a falhas, testando a habilidade do sistema DCT de se adaptar e recuperar de tais eventos adversos sem perda significativa de desempenho ou funcionalidade, visando demonstrar sua capacidade de autorregeneração

O segundo experimento, aplicando o DCT ao World Server Coppelia, explora a integração do DCT em ambientes simulados, oferecendo uma oportunidade de observar a interação entre Agentes Cognitivos e um ambiente virtual complexo. Este experimento tem o intuito de demonstrar a flexibilidade do DCT em aplicações que vão além de simples tarefas computacionais, abrangendo a interação com ambientes que simulam condições do mundo real. Através desta aplicação, pretende-se avaliar a capacidade do DCT em suportar o desenvolvimento de agentes que possam operar e interagir de maneira convincente em ambientes virtuais.

Finalmente, o terceiro experimento foca diretamente no conceito de *Cognitive Twins*. Este experimento é projetado para testar a viabilidade de construir um modelo computacional que imite o comportamento de uma entidade externa, baseando-se em dados coletados de interações desta entidade. O objetivo aqui é investigar até que ponto é possível criar um "gêmeo cognitivo" que não apenas replica o comportamento observável mas também oferece *insights* sobre os processos cognitivos subjacentes da entidade modelada. Este experimento visa destacar o potencial do DCT como uma ferramenta para o avanço da pesquisa em *Cognitive Twins*, enfatizando a importância de modelos cognitivos distribuídos.

6.1 Experimento 1 - Resiliência de Sistemas DCT

Para demonstrar a propriedade de autorregeneração — que fornece resiliência aos sistemas DCT — foi construído um experimento de passagem de mensagens no qual um sistema arbitrário teria alguns de seus nós falhando momentaneamente, de modo aleatório, enquanto o desempenho total do sistema é comparada com o seu valor-base.

Para realizar o experimento, primeiro foi necessário criar um sistema DCT com vários nós interconectados. Cada nó conseguia enviar e receber mensagens de outros nós do sistema, sendo essa mensagem uma string com uma *hash* única, que o identificava. Para acompanhar a resposta do sistema, um monitor enviava requisições periódicas de acesso a memórias específicas de cada nó (*internal-memory*) e, caso não obtivesse resposta devido à indisponibilidade do nó, registraria como uma string vazia. Ou seja, a métrica utilizada para medir o desempenho da rede é o percentual de mensagens corretamente recuperadas, que indica um percentual de disponibilidade dos nós da rede. Espera-se que os nós que falharam recuperem-se sozinhos. Assim, basta o monitoramento contínuo do sistema para comprovar ou não este comportamento de autorregeneração.

Com o sistema e o algoritmo de monitoramento em funcionamento, é possível simular as falhas aleatórias nos nós. Para isso, basta selecionar alguns nós aleatoriamente (seguindo uma distribuição uniforme) e finalizar a força os processos do *server* e *codelet* interno destes nós. Esta etapa foi realizada sistematicamente por um método que recebia parâmetros, que foram variados ao longo deste experimento para estressar a propriedade a ser demonstrada. O comportamento era o de rápida recuperação do desempenho base.

Os parâmetros variados ao longo do experimento foram:

- **Tamanho da rede**: 10 ou 20 Nós, construídos a partir de uma base de 5 nós completamente conectados e tendo nós adicionados posteriormente a partir de uma lei de progressão exponencial para obtermos configurações de conexões que possam ser observadas em sistemas reais;
- **Fração de nós a falhar por turno**: Valor $\in [0.1, 0.2, 0.3, 0.4]$. Determina o número de nós que falham (parte inteira da multiplicação `Tamanho da rede` x `fração`) a cada turno de um experimento em particular;
- **Número de turnos**: Valor $\in [1, 2, 3, 4, 5]$;
- **Intervalo entre os turnos**: Valor $\in [40, 30, 20, 10, 5, 3]$, em segundos.

Cada combinação da configuração acima foi realizada dez vezes, a fim de se obter algum valor estatístico, mas permitindo que o experimento fosse realizado em tempo hábil.

Considerando as diversas combinações possíveis, o número total de execuções foi de 2400. a Figura 38 ilustra uma rede construída com 10 nós para os experimentos.

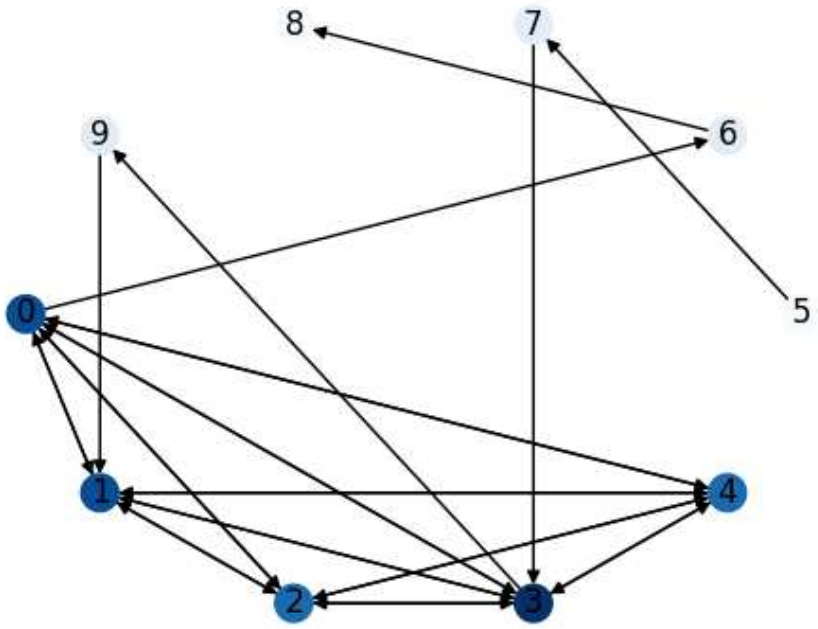


Figura 38 – Exemplo de Rede de 10 nós construída proceduralmente

Os resultados e discussões deste experimento encontram-se na subseção 6.4.1.

6.2 Experimento 2 - Aplicação no *World Server Coppelia*

Neste segundo experimento, nosso objetivo foi demonstrar que, em termos de eficiência no controle de agentes, as capacidades de um sistema cognitivo distribuído são ao menos equivalentes àqueles construídos de forma centralizada. Desenvolvemos, portanto, uma versão distribuída do DemoCST[1], uma arquitetura cognitiva para o controle de um agente no "*World Server 3D*" (WS3D), disponível como tutorial de demonstração do CST[2]. Na versão original, um robô simples cuja mente foi construída com o CST deve se deslocar pelo ambiente e coletar joias para cumprir tarefas predeterminadas. Aqui, utilizamos como ambiente o CoppeliaSim[3] — anteriormente conhecido como V-Rep

[1] <https://github.com/CST-Group/DemoCST>
[2] <https://cst.fee.unicamp.br/examples/ws3dexample>
[3] <https://coppeliarobotics.com/>

(ROHMER *et al.*, 2013) — um simulador de robótica que dispõe de APIs que podem ser utilizadas para execução de scripts por programas externos.

Nesta aplicação, o DCT é usado para construir a mente artificial de um robô simples que existe em uma pequena plataforma no CoppeliaSim e é populada com maçãs e joias de várias cores que surgem periodicamente em lugares e quantidades aleatórias, seguindo uma heurística. O robô pode rotacionar em ambas as direções e se movimentar para frente ou para trás com velocidade fixa. O objetivo do robô é recolher joias do ambiente segundo uma tabela de trocas (*leaflets*) que associa a cada grupo de joias com cores uma certa quantidade de pontos. O robô ainda deve comer maçãs periodicamente, o que lhe proporciona energia, de modo a evitar ficar totalmente sem energia, o que impossibilitaria a execução de quaisquer ações e, consequentemente, a falha no cumprimento de seus objetivos.

As seguintes informações podem ser obtidas dos sensores virtuais:

- **position**: vetor com os valores de posição X e Y em relação à origem da plataforma;
- **fuel**: indica a quantidade de energia do robô;
- **things_in_vision**: lista com os objetos no campo de visão do robô.

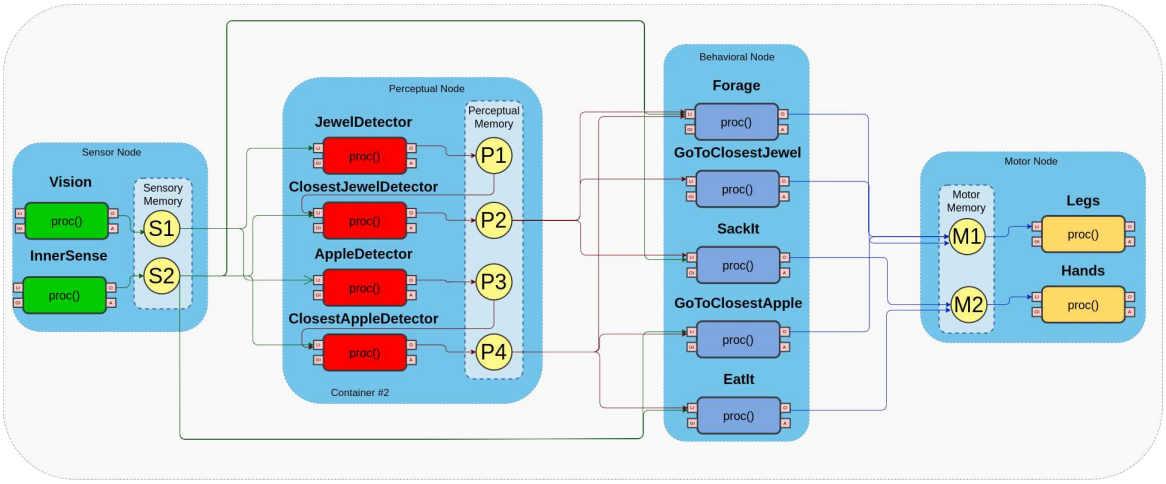


Figura 39 – Topologia da mente do agente para a aplicação do WS3D

A topologia da mente construída para o agente do WS3D, distribuída em 4 diferentes *Nodes*, pode ser vista na Figura 39. Ela possui os seguintes *Nodes* (e respectivos Codelets):

- **Sensory Node**
  - **Vision**: Codelet responsável por capturar os objetos presentes na faixa de visão do robô;

  - **InnerSense**: Codelet responsável por requisitar informações diretamente relacionadas ao robô, como posição e combustível.

- **Perceptual Node**

  - **AppleDetector**: Codelet responsável por identificar quais objetos são maçãs e guardar um vetor com as informações destes objetos;
  - **ClosestAppleDetector**: Codelet responsável por identificar qual maçã está mais próxima do robô;
  - **JewelDetector**: Codelet responsável por identificar quais objetos são joias e guardar um vetor com as informações destes objetos. Caso um determinado tipo de joia não esteja presente nos requisitos de nenhum *leaflet*, ele não será incluído;
  - **ClosestJewelDetector**: Codelet responsável por identificar qual joia está mais próxima do robô.

- **Behavioral Node**

  - **GoToClosestApple**: Codelet responsável por disparar um comportamento que leva o robô até à maçã mais próxima. Esse comportamento só é disparado se houver ao menos uma maçã conhecida e o combustível estiver abaixo de 300 (30%);
  - **EatClosestApple**: Codelet responsável por disparar um comportamento que faz o robô comer uma maçã, caso essa maçã encontre-se ao alcance;
  - **GoToClosestJewel**: Codelet responsável por disparar um comportamento que leva o robô até à joia mais próxima. Esse comportamento só é disparado se houver ao menos uma joia conhecida e o combustível estiver acima de 300 (30%);
  - **SackClosestJewel**: Codelet responsável por disparar um comportamento que faz o robô pegar e guardar uma joia em sua sacola, caso essa joia encontre-se ao alcance;
  - **Forage**: Codelet responsável por, na falta de joias ou maçãs desejáveis, disparam um comportamento de busca, enviando comando de ”rotacionar” para o Codelet *Legs*.

- **Motor Node**

  - **Hands**: Codelet responsável por enviar comandos motores relacionados às mãos do robô, sendo eles *eat_it* e *sack_it*;

– **Legs**: Codelet responsável por enviar comandos motores relacionados às pernas do robô, sendo eles *rotate*, *move_to* e *stop*.

Sempre que um *leaflet* é satisfeito, o mesmo é automaticamente entregue e o agente recebe pontos como recompensa. Quando todos os três *leaflets* forem satisfeitos, o programa é encerrado.

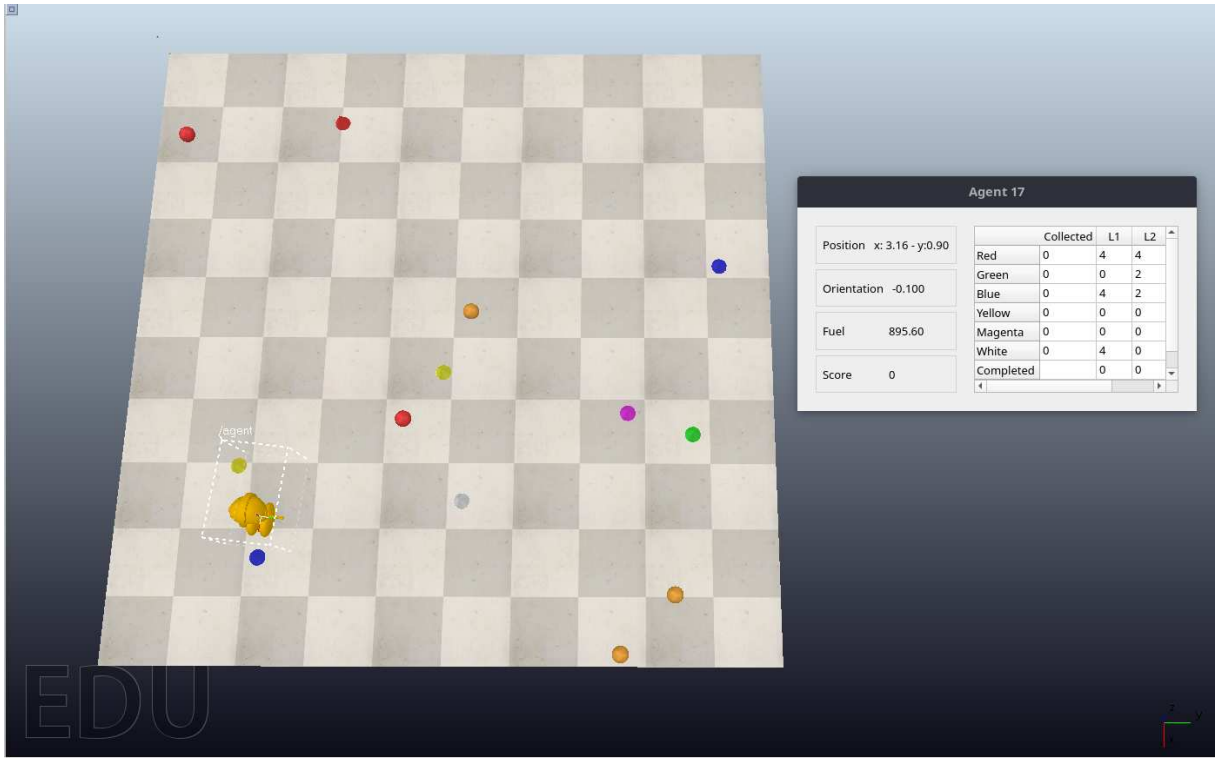


Figura 40 – Captura de tela do WS3D em execução

A Figura 40 mostra uma captura de tela do WS3D em execução. O Ambiente é inicializado com o robô numa posição fixa e uma quantidade, tipo e posição de outros objetos seguindo uma heurística. A cada 1 minuto, novas joias e maçãs são geradas, seguindo a mesma heurística da inicialização. Para efeitos comparativos, coletamos o tempo e o consumo de memória de um agente criado com o DCT e outro com o CST para o mesmo cenário. Foram coletados 25 resultados de cada versão (totalizando 50 coletas) contados a partir do momento em que o agente está completamente funcional. Ambos os experimentos foram realizados na versão 4.4.0 do CoppeliaSim. O código do `dct_ws3d` pode ser encontrado em <https://github.com/wandgibaut/dct_ws3d> enquanto o do `ws3dCoppelia` pode ser visto em <https://github.com/wandgibaut/WS3D-Coppelia-CST>. Os resultados desse experimento encontram-se na subseção 6.4.2.

6.3 Experimento 3 - *Cognitive Twin*

Para testar nossa proposta de construção de *Cognitive Twin*, montamos um conjunto de experimentos que simulam interações de um agente (que pode ser, por exemplo, um humano) com alguns dispositivos — sensores e atuadores, listados mais a frente — em uma casa inteligente. Como sensores, temos, por exemplo, detectores de presença e termômetros, que geram os dados relevantes à medida que o agente tem seus comportamentos usuais. Como atuadores, temos alguns dispositivos que exigiriam interação direta, como interruptores de luz, dos quais obtemos as preferências do agente. Nosso objetivo foi mapear as leituras dos sensores e o acionamento dos dispositivos atuadores. Encontramos trabalhos que usaram dados centrados na interação, como o de Engelmann *et al.* (2016), mas nenhum deles tinha um conjunto de dados disponível publicamente. Sendo assim, optamos por simular o comportamento do agente a ser copiado levemente baseado no trabalho de Mendez-Vazquez *et al.* (2009), que utiliza processos de decisão markovianos para simular comportamento humano simplificado.

Especificamente, simulamos a casa e seus dispositivos como um servidor que escuta uma porta específica para solicitações de informação e de alterações nos dados referentes aos dispositivos. Um sensor pode solicitar as leituras atuais de seu dispositivo específico e um atuador pode definir um dispositivo como "ligado" ou "desligado".

Como explicado anteriormente, um dos principais processos no treinamento do nosso *Cognitive Twin* é a Estratégia Evolutiva, onde deixamos um conjunto de escolhas aleatórias de Codelets Perceptuais e Comportamentais fazer parte do agente. Cada Codelet Perceptual difere do outro pelos sensores que coleta informações: este pode ser qualquer número entre metade dos sensores disponíveis e todos eles, escolhidos aleatoriamente com distribuição uniforme. Cada *Codelet* Comportamental difere dos demais no que tange a qual *Codelet* Motor (ou *Codelets* Motores) alimenta, em termos de estrutura interna, sendo todos eles completamente iguais, caso contrário.

Cada Indivíduo (da População da Estratégia Evolutiva) é definido por um vetor de valores binários que representam o pertencimento ao *Cognitive Twin*. Utilizamos para o processo de mutação uma probabilidade de "bit-flip" e optamos por não usar métodos de *crossover*. O método de seleção é o dos "cinco melhores", o que significa que mantemos os cinco melhores Indivíduos em cada geração. A População no início de cada geração tem vinte indivíduos e a População inicial foi gerada aleatoriamente, com cada Indivíduo tendo uma probabilidade de 20% de '1' em cada gene, refletindo aproximadamente como um "número médio de codelets" de 20% do total possível.

Pretendíamos que os experimentos fossem suficientemente semelhantes ao comportamento de um humano, ainda que fossem sintéticos. Com base no trabalho de Mendez-Vazquez *et al.* (2009), geramos dados baseados em uma matriz de probabilidade de transi-

ção associada a funções de probabilidade de interação com cada dispositivo. Assim, nossos dados são gerados com base em qual sala o agente se encontra e, a partir dai, com quais dispositivos ele interage.

Nosso agente primordial tem algumas propriedades que caracterizam suas preferências, como e.g., conforto térmico. Essas propriedades — e como seus valores são escolhidos — são:

- **thermal_comfort**: um número inteiro aleatório com distribuição uniforme no intervalo [20, 30] que representa o limiar de temperatura em que nosso agente primordial passa a se sentir desconfortável e liga (ou desliga) o ar-condicionado;
- **light_comfort**: um número real aleatório com distribuição uniforme no intervalo [0.4, 1] que representa a luminosidade que nosso agente primordial passa a se sentir desconfortável e acende (ou apaga) as luzes;
- **voice_prob**: um número real aleatório com distribuição uniforme no intervalo [0, 1] que representa a probabilidade de nosso agente primordial usar um dispositivo ativado por voz;
- **coffee_prob**: um número real aleatório com distribuição uniforme no intervalo [0, 1] que representa a probabilidade de nosso agente primordial usar a máquina de café, uma vez na sala;
- **plant_water_th**: um número inteiro aleatório com distribuição uniforme no intervalo [40, 100] que representa um limite acima do qual uma planta deve ser regada.

A Figura 41 ilustra a casa projetada. Um agente só poderia se mover entre salas consecutivas, como um humano faria. Isso pode ser representado por uma matriz de adjacência. Considerando que cada elemento não nulo da matriz de adjacência tem igual probabilidade de ser escolhido, a partir destes foi construída uma matriz de transição. Isso representa uma "matriz de transição de salas" e, em cada sala, o agente terá algumas interações com os elementos presentes nela. Esses elementos, sensores e atuadores, são mostrados na lista abaixo com uma breve explicação de como funcionam seus estados. A Figura 42 mostra um diagrama de transição de estados para o agente simulado aqui descrito.

- **Sensores existentes**
  - **presença**: se o agente estiver na sala, é '1'. Caso contrário é '0';
  - **temperatura**: um número inteiro aleatório no intervalo [20, 30];
  - **luminosidade**: um número inteiro aleatório no intervalo [0, 10];

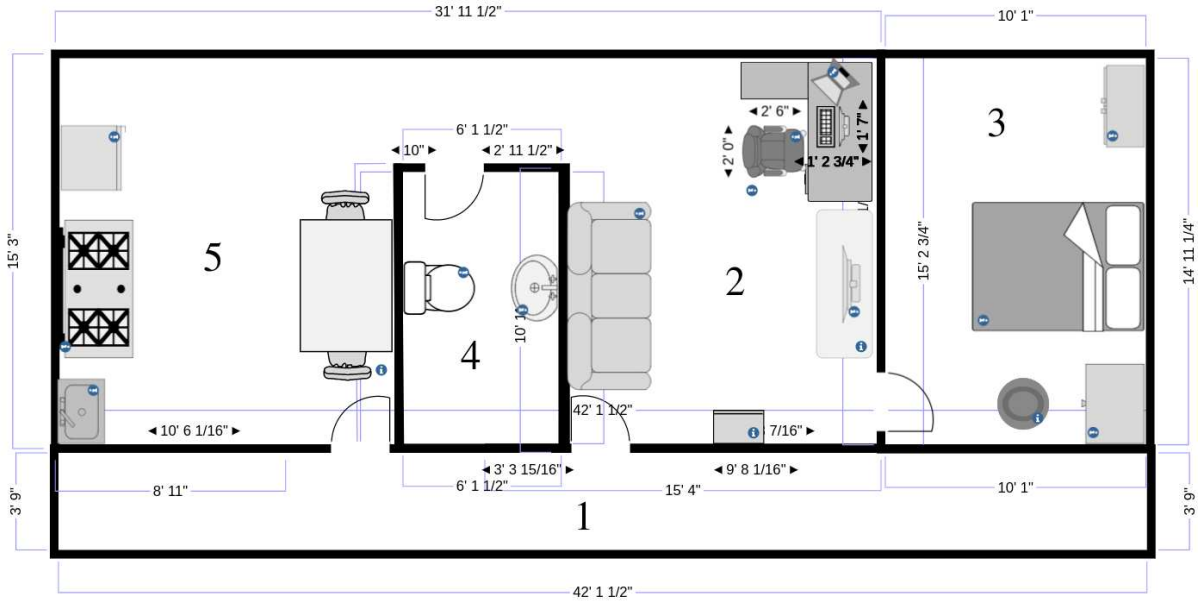


Figura 41 – Casa utilizada no design dos experimentos. Note que é apenas uma representação, utilizada somente para se ter uma imagem clara ao construir a matriz de adjacência

  - **voz**: uma escolha binária aleatória com p_1 igual a *p_voice*;
  - **detector_de_fumaça**: uma escolha binária aleatória com probabilidade $p_1 = 0.1\%$ representando uma ocorrência de incêndio;
  - **chuveiro_temperatura**: um número inteiro aleatório no intervalo [20, 30];
  - **umidade**: um número inteiro aleatório no intervalo [0, 10].

- **Atuadores existentes**
  - **ar_condicionado**: '1' se o sensor **temperatura** for menor que *thermal_comfort* e '0' caso contrário;
  - **luzes**: '1' se o sensor **luminosidade** for menor que *light_comfort* e '0' caso contrário;
  - **caixa_de_som**: é '1' se o sensor **voz** foi ativado e '0' caso contrário;
  - **café_máquina**: uma escolha binária aleatória com p_1 igual a *coffee_prob*;
  - **anti_incêndio**: é '1' se o sensor **detetor_de_fumaça** foi ativado e '0' caso contrário;
  - **chuveiro_controle**: '1' se o sensor **chuveiro_temperatura** for menor que *thermal_comfort* e '0' caso contrário;
  - **regagem_plantas**: '1' se o sensor **umidade** for menor que *plant_water_th* e '0' caso contrário.

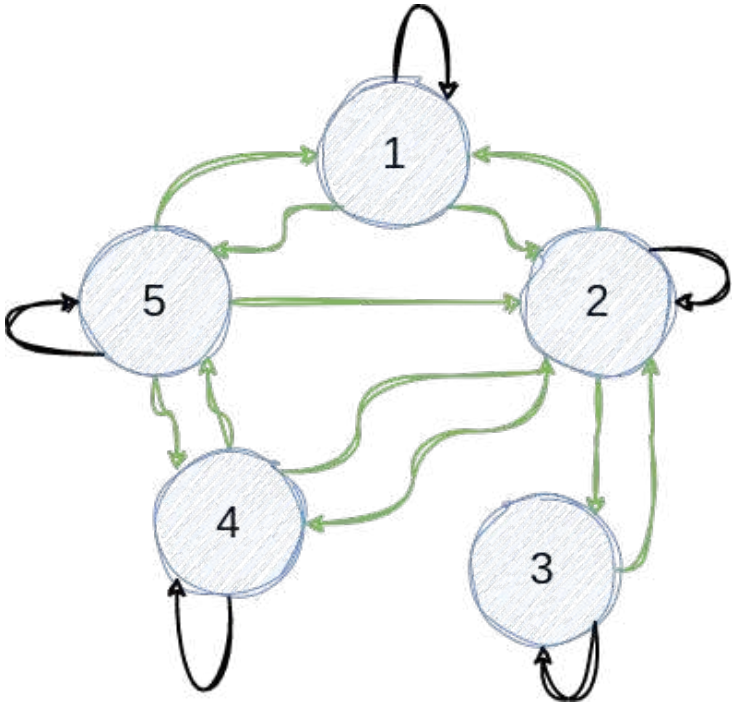


Figura 42 – Diagrama de transição de estados do agente simulado. Cada transição a partir de um estado tem igual probabilidade de ser escolhido

Após transitar para um estado específico (um cômodo), todos os estados de sensores e atuadores relevantes mudam de acordo com seus comportamentos. Listas de sensores e atuadores que cada sala possui são mostradas abaixo.

- **sala**:
  - sensores: presença, temperatura, luminosidade, voz;
  - atuadores: ar_condicionado, luzes, caixa_de_som.
- **quarto**:
  - sensores: presença, temperatura, luminosidade, voz;
  - atuadores: luzes, caixa_de_som.
- **cozinha**:
  - sensores: presença, temperatura, luminosidade, detector_de_fumaça;
  - atuadores: luzes, café_máquina, anti_incêndio.
- **banheiro**:
  - sensores: presença, temperatura, luminosidade, chuveiro_temperatura;
  - atuadores: luzes, chuveiro_controle.

- **fora**:
  - sensores: presença, luminosidade, voz, umidade;
  - atuadores: luzes, caixa_de_som, regagem_plantas.

Para cada execução, coletamos 400 amostras com base em uma caminhada aleatória usando a matriz de transição e a função de interação do dispositivo mostrada acima. Executamos um número total de 5.000 execuções do experimento, nos quais construímos agentes primordiais ligeiramente diferentes e conjuntos de amostras diferentes.

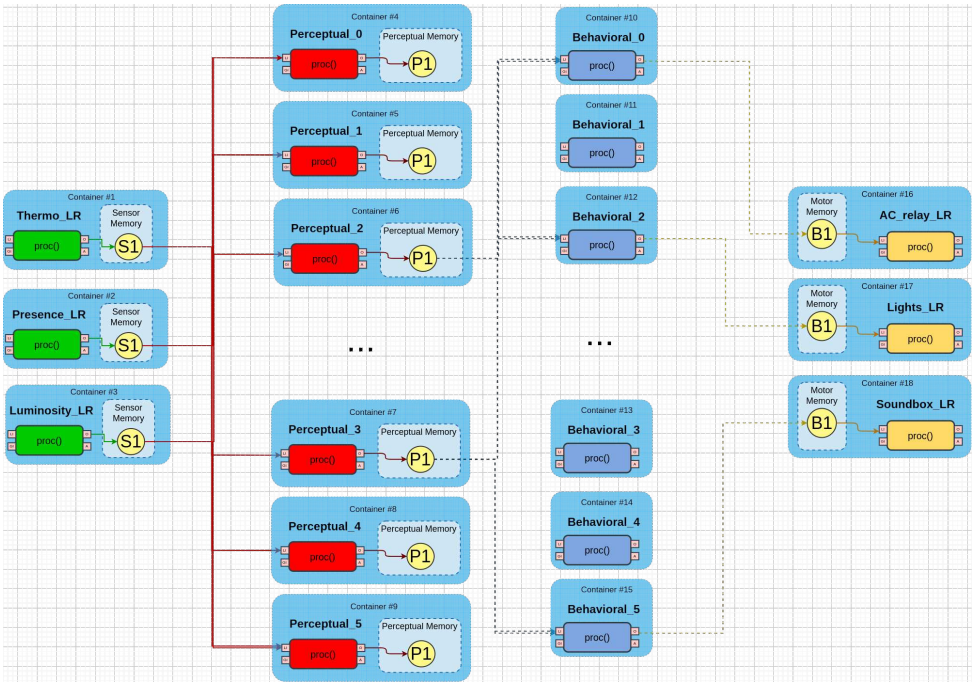


Figura 43 – Representação visual da estrutura de um agente e suas conexões internas. As linhas pontilhadas representam conexões específicas de um candidato durante o processo evolutivo, mas que podem diferir (ou seja, apresentar outras conexões) em outros candidatos

Os experimentos têm um objetivo principal muito específico: alcançar uma topologia de *Cognitive Twin* que apresente a menor (idealmente zero) diferença entre os valores de saída esperados e os reais, seguindo a métrica da distância de Hamming. Uma ilustração de um possível agente alcançado ao término do processo é mostrado na Figura 43. Os resultados desse experimento encontram-se na subseção 6.4.3

Na próxima seção, apresentamos os resultados e discussões de todos os experimentos.

6.4 Resultados e Discussões

Nesta seção, apresentamos os resultados, fazemos uma análise sobre algumas de suas características e tiramos algumas conclusões sobre os experimentos.

6.4.1 Resultados Experimento 1

Neste experimento buscamos avaliar a resiliência do sistema, isto é a capacidade do sistema de continuar operando e recuperar seu desempenho nominal rapidamente após falhas simultâneas de vários nós ou interrupções nas conexões de rede. Aqui o importante é verificar se o sistema conseguiu retornar ao seu desempenho nominal dentro do tempo máximo de execução de cada rodada do experimento.

Os resultados do experimento mostraram que o sistema de troca de mensagens construído com o DCT pode ser considerado de fato resiliente a falhas. Mesmo quando vários Nós falharam simultaneamente ou quando as conexões de rede foram interrompidas, o sistema continuou a funcionar, recuperando seu desempenho nominal rapidamente.As Figuras 44, 45, 46 e 47 mostram o percentual em relação ao desempenho nominal de redes de 10 e 20 nós, sob condições diferentes. É importante ressaltar que essa recuperação se deu de forma completamente automática, sem qualquer intervenção humana. Esse é um recurso importante para sistemas que precisam estar disponíveis 24 horas $\times$ 7 dias por semana, como serviços de computação em nuvem ou sistemas financeiros.

No entanto, a resiliência do sistema não foi perfeita. Em alguns casos, o sistema não se recuperou completamente no tempo estabelecido em cada rodada, como pode ser visto na Figura 47. Isso porque o sistema sofreu falhas mais rapidamente do que conseguiu se recuperar efetivamente.

As figuras 48 a 50 mostram a distribuição de tempo necessário à total recuperação do sistema (quando possível) considerando as variações de tamanho de rede, percentual de falhas e intervalo de tempo entre uma rodada de falhas e outra. Sem muita surpresa, o sistema necessita de mais tempo em redes maiores, percentuais de falhas maiores e intervalos de tempo menores (falhas acontecendo rapidamente).

Os resultados do experimento sugerem que, desde que as falhas ou ataques a um sistema construído com o DCT não sejam simultaneamente volumosos (muitos nós por vez) e muito frequentes (intervalo curto entre as ocorrências), o mesmo pode sustentar-se com desempenho nominal ou próximo disso.

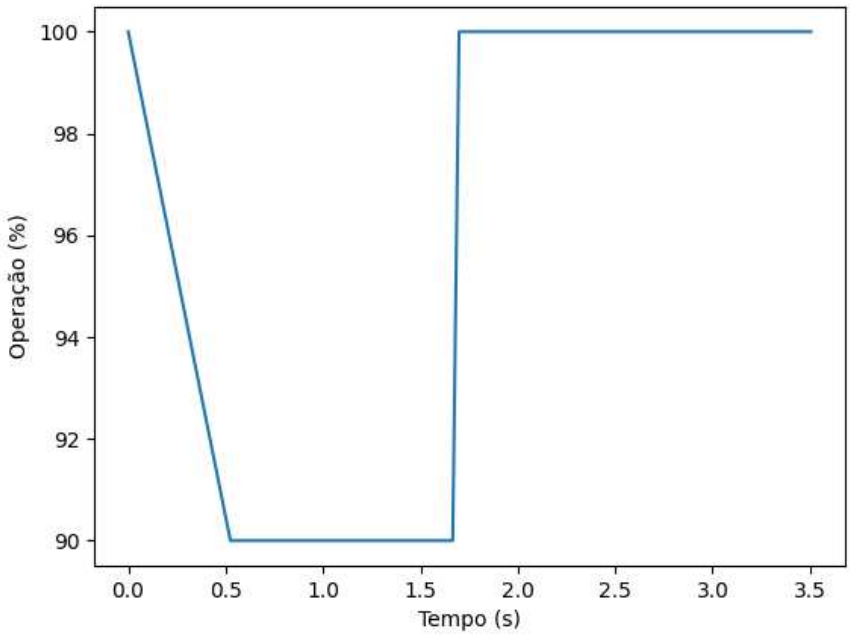


Figura 44 – Variação percentual de desempenho de um sistema DCT sujeito a falhas aleatórias. O sistema possuía 10 nós e estava sujeito a falhas de 10% de sua estrutura 1 única vez

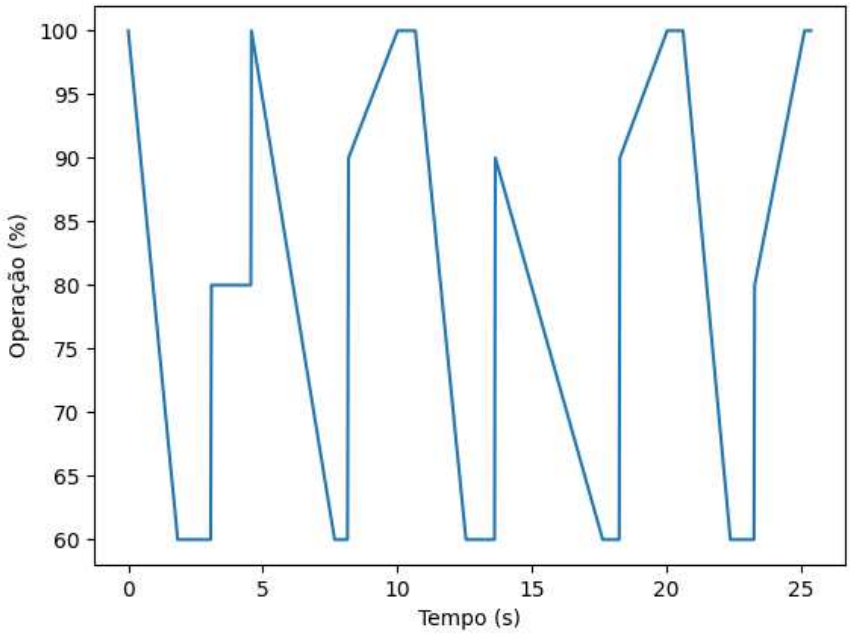


Figura 45 – Variação percentual de desempenho de um sistema DCT sujeito a falhas aleatórias. O sistema possuía 10 nós e estava sujeito a falhas de 40% de sua estrutura 5 vezes com 3 segundos de intervalo entre cada rodada de falhas

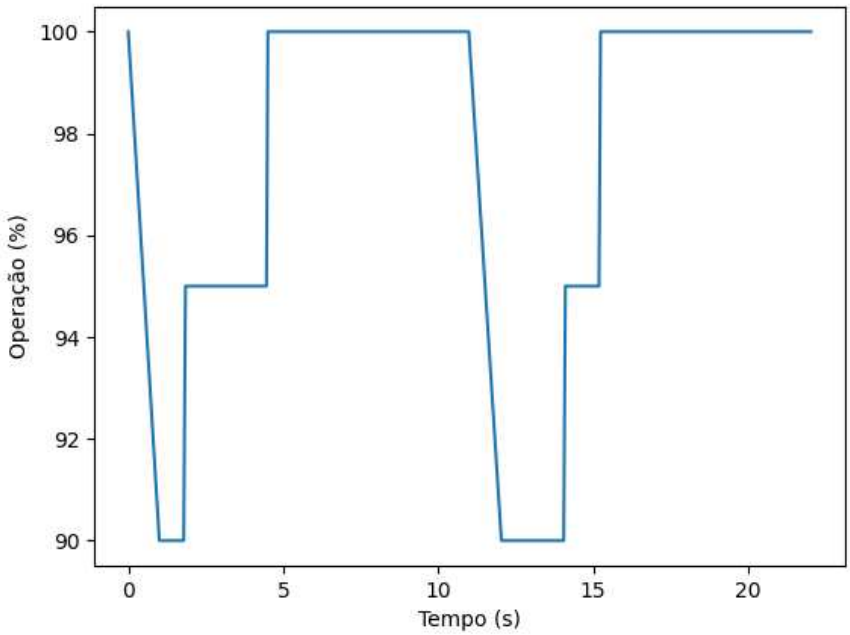


Figura 46 – Variação percentual de desempenho de um sistema DCT sujeito a falhas aleatórias. O sistema possuía 20 nós e estava sujeito a falhas de 10% de sua estrutura 2 vezes com 10 segundos de intervalo entre cada rodada de falhas

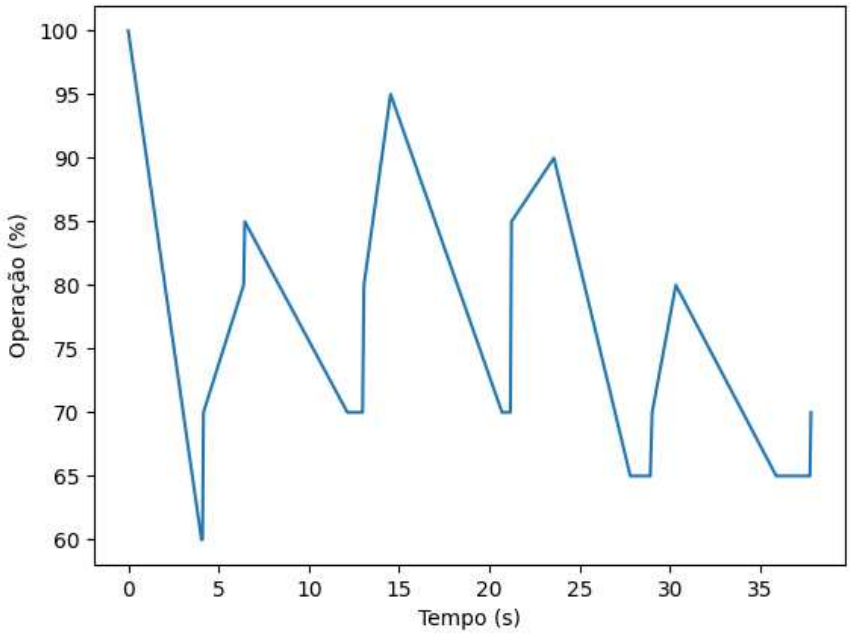


Figura 47 – Variação percentual de desempenho de um sistema DCT sujeito a falhas aleatórias. O sistema possuía 20 nós e estava sujeito a falhas de 40% de sua estrutura 5 vezes com 3 segundos de intervalo entre cada rodada de falhas

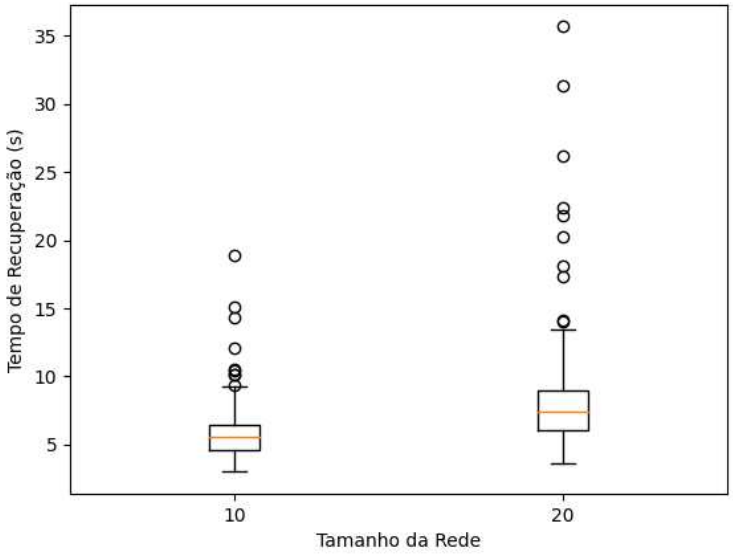


Figura 48 – Boxplot da distribuição do tempo necessário para a recuperação total do sistema para cada um dos tamanhos de rede. Note que, conforme a rede aumentou de tamanho, tanto a mediana do tempo aumentou como também a dispersão dos resultados em valores maiores, indicando uma menor estabilidade.

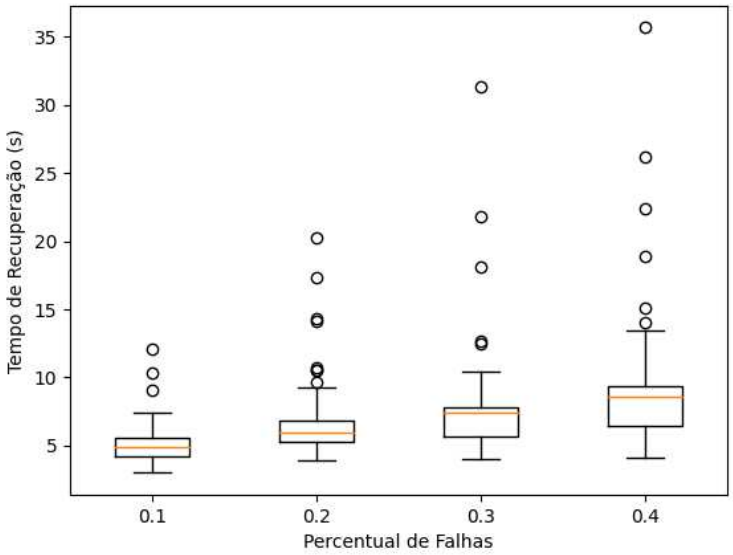


Figura 49 – Boxplot da distribuição do tempo necessário para a recuperação total do sistema para cada percentual de falhas por rodada. Note que percentuais maiores de falhas por rodada culminam em valores e dispersões maiores (sempre para mais).

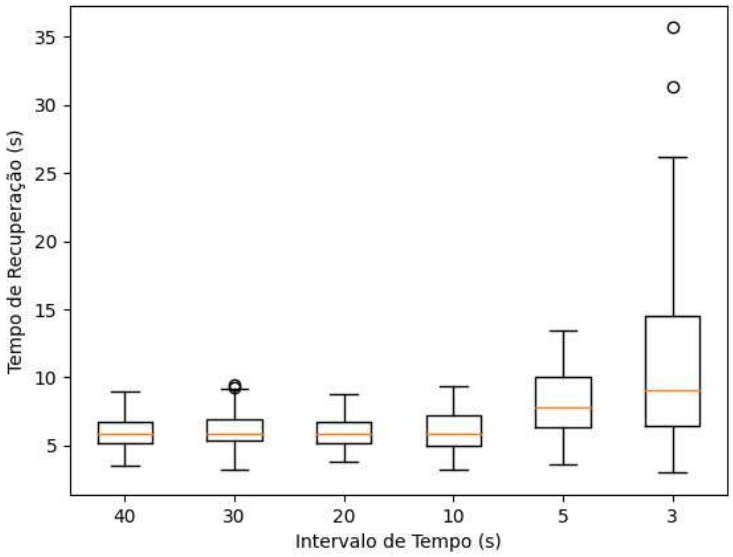


Figura 50 – Boxplot da distribuição do tempo necessário para a recuperação total do sistema para cada intervalo de tempo entre eventos de falha. Conforme pode-se perceber no gráfico, existe pouca diferença na mediana e dispersão para valores de intervalo de tempo até 10 segundos, mudando drasticamente quando este intervalo cai para 5 ou 3s.

6.4.2 Resultados Experimento 2

Nesta aplicação, nosso objetivo foi demonstrar que podemos utilizar um agente cognitivo distribuído para controle de autômatos em ambientes virtuais sem quaisquer prejuízos quando comparados com a alternativa centralizada em relação ao desempenho de tempo. O agente cumpre todas as tarefas propostas em um tempo hábil, dada a disponibilidade das joias necessárias e da não obstrução de sua visão a tais joias, impossibilitando a detecção. O agente ainda periodicamente para de procurar joias e busca e consome maçãs, atendendo a necessidades mais urgentes sempre que seu nível de energia está demasiadamente baixo. As tabelas 5 e 6 detalham os resultados do experimento comparando os desempenhos em tempo e consumo de memória e a Figura 51 traz o histograma da distribuição.

Tabela 5 – Métricas relativas ao tempo total para sucesso do agente

Nome	**CST**	**DCT**
Tempo Médio (segundos)	147,6	137,36
Desvio Padrão (segundos)	51,88	38,41
Mínimo (segundos)	70	62
Q1 (segundos)	125	112
Mediana (segundos)	140	123
Q3 (segundos)	180	170
Máximo (segundos)	265	215

Tabela 6 – Métricas relativas ao uso total de memória pelo agente

Nome	**CST**	**DCT**
Uso Médio de Memória (MB)	46,32	1242,40
Desvio Padrão (MB)	12,34	28,54
Mínimo (MB)	28	1200
Q1 (MB)	38	1208
Mediana (MB)	44	1262
Q3 (MB)	55	1263
Máximo (MB)	72	1272

Pode-se notar tanto pela tabela 6 como pela imagem à direita na Figura 51 que o DCT consome muito mais memória que o CST para a mesma tarefa (aproximadamente 27x mais, em média). Isso se dá devido a uma série de fatores tais como a conteinerização em 4 subsistemas, as rotinas de autorregeneração, a comunicação via REST entre seus subsistemas e a escrita e leitura constante do disco (algo a ser melhorado em versões futuras). Reitera-se que a autorregeneração e a comunicação via REST foi uma decisão de projeto que possibilita o sistema ser resiliente e distribuído ao longo de diversos dispositivos físicos e virtuais, ainda que acabe por consumir mais memória. Já em relação ao tempo de execução, não houve diferenças significativas entre as duas versões do agente.

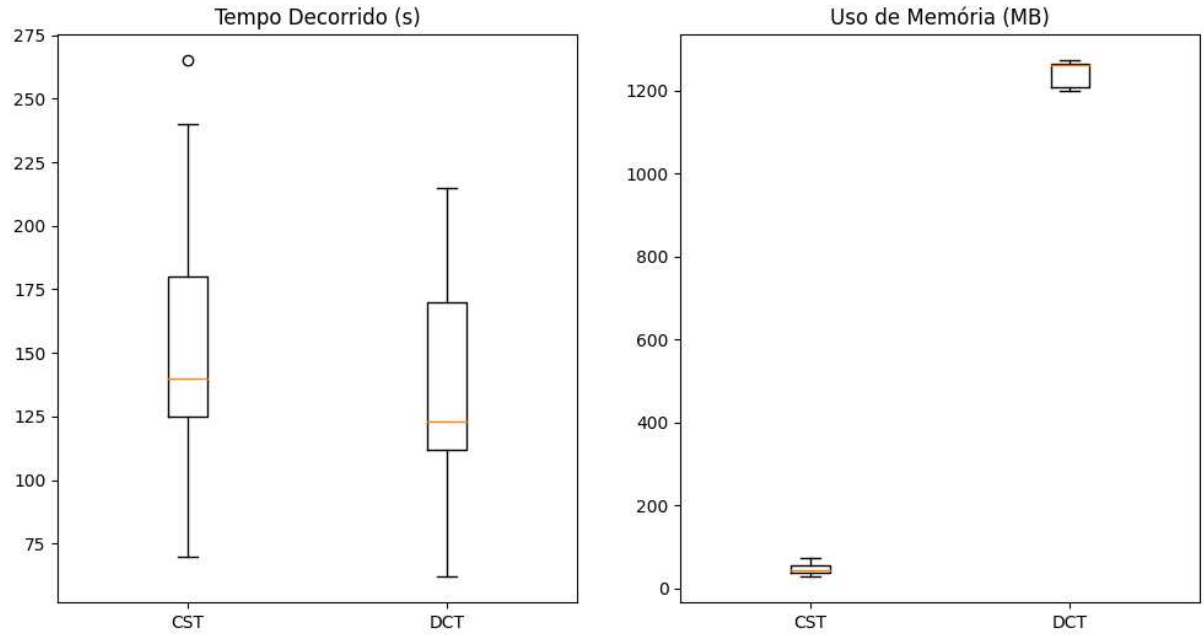


Figura 51 – Histogramas de tempo e consumo de memória para cada versão do agente. Note que, apesar do desempenho em tempo ser muito próximo, o DCT consumiu muito mais memória

6.4.3 Resultados Experimento 3

Utilizando um cenário com dados sintéticos, porém fundamentados na literatura, os experimentos apresentam alguns resultados interessantes. As figuras 52 e 53 mostram, respectivamente, histogramas de pontuação final após o processo de treinamento completo e número de gerações que o processo levou. Observe que a maioria das execuções parou em 20 gerações (máximo) e não pode melhorar ainda mais, como sugere a Figura 53. Apesar disso, como pode ser visto na Fig. 52, mais de 80% das execuções terminaram com pontuação igual ou inferior a 2, significando no máximo duas ativações erradas do dispositivo em 260 interações.

As figuras 54 e 55 refletem o número de Codelets Comportamentais e Perceptuais, respectivamente, para obter o melhor resultado em cada execução. Como podemos observar na Fig. 54, a maioria das execuções necessitava de 13 Codelets Comportamentais, um para cada Codelet/dispositivo de atuação do Motor. A Fig. 56a mostra a correlação entre o número de codelets comportamentais e a pontuação. A resposta do sistema é melhor (menor pontuação) à medida que mais Codelets Comportamentais são usados. O número de Codelets Perceptuais, por outro lado, mostra distribuições normais aproximadas com um leve viés à direita, o que significa que a incorporação pode variar e a saída ainda ser boa. Esse viés se reflete na ligeira correlação negativa do número de Codelets Perceptuais e Score (menor que em relação aos Codelets Comportamentais).

A tabela 7 mostra algumas estatísticas retiradas dos experimentos. Observe que,

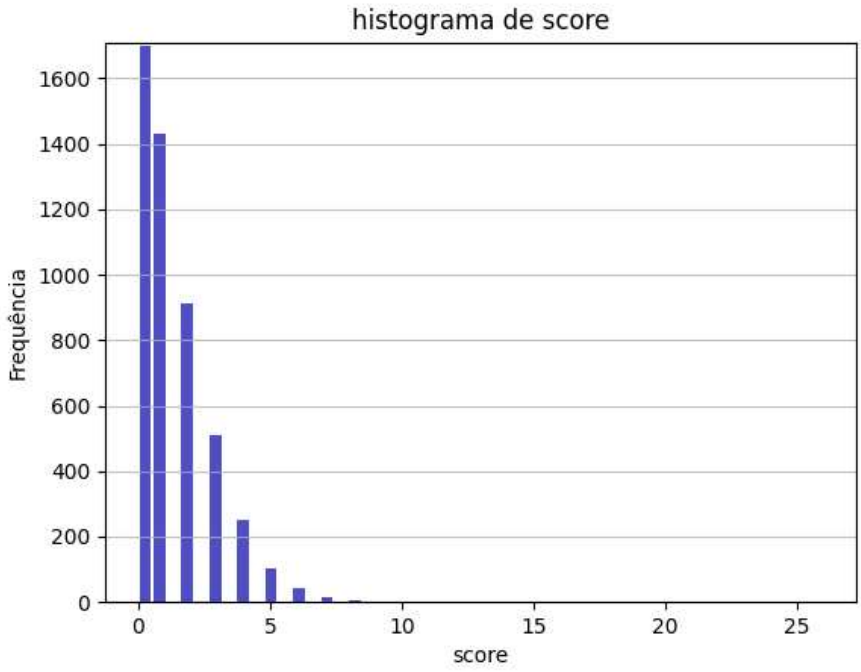


Figura 52 – Histograma de *score* obtido em execuções de experimentos. Observe que quanto menor, melhor

enquanto o número individual de Codelet Perceptual e Comportamental pode chegar a 2, os Codelets "Internos" combinados precisam de um número maior para apresentar resultados.

Tabela 7 – Métricas sobre os Experimentos 3

	média	mediana	desvio padrão	mínimo	máximo
score	1.438	1.0	1.894	0	26
generações	13.6784	20.0	8.905	0	20
nº Perceptuais	9.5326	10.0	2.213	2	15
nº Behaviorals	11.2674	13.0	2.478	2	13
nº Internals	20.8	22.0	3.998	7	28

Como os resultados dos experimentos sugerem, é possível e uma boa opção combinar dispositivos simples (físicos e virtuais) para construir um *Cognitive Twin* de um agente. É claro que o método pode ser aprimorado, mas com apenas métodos simples de *Machine Learning* em cada codelet e reconfiguração de topologia, podemos obter um comportamento geral bastante semelhante a um agente primordial.

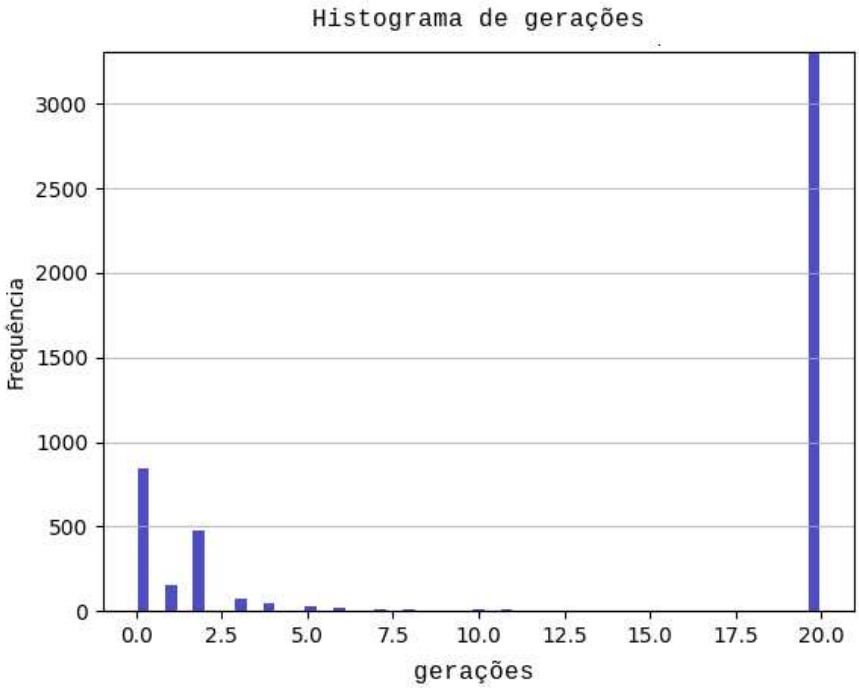


Figura 53 – Histograma de *gerações* necessário para atingir a pontuação mais baixa. Aqui, a maioria das execuções precisou do número máximo de gerações, o que indica que pode ser necessário um número maior de gerações.

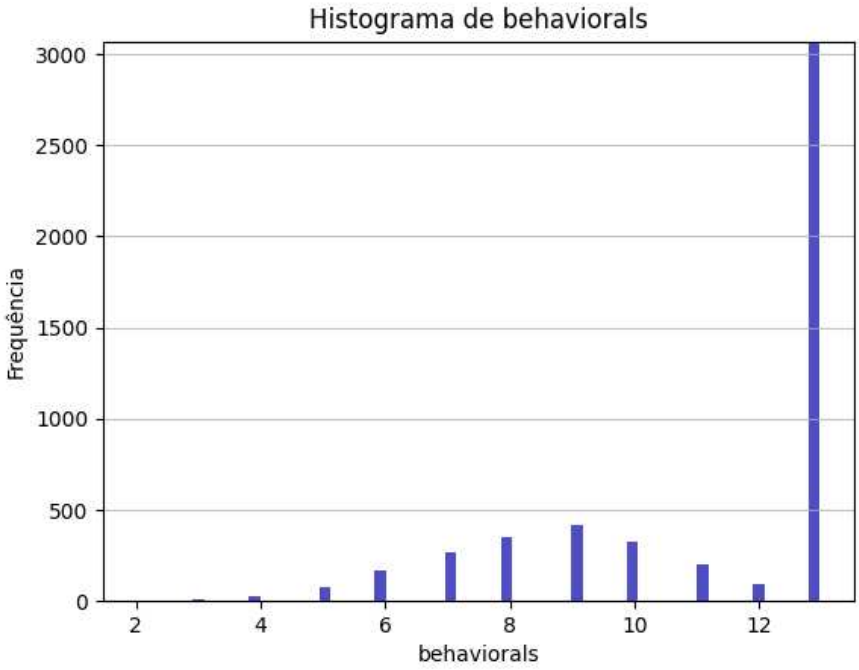


Figura 54 – Histograma do número de Codelets Comportamentais necessários para atingir a pontuação mais baixa. A maioria das corridas precisava de 13, o mesmo número de Codelets de Motor (e atuadores)

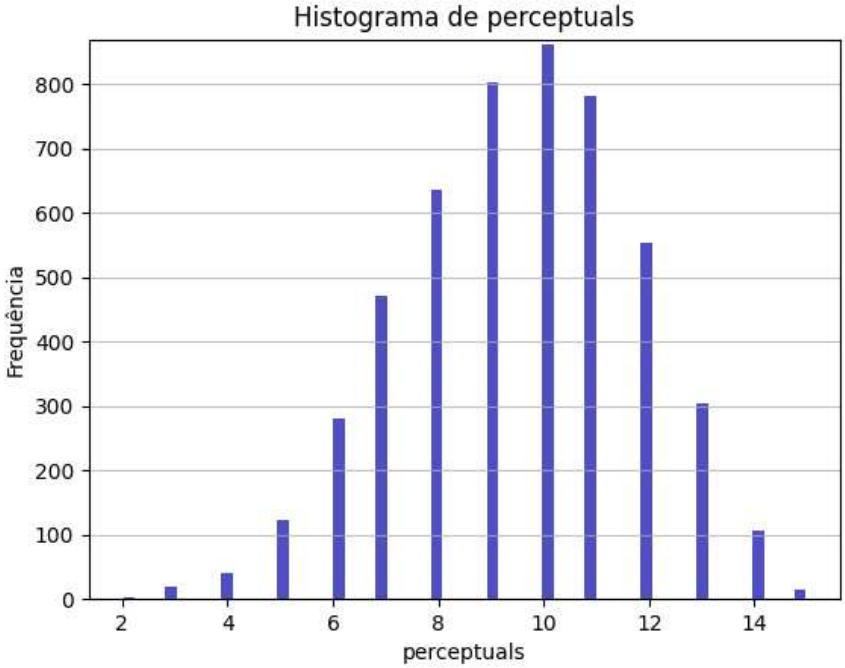


Figura 55 – Histograma do número de Codelets Perceptuais necessários para atingir a pontuação mais baixa. Note que o gráfico assemelha-se a uma gaussiana levemente enviesada para a direita.

(a) Correlação entre o número de Codelets Comportamentais e Score.

(b) Correlação entre o número de Codelets Perceptuais e Score.

Figura 56 – Correlação entre o número de Codelets ”internos” e Score.

6.5 Discussão

Os três experimentos propostos tiveram como objetivo mostrar um pouco do potencial de se ter um sistema cognitivo distribuído. No primeiro experimento, mostramos como o design de um sistema com uma abordagem distribuída utilizando o DCT é resiliente, isto é, consegue se recuperar de falhas e ataques a seus componentes. Mais especificamente, a propriedade de autocorreção do DCT permite que o sistema recupere seu desempenho — completo ou parcial — mesmo após alguns de seus nós terem processos internos derrubados. O tempo necessário para recuperação mostrou-se proporcional ao percentual de nós que falharam a cada rodada e inversamente proporcional ao tempo de intervalo entre cada rodada. Nota-se ainda uma proporcionalidade em relação ao tamanho da rede neste tempo de recuperação. É interessante ainda notar que, com exceção dos piores cenários (rede com 20 nós, curto intervalo de tempo e alto percentual de falhas), o sistema conseguiu recuperar seu desempenho nominal dentro do tempo máximo de cada execução do experimento.

No segundo experimento mostramos que o desempenho na execução de tarefas de um agente distribuído é equivalente a um construído de forma centralizada. O uso de memória, no entanto, mostrou-se muito maior, que já era esperado devido às funcionalidades de um Node DCT, como o acesso via chamadas de API e a já citada autocorreção. Cada Node consumiu, em média, 310 MB de memória (calculados utilizando as ferramentas do Docker), quase 7 vezes mais que a aplicação completa da versão com o CST. Apesar do experimento ter sido executado em contêineres docker em uma única máquina, cada um destes Nodes poderia estar em um dispositivo físico diferente, distribuindo assim o custo computacional da aplicação e paralelizando as etapas do ciclo cognitivo.

No terceiro e último experimento mostramos uma aplicação que faz uso direto do benefício de ser distribuído: como as conexões entre os Nodes podem ser reconfiguradas, é possível reutilizar Nodes já existentes para formar novos arranjos e se ter uma variedade de agentes (ou candidatos a) com um número fixo de dispositivos (físicos ou virtuais). A utilização de uma Estratégia Evolutiva como método de definição da topologia do agente e de um treinamento supervisionado como otimizador de cada candidato permitiu a desoneração de se estabelecer uma configuração *a priori* para o agente, incluindo especificidades internas de cada Codelet. Isto é, ao invés de definir cada Codelet perceptual e comportamental, definindo o fluxo de informações através do agente no momento do design, foi possível estabelecer um *Cognitive Twin* a partir de um reservatório de Codelets geradas de maneira estocástica a partir de uma determinada heurística. No entanto, devido ao alto número de rodadas em que o número máximo de gerações foi atingido, é possível perceber que um limiar mais alto é necessário.

Pode-se elencar algumas vantagens em se ter um Sistema Cognitivo Distribuído

com base nos argumentos teóricos trazidos e experimentos realizados:

- Flexibilidade e Escalabilidade: Sistemas distribuídos, como os baseados em DCT, podem se adaptar facilmente a mudanças nos requisitos de processamento e na quantidade de dados a serem processados. Eles permitem a inclusão ou remoção de nós de processamento conforme necessário sem interromper o funcionamento do sistema;

- Robustez e Resiliência: Um sistema cognitivo distribuído pode oferecer maior robustez contra falhas de hardware ou software. Se um nó falha, outros nós podem assumir suas funções, mantendo o sistema operacional. Isso é crucial para aplicações críticas onde a continuidade do serviço é vital;

- Interoperabilidade e Integração: O uso do DCT facilita a integração de diferentes dispositivos e plataformas, promovendo uma interação mais sofisticada com o ambiente. Isso é particularmente importante no contexto da Internet das Coisas Cognitiva, onde a capacidade de operar em um ecossistema heterogêneo é uma vantagem significativa;

- Processamento Paralelo e Eficiência: A arquitetura distribuída permite o processamento paralelo de tarefas, o que pode levar a uma eficiência significativamente maior em comparação com sistemas centralizados. Isso é especialmente relevante para tarefas de computação intensiva, como o processamento de grandes conjuntos de dados ou a realização de operações complexas de simulação.

De forma geral, a implementação em contêineres Docker não apresentou problemas relacionados a latência pois, devido ao uso da rede interna do computador físico, os atrasos de comunicação foram desprezíveis.

A escolha de um *Digital Twin* como aplicação se deu num oportuno momento — levemente anterior — em que outras áreas da Inteligência Artificial debruçam-se sobre a criação de comportamento humano artificial crível, permitindo que a abordagem aqui presente seja futuramente combinada com outras presentes na Literatura.

6.6 Resumo do Capítulo

Neste Capítulo falamos sobre como os experimentos foram desenvolvidos, as considerações tomadas em seu design, as métricas, os resultados, bem como uma pequena discussão sobre os mesmos.

7 Conclusão e Trabalhos Futuros

7.1 Conclusão

Este trabalho visou aplicar conceitos de Internet das Coisas, Sistemas de Sistemas, Sistemas Ciber-Físicos e Arquiteturas Cognitivas para a implementação de um método de construção de um *Cognitive Twin* que possa ser treinado de "ponta a ponta" com dados de interação de um agente — humano ou não — com sensores e atuadores em um ambiente inteligente e que seja capaz de mimetizar o comportamento de tal agente. Idealmente, este *Cognitive Twin* pode ser usado em diversas aplicações, como controle automatizado de casas inteligentes, a criação de autômatos com assistentes pessoais digitais, a criação de agentes inteligentes com base em conhecimento especialista (para controle de veículos, por exemplo), e simulação de personagens ou usuários com comportamento verossímil, entre outras.

Para atingir esse objetivo, foi desenvolvido um *framework* de desenvolvimento de arquiteturas cognitivas distribuídas, o DCT, e uma implementação na linguagem de programação Python. Esse *framework* serviu de base para, em conjunto com métodos de Aprendizado de Máquina como árvores de decisão e estratégias evolutivas, construir um *Cognitive Twin* cuja topologia do agente e modelo interno de cada Codelet treinável foram totalmente definidos pelas meta-heurísticas sem a necessidade de design especial ou seleção prévia de quais *features* serão ou não consideradas na construção do modelo final.

Embora não tenham sido utilizados dados reais devido à dificuldade de encontrar datasets públicos do tipo, foram gerados dados sintéticos cuidadosamente para que apresentassem verossimilhança com dados que poderiam ser coletados da interação de um ser humano com um ambiente sensoreado. Isso foi importante para garantir que o modelo fosse treinado com dados que se assemelham à realidade e, portanto, tivesse um comportamento mais próximo ao de um agente real.

Os resultados apresentados demonstram o potencial da utilização de grupos de dispositivos de baixo poder computacional orquestrados em conjunto para mimetizar o comportamento de um agente real sem a necessidade de um dispositivo centralizado de alto poder computacional. Além disso, o modelo apresenta uma robustez devido às suas capacidades de autorregeneração e reconfiguração.

É importante destacar que, embora os resultados obtidos tenham sido promissores, ainda há muito a ser feito para aprimorar nosso modelo de *Cognitive Twin*, em especial para que possa servir de material de debate na comunidade acadêmica em dire-

ção a modelos unificados. Um dos principais desafios é encontrar datasets públicos de alta qualidade que possam ser usados para treinar o modelo com dados reais. Além disso, é necessário aprimorar os métodos de treinamento e validação do modelo para garantir que ele tenha um comportamento mais preciso e consistente.

Por fim, é importante ressaltar as implicações éticas de copiar um agente de forma digital. Os dados de pessoas reais estão sujeitos a esferas legais, éticas e morais sobre a validade ou não de sua utilização. No entanto, acredita-se que as abordagens aqui apresentadas possam ter um grande impacto em diversas áreas, desde o controle automatizado de casas inteligentes até a criação de autômatos com assistentes pessoais digitais e a simulação de personagens ou usuários com comportamento verossímil.

7.2 Limitações

As limitações da proposta apresentada no Capítulo 5 envolvem principalmente as restrições técnicas e conceituais inerentes ao modelo de *Cognitive Twin* Evolutivo utilizando sistemas distribuídos. Essas limitações podem ser categorizadas da seguinte forma:

- A proposta assume o uso de dispositivos simples e de baixo poder computacional. Isso implica limitações na capacidade de processamento e armazenamento, o que pode restringir a complexidade dos modelos cognitivos que podem ser implementados nos dispositivos.
- A proposta também assume que seria impraticável ter todos os dispositivos totalmente conectados devido à sobrecarga de comunicação que isso implicaria. Essa limitação afeta a capacidade do sistema de compartilhar informações em tempo de execução e coordenar ações entre dispositivos de forma eficiente, especialmente em sistemas físicos onde a comunicação contínua é fundamental para operações sincronizadas.

Estas limitações destacam desafios significativos no design e na implementação de Sistemas Cognitivos distribuídos, particularmente em relação à seleção de hardware, gestão de recursos, estratégias de aprendizado e adaptação, e otimização da comunicação entre dispositivos.

Em relação ao DCT, não foi tratado neste trabalho possíveis problemas de “condições de corrida” em relação à escrita e leitura de Memórias, isto é, não são tratados problemas de dados esperados incorretos devido a leituras e escritas concomitantes com ordem imprevisível.

7.3 Contribuições deste Trabalho

As principais contribuições deste trabalho podem ser sumarizadas da seguinte maneira:

- **Revisão da literatura sobre Internet das Coisas, Sistemas Ciber-Físicos e Sistemas de Sistemas**: Esta tese apresenta uma revisão bibliográfica atualizada de tópicos relevantes ao problema da construção de um *Cognitive Twin*, tema principal desta tese, ou seja, a Internet das Coisas, os Sistemas Ciber-Físicos e os Sistemas de Sistemas, trazendo uma visão introdutória que pode ser útil a um leitor que não esteja atualizado sobre esses temas;

- **Especificação de um *Framework* para Sistemas Cognitivos Distribuídos (DCT)**: Do ponto de vista teórico, desenvolvemos a especificação de uma ferramenta multi-linguagem projetada como um *framework* para a criação de Sistemas Cognitivos Distribuídos, permitindo a operação em múltiplos dispositivos físicos e virtuais (como desktops, Raspberries, Arduínos, e contêineres Docker), a qual demos o nome de DCT (Distributed Cognitive Toolkit). O objetivo do DCT é facilitar a construção de aplicativos cognitivos distribuídos, onde *Nodes* contendo *Codelets* e Memórias comunicam-se entre si. O DCT integra servidores que atendem a requisições REST para interconexão de sistemas e subsistemas. Sobre essa ferramenta, os seguintes pontos merecem destaque:

  - **Orquestração de múltiplos dispositivos**: O DCT possui a capacidade de orquestrar diversos dispositivos, tanto físicos quanto virtuais, colocando-os para operar de maneira integrada;
  - **Independência de Linguagem de Implementação**: O DCT foi concebido de modo independente de linguagem, ou seja, desde que sejam atendidos os requisitos de interface e protocolos de comunicação, os *Nodes* de um sistema DCT podem ser implementados em qualquer linguagem;
  - **Arquitetura Modular**: Um Node no DCT é uma abstração para um subsistema virtual contendo um número arbitrário de Codelets e/ou Memórias que fazem parte da arquitetura cognitiva.

  Essas características tornam o DCT uma ferramenta versátil e poderosa para o desenvolvimento de sistemas cognitivos distribuídos, com amplas possibilidades de aplicação em diversos dispositivos e plataformas.

- **Implementação de um protótipo do DCT em linguagem Python**: Construímos, utilizando a linguagem Python, um protótipo da especificação do DCT, com a

qual pudemos desenvolver alguns testes e os experimentos desta tese. O desenvolvimento desse protótipo foi importante no aprimoramento e consolidação da própria especificação;

- **Desenvolvimento de um experimento, utilizado como prova de conceito para a capacidade de autorregeneração do DCT**: O experimento relatado na seção 6.1 foi desenvolvido com o objetivo de avaliar a capacidade de autorregeneração, servindo como uma prova de conceito que demonstra a resiliência dos sistemas construídos com o DCT, exercitando sua capacidade de autorregeneração;

- **Desenvolvimento de um experimento, utilizado como prova de conceito da possibilidade da utilização do DCT para a construção de arquiteturas cognitivas distribuídas**: O experimento relatado na seção 6.2 foi desenvolvido com o intuito de demonstrar que, de modo efetivo, o DCT poderia ser utilizado na construção de uma arquitetura cognitiva distribuída funcional. Neste caso, desenvolvemos uma versão distribuída de uma Arquitetura Cognitiva que havia sido criada anteriormente com o CST, para o controle de um robô autônomo em um ambiente virtual, com resultados semelhantes aos obtidos a uma versão monolítica, mostrando a viabilidade do uso de uma arquitetura distribuída, com propósitos semelhantes, sem perda significativa de desempenho;

- **Proposta de um método de construção de *Cognitive Twins* utilizando uma arquitetura cognitiva evolutiva**: O capítulo 3 propõe uma técnica evolutiva para a construção de *Cognitive Twins* que dispensa a necessidade de design específico da topologia do agente pelo programador. Esses sistemas são baseados em dados de interação de um agente externo, que utilizam treinamento supervisionado (input-output) e uma Estratégia Evolutiva em um framework para Arquiteturas Cognitivas Distribuídas;

- **Desenvolvimento de um experimento, utilizado como prova de conceito para o método de construção de *Cognitive Twins*, aplicado à automação residencial**: Utilizando o método de construção de *Cognitive Twins*, a seção 6.3 apresenta um experimento, onde um *Cognitive Twin* foi construído, utilizando o DCT, de modo a imitar a interação de um usuário artificial com dispositivos de uma residência automatizada, demonstrando a viabilidade do método.

Dessa forma, entendemos que a tese envolve diversas contribuições transversais, que merecem seu destaque, ao longo desse trabalho. Essas contribuições são significativas para o campo de Inteligência Artificial baseados em Agentes, especialmente no que diz respeito à criação de Sistemas Cognitivos e a simulação de comportamentos humanos por meio de tecnologias avançadas.

7.4 Trabalhos Futuros

Em trabalhos futuros, buscaremos melhorar nossos experimentos relacionados ao Cognitive Twin, modificando-os de modo a permitir mais gerações, capturar dados de interação reais, usando dispositivos físicos, para desenvolver ainda mais o método e entender melhor o papel do *Cognitive Twin* na melhoria da sociedade, tanto ajudando as pessoas como assistentes quanto fornecendo insights sobre o comportamento humano. Buscaremos também explorar diferentes métodos de representação dos dados sensoriais e perceptuais, visto que a representação do conhecimento é um ponto importante para a performance de sistemas de Inteligência Artificial. Por fim, modelos atencionais e sistemas neurosimbólicos destacam-se como possíveis bons caminhos para melhoria do que foi aqui apresentado

Referências

ABBURU, S.; BERRE, A.; JACOBY, M.; ROMAN, D.; STOJANOVIC, L.; STOJANOVIC, N. Cognitive digital twins for the process industry. In: *Proceedings of the The Twelfth International Conference on Advanced Cognitive Technologies and Applications (COGNITIVE 2020), Nice, France*. [S.l.: s.n.], 2020. p. 25–29. Citado 5 vezes nas páginas , 63, 64, 65 e 76.

ABBURU, S.; BERRE, A. J.; JACOBY, M.; ROMAN, D.; STOJANOVIC, L.; STOJANOVIC, N. Cognitwin–hybrid and cognitive digital twins for the process industry. In: IEEE. *2020 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. [S.l.], 2020. p. 1–8. Citado 6 vezes nas páginas , 63, 65, 72, 74 e 76.

ALLENDER, L. Modeling human performance: impacting system design, performance, and cost. *Simulation Series*, Citeseer, v. 32, n. 3, p. 139–144, 2000. Citado na página 29.

ALUR, R.; BERGER, E.; DROBNIS, A. W.; FIX, L.; FU, K.; HAGER, G. D.; LOPRESTI, D.; NAHRSTEDT, K.; MYNATT, E.; PATEL, S. *et al.* Systems computing challenges in the internet of things. *arXiv preprint arXiv:1604.02980*, 2016. Citado 3 vezes nas páginas 36, 37 e 38.

ANDERSON, J. R. A theory of the origins of human knowledge. *Artificial intelligence*, Elsevier, v. 40, n. 1-3, p. 313–351, 1989. Citado na página 27.

ANDERSON, J. R. *How can the human mind occur in the physical universe?* [S.l.]: Oxford University Press, 2009. Citado 2 vezes nas páginas 27 e 76.

ANDERSON, J. R.; BOTHELL, D.; BYRNE, M. D.; DOUGLASS, S.; LEBIERE, C.; QIN, Y. An integrated theory of the mind. *Psychological review*, American Psychological Association, v. 111, n. 4, p. 1036, 2004. Citado 4 vezes nas páginas , 27, 28 e 76.

ARUP. *Digital Twin: Towards a meaningful framework*. 2019. Disponível em: <https://www.arup.com/perspectives/publications/research/section/digital-twin-towards-a-meaningful-framework>. Citado na página 62.

ATKINS, E. M. Cyber-physical aerospace: Challenges and future directions in transportation and exploration systems. In: *National Science Foundation Workshop on Smart Transportation and Aviation*. [S.l.: s.n.], 2006. Citado na página 53.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer networks*, Elsevier, v. 54, n. 15, p. 2787–2805, 2010. Citado 3 vezes nas páginas , 31 e 32.

AZEEZ, A.; PERERA, S.; GAMAGE, D.; LINTON, R.; SIRIWARDANA, P.; LEELARATNE, D.; WEERAWARANA, S.; FREMANTLE, P. Multi-tenant soa middleware for cloud computing. In: IEEE. *2010 IEEE 3rd International Conference on Cloud Computing*. [S.l.], 2010. p. 458–465. Citado na página 37.

BAARS, B. J. *A Cognitive Theory of Consciousness*. [S.l.]: Cambridge University Press, 1988. ISBN 0-521-30133-5. Citado na página 80.

BAARS, B. J.; FRANKLIN, S. An architectural model of conscious and unconscious brain functions: Global workspace theory and ida. *Neural Networks*, Elsevier, v. 20, n. 9, p. 955–961, 2007. Citado na página 80.

BALAJI, B.; FARUQUE, A.; ABDULLAH, M.; DUTT, N.; GUPTA, R.; AGARWAL, Y. Models, abstractions, and architectures: the missing links in cyber-physical systems. In: ACM. *Proceedings of the 52nd Annual Design Automation Conference*. [S.l.], 2015. p. 82. Citado na página 51.

BENDER, E. M.; GEBRU, T.; MCMILLAN-MAJOR, A.; SHMITCHELL, S. On the dangers of stochastic parrots: Can language models be too big? In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. [S.l.: s.n.], 2021. p. 610–623. Citado na página 25.

BEYER, H.-G.; ARNOLD, D. V. Theory of evolution strategies—a tutorial. *Theoretical aspects of evolutionary computing*, Springer, p. 109–133, 2001. Citado na página 105.

BOARDMAN, J.; SAUSER, B. System of systems-the meaning of of. In: IEEE. *2006 IEEE/SMC International Conference on System of Systems Engineering*. [S.l.], 2006. p. 6–pp. Citado 2 vezes nas páginas 41 e 44.

BOULDING, K. E. General systems theory—the skeleton of science. *Management Science*, v. 2, n. 3, p. 197–208, 1956. Disponível em: <http://dx.doi.org/10.1287/mnsc.2.3.197>. Citado na página 39.

BREIMAN, L. *Classification and regression trees*. [S.l.]: Routledge, 2017. Citado 2 vezes nas páginas 108 e 109.

BRIA, F.; MOROZOV, E. *A cidade inteligente: tecnologias urbanas e democracia*. [S.l.]: Ubu Editora, 2020. Citado na página 35.

BROOKS, R. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, IEEE, v. 2, n. 1, p. 14–23, 1986. Citado na página 103.

BROOKS, R. A. A robot that walks; emergent behaviors from a carefully evolved network. *Neural computation*, MIT Press, v. 1, n. 2, p. 253–262, 1989. Citado na página 29.

CASTRO, L. N. D. *Fundamentals of natural computing: basic concepts, algorithms, and applications*. [S.l.]: CRC Press, 2006. Citado 2 vezes nas páginas 94 e 105.

CAVOUKIAN, A. *Privacy and drones: Unmanned aerial vehicles*. [S.l.]: Information and Privacy Commissioner of Ontario, Canada Ontario, Canada, 2012. Citado na página 53.

CHAN, M.; ESTÈVE, D.; FOURNIOLS, J.-Y.; ESCRIBA, C.; CAMPO, E. Smart wearable systems: Current status and future challenges. *Artificial intelligence in medicine*, Elsevier, v. 56, n. 3, p. 137–156, 2012. Citado na página 34.

COLE, R. The changing role of requirements and architecture in systems engineering. In: IEEE. *2006 IEEE/SMC International Conference on System of Systems Engineering*. [S.l.], 2006. p. 5–pp. Citado na página 44.

CONTI, M.; DAS, S. K.; BISDIKIAN, C.; KUMAR, M.; NI, L. M.; PASSARELLA, A.; ROUSSOS, G.; TRÖSTER, G.; TSUDIK, G.; ZAMBONELLI, F. Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber–physical convergence. *Pervasive and Mobile Computing*, Elsevier, v. 8, n. 1, p. 2–21, 2012. Citado na página 49.

CUI, Y.; AHMAD, S.; HAWKINS, J. Continuous online sequence learning with an unsupervised neural network model. *Neural computation*, MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , v. 28, n. 11, p. 2474–2504, 2016. Citado na página 29.

DELAURENTIS, D. Understanding transportation as a system-of-systems design problem. In: RENO, NV NEW YORK, NY. *43rd AIAA Aerospace Sciences Meeting and Exhibit*. [S.l.], 2005. v. 1. Citado na página 45.

DIMARIO, M. J. System of systems interoperability types and characteristics in joint command and control. In: IEEE. *System of Systems Engineering, 2006 IEEE/SMC International Conference on*. [S.l.], 2006. p. 236–241. Citado 2 vezes nas páginas 41 e 44.

DOMINGUE, J.; ZAHARIADIS, T.; LAMBERT, D.; CLEARY, F.; DARAS, P.; KRCO, S.; LI, M.-S.; SCHAFFERS, H.; LOTZ, V.; STILLER, B. *et al. The Future Internet*. [S.l.]: Springer Berlin Heidelberg„ 2011. Citado na página 36.

DU, J.; ZHU, Q.; SHI, Y.; WANG, Q.; LIN, Y.; ZHAO, D. Cognition digital twins for personalized information systems of smart cities: Proof of concept. *Journal of Management in Engineering*, American Society of Civil Engineers, v. 36, n. 2, p. 04019052, 2020. Citado 3 vezes nas páginas 70, 72 e 77.

EIRINAKIS, P.; KALABOUKAS, K.; LOUNIS, S.; MOURTOS, I.; ROŽANEC, J. M.; STOJANOVIC, N.; ZOIS, G. Enhancing cognition for digital twins. In: IEEE. *2020 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. [S.l.], 2020. p. 1–7. Citado 4 vezes nas páginas , 66, 68 e 72.

EISNER, H.; MARCINIAK, J.; MCMILLAN, R. Computer-aided system of systems (s2) engineering. In: IEEE. *Systems, Man, and Cybernetics, 1991.'Decision Aiding for Complex Systems, Conference Proceedings., 1991 IEEE International Conference on*. [S.l.], 1991. p. 531–537. Citado na página 40.

EISNER, H.; MCMILLAN, R.; MARCINIAK, J.; PRAGLUSKI, W. Rcasse: Rapid computer-aided system of systems (s2) engineering. In: WILEY ONLINE LIBRARY. *INCOSE International Symposium*. [S.l.], 1993. v. 3, n. 1, p. 267–273. Citado na página 40.

ENGELMANN, K. F.; HOLTHAUS, P.; WREDE, B.; WREDE, S. An interaction-centric dataset for learning automation rules in smart homes. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. [S.l.: s.n.], 2016. p. 1432–1437. Citado na página 117.

EVERTSZ, R.; PEDROTTI, M.; BUSETTA, P.; ACAR, H.; RITTER, F. E. Populating vbs2 with realistic virtual actors. In: *Proceedings of the 18th conference on behavior representation in modeling and simulation*. [S.l.: s.n.], 2009. p. 1–8. Citado na página 29.

FAN, X.; MCNEESE, M.; YEN, J. Ndm-based cognitive agents for supporting decision-making teams. *Human–Computer Interaction*, Taylor & Francis, v. 25, n. 3, p. 195–234, 2010. Citado na página 29.

FERNÁNDEZ, F.; SÁNCHEZ, Á.; VÉLEZ, J. F.; MORENO, A. B. Symbiotic autonomous systems with consciousness using digital twins. In: SPRINGER. *International Work-Conference on the Interplay Between Natural and Artificial Computation*. [S.l.], 2019. p. 23–32. Citado 5 vezes nas páginas , 64, 66, 67 e 72.

FOTEINOS, V.; KELAIDONIS, D.; POULIOS, G.; VLACHEAS, P.; STAVROULAKI, V.; DEMESTICHAS, P. Cognitive management for the internet of things: A framework for enabling autonomous applications. *IEEE Vehicular Technology Magazine*, v. 8, n. 4, p. 90–99, Dec 2013. ISSN 1556-6072. Citado 3 vezes nas páginas , 57 e 58.

FRANKLIN, S.; GRAESSER, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. In: MÜLLER, J. P.; WOOLDRIDGE, M. J.; JENNINGS, N. R. (Ed.). *Intelligent Agents III Agent Theories, Architectures, and Languages*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997. p. 21–35. ISBN 978-3-540-68057-4. Citado na página 30.

FRANKLIN, S.; KELEMEN, A.; MCCAULEY, L. Ida: A cognitive agent architecture. In: IEEE. *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*. [S.l.], 1998. v. 3, p. 2646–2651. Citado na página 80.

FRANKLIN, S.; MADL, T.; D'MELLO, S.; SNAIDER, J. Lida: A systems-level architecture for cognition, emotion, and learning. *Autonomous Mental Development, IEEE Transactions on*, IEEE, v. 6, n. 1, p. 19–41, 2014. Citado 4 vezes nas páginas 21, 79, 80 e 82.

GIBAUT, W.; GUDWIN, R. Using environment exploration for inverse kinematics learning. *Memory*, v. 2, p. E1, 2019. Citado 2 vezes nas páginas 29 e 161.

GIBAUT, W.; GUDWIN, R. Extending the cst: The distributed cognitive toolkit. In: IEEE. *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*. [S.l.], 2020. p. 474–481. Citado 9 vezes nas páginas , 22, 82, 83, 84, 89, 90, 91 e 161.

GIBAUT, W.; OSORIO, A.; MUNOZ, A.; NETO, S. F.; MIRANDA, D.; SANTOS, F.; SCARASSATI, M.; GRASSIOTTO, F. Toward a federated model for human context recognition on edge devices. 2022. Citado na página 162.

GIBAUT, W. S. P. Uma arquitetura cognitiva para o aprendizado instrumental em agentes inteligentes. [sn], 2018. Citado na página 29.

GIFFINGER, R.; GUDRUN, H. Smart cities ranking: an effective instrument for the positioning of the cities? *ACE: Architecture, City and Environment*, Centre de Política del Sòl i Valoracions-Universitat Politècnica de Catalunya, v. 4, n. 12, p. 7–26, 2010. Citado 2 vezes nas páginas 34 e 36.

GOERTZEL, B.; PENNACHIN, C.; GEISWEILLER, N. Brief survey of cognitive architectures. In: *Engineering General Intelligence, Part 1*. [S.l.]: Springer, 2014. p. 101–142. Citado na página 29.

GOROD, A.; SAUSER, B.; BOARDMAN, J. System-of-systems engineering management: A review of modern history and a path forward. *Systems Journal, IEEE*, IEEE, v. 2, n. 4, p. 484–499, 2008. Citado 5 vezes nas páginas , 39, 40, 41 e 44.

GOROD, A.; WHITE, B. E.; IRELAND, V.; GANDHI, S. J.; SAUSER, B. *Case Studies in System of Systems, Enterprise Systems, and Complex Systems Engineering*. [S.l.]: CRC Press, 2014. Citado 3 vezes nas páginas , 40 e 41.

GPO, U. Restructuring of the strategic defense initiative (sdi) program. In: *Services, USCS C. o. A.(Ed.), United States Congress*. [S.l.: s.n.], 1989. p. 16. Citado na página 39.

GREGOR, D.; TORAL, S.; ARIZA, T.; BARRERO, F.; GREGOR, R.; RODAS, J.; ARZAMENDIA, M. A methodology for structured ontology construction applied to intelligent transportation systems. *Computer Standards & Interfaces*, Elsevier, v. 47, p. 108–119, 2016. Citado na página 54.

GRIEVES, M. Digital twin: manufacturing excellence through virtual factory replication. *White paper*, Florida Institute of Technology, v. 1, n. 2014, p. 1–7, 2014. Citado na página 62.

GRINSZTAJN, L.; OYALLON, E.; VAROQUAUX, G. Why do tree-based models still outperform deep learning on tabular data? *arXiv preprint arXiv:2207.08815*, 2022. Citado na página 98.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, Elsevier, v. 29, n. 7, p. 1645–1660, 2013. Citado 2 vezes nas páginas 35 e 39.

GUDWIN, R.; PARAENSE, A.; PAULA, S. M. de; FRÓES, E.; GIBAUT, W.; CASTRO, E.; FIGUEIREDO, V.; RAIZER, K. The multipurpose enhanced cognitive architecture (meca). *Biologically Inspired Cognitive Architectures*, Elsevier, v. 22, p. 20–34, 2017. Citado 9 vezes nas páginas , 21, 22, 80, 82, 93, 96, 97 e 99.

GUDWIN, R.; PARAENSE, A.; PAULA, S. M. de; FRÓES, E.; GIBAUT, W.; CASTRO, E.; FIGUEIREDO, V.; RAIZER, K. An urban traffic controller using the MECA cognitive architecture. *Biologically inspired cognitive architectures*, Elsevier, v. 26, p. 41–54, 2018. Citado 2 vezes nas páginas 29 e 80.

GUDWIN, R.; ROHMER, E.; PARAENSE, A.; FRÓES, E.; GIBAUT, W.; OLIVEIRA, I.; ROCHA, S.; RAIZER, K.; FELJAN, A. V. The TROCA project: An autonomous transportation robot controlled by a cognitive architecture. *Cognitive Systems Research*, Elsevier, v. 59, p. 179–197, 2020. Citado 3 vezes nas páginas 29, 80 e 162.

GUDWIN, R.; ROHMER, E.; PARAENSE, A.; FRÓES, E.; GIBAUT, W.; RAIZER, K.; FELJAN, A. V. A double-layer subsumption mechanism for enforcing sequential behaviors in a cognitive architecture. In: IEEE. *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. [S.l.], 2021. p. 1–7. Citado na página 162.

GUDWIN, R.; ROHMER, E.; PARAENSE, A. L. O.; FRÓES, E.; GIBAUT, W.; OLIVEIRA, I.; ROCHA, S.; RAIZER, K.; FELJAN, A. V. A cognitive architecture for a transportation robotic system. In: SPRINGER. *Biologically Inspired Cognitive Architectures Meeting*. [S.l.], 2019. p. 110–116. Citado na página 162.

GUNES, V.; PETER, S.; GIVARGIS, T.; VAHID, F. A survey on concepts, applications, and challenges in cyber-physical systems. *TIIS*, v. 8, n. 12, p. 4242–4268, 2014. Citado 5 vezes nas páginas , 51, 52, 53 e 54.

HÅKANSSON, A.; HARTUNG, R.; MORADIAN, E. Reasoning strategies in smart cyber-physical systems. *Procedia Computer Science*, Elsevier, v. 60, p. 1575–1584, 2015. Citado na página 50.

HAMADI, Y.; JABBOUR, S.; SAIS, L. Learning for dynamic subsumption. *International Journal on Artificial Intelligence Tools*, World Scientific, v. 19, n. 04, p. 511–529, 2010. Citado na página 103.

HECKEL, F.; YOUNGBLOOD, G. Multi-agent coordination using dynamic behavior-based subsumption. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. [S.l.: s.n.], 2010. v. 6, n. 1, p. 132–137. Citado na página 103.

HEILER, S. Semantic interoperability. *ACM Computing Surveys (CSUR)*, ACM, v. 27, n. 2, p. 271–273, 1995. Citado na página 54.

HELJAKKA, A.; GOERTZEL, B.; SILVA, W.; PENNACHIN, C.; SENNA, A.; GOERTZEL, I. Probabilistic logic based reinforcement learning of simple embodied behaviors in a 3d simulation world. *Frontiers in Artificial Intelligence and Applications*, IOS Press, v. 157, p. 253, 2007. Citado na página 29.

HENDERSON, P.; HU, J.; ROMOFF, J.; BRUNSKILL, E.; JURAFSKY, D.; PINEAU, J. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, v. 21, n. 248, p. 1–43, 2020. Citado na página 25.

HINTON, G.; LECUNN, Y.; BENGIO, Y. *AAAI 20 / AAAI 2020 keynotes turing award winners event / Geoff Hinton, Yann LeCunn, Yoshua Bengio*. YouTube, 2020. Disponível em: <https://www.youtube.com/watch?v=UX8OubxsY8w>. Citado na página 21.

HOFSTADTER, D.; MITCHELL, M. The Copycat Project: A Model of Mental Fluidity and Analogy-making. In: _____. *Fluid Concepts And Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. 10, East 53rd Street, New York, NY 10022-5299: BasicBooks, 1994. p. 205–268. Citado na página 80.

HOLLAND, J. H. *Hidden order: How adaptation builds complexity*. [S.l.]: Basic Books, 1995. Citado na página 40.

HOLLER, J.; TSIATSIS, V.; MULLIGAN, C.; AVESAND, S.; KARNOUSKOS, S.; BOYLE, D. *From Machine-to-machine to the Internet of Things: Introduction to a New Age of Intelligence*. [S.l.]: Academic Press, 2014. Citado 2 vezes nas páginas e 33.

HORVATH, I.; GERRITSEN, B. H. Cyber-physical systems: Concepts, technologies and implementation principles. In: *Proceedings of TMCE*. [S.l.: s.n.], 2012. v. 1, p. 7–11. Citado na página 49.

HOWLEY, D. *These 169 industries are being hit by the Global Chip Shortage*. Yahoo!, 2021. Disponível em: <https://finance.yahoo.com/news/these-industries-are-hit-hardest-by-the-global-chip-shortage-122854251.html>. Citado na página 26.

HUNTSBERGER, T.; WOODWARD, G. Intelligent autonomy for unmanned surface and underwater vehicles. In: IEEE. *OCEANS'11 MTS/IEEE KONA*. [S.l.], 2011. p. 1–10. Citado na página 29.

JACKSON, M. C.; KEYS, P. Towards a system of systems methodologies. *The Journal of the Operational Research Society*, Palgrave Macmillan Journals, v. 35, n. 6, p. 473–486, 1984. Citado na página 39.

JACOB, F.; SPILLMANN, B. E. *La logique du vivant. The logic of living systems. A history of heredity. Translated by Betty E. Spillmann*. [S.l.]: Allen Lane, 1974. Citado na página 39.

JAMSHIDI, M. System of systems engineering-new challenges for the 21st century. *IEEE Aerospace and Electronic Systems Magazine*, IEEE, v. 23, n. 5, p. 4–19, 2008. Citado na página 46.

JAMSHIDI, M. *System of systems engineering: innovations for the twenty-first century*. [S.l.]: John Wiley & Sons, 2011. v. 58. Citado 6 vezes nas páginas , 41, 44, 45, 46 e 47.

JAYNES, E. T. Probability theory as logic. In: *Maximum entropy and Bayesian methods*. [S.l.]: Springer, 1990. p. 1–16. Citado na página 30.

JIANG, Y.; XIE, W.; WANG, F.; LI, N. An implementation of cognitive management framework for the internet of things system. In: *Information Technology and Electronic Commerce (ICITEC), 2014 2nd International Conference on*. [S.l.: s.n.], 2014. p. 103–106. Citado 3 vezes nas páginas , 58 e 60.

JING, Q.; VASILAKOS, A. V.; WAN, J.; LU, J.; QIU, D. Security of the internet of things: Perspectives and challenges. *Wireless Networks*, Springer, v. 20, n. 8, p. 2481–2501, 2014. Citado na página 37.

JOHNSON-LAIRD, P. Mental models in cognitive science. *Cognitive Science*, v. 4, n. 1, p. 71 – 115, 1980. ISSN 0364-0213. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0364021381800055>. Citado na página 31.

JONES, R. M.; LAIRD, J. E.; NIELSEN, P. E.; COULTER, K. J.; KENNY, P.; KOSS, F. V. Automated intelligent pilots for combat flight simulation. *AI magazine*, v. 20, n. 1, p. 27–27, 1999. Citado na página 29.

KEATING, C. B. Emergence in system of systems. *System of Systems Engineering*, Wiley Online Library, p. 169–190, 2008. Citado na página 46.

KELAIDONIS, D.; SOMOV, A.; FOTEINOS, V.; POULIOS, G.; STAVROULAKI, V.; VLACHEAS, P.; DEMESTICHAS, P.; BARANOV, A.; BISWAS, A. R.; GIAFFREDA, R. Virtualization and cognitive management of real world objects in the internet of things. In: *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*. [S.l.: s.n.], 2012. p. 187–194. Citado na página 56.

KHAITAN, S. K.; MCCALLEY, J. D. Design techniques and applications of cyberphysical systems: A survey. *IEEE Systems Journal*, IEEE, v. 9, n. 2, p. 350–365, 2015. Citado 3 vezes nas páginas 49, 50 e 55.

KOMNINOS, N.; SCHAFFERS, H.; PALLOT, M. Developing a policy roadmap for smart cities and the future internet. In: IMC INTERNATIONAL INFORMATION MANAGEMENT CORPORATION. *eChallenges e-2011 Conference Proceedings, IIMC International Information Management Corporation*. [S.l.], 2011. Citado na página 35.

KOTOV, V. *Systems of systems as communicating structures*. [S.l.]: Hewlett Packard Laboratories, 1997. Citado na página 41.

KOTSERUBA, I.; TSOTSOS, J. K. 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review*, Springer, v. 53, n. 1, p. 17–94, 2020. Citado 2 vezes nas páginas 21 e 29.

KOTSIANTIS, S. B. Decision trees: a recent overview. *Artificial Intelligence Review*, Springer, v. 39, p. 261–283, 2013. Citado na página 97.

KWON, O.-y.; SHIN, S.-h.; SHIN, S.-j.; KIM, W.-s. Design of u-health system with the use of smart phone and sensor network. In: IEEE. *Ubiquitous Information Technologies and Applications (CUTE), 2010 Proceedings of the 5th International Conference on*. [S.l.], 2010. p. 1–6. Citado na página 34.

LAIRD, J. E. *The Soar cognitive architecture*. [S.l.]: MIT Press, 2012. Citado na página 27.

LAIRD, J. E.; COULTER, K. J.; JONES, R. M.; KENNY, P. G.; KOSS, F.; NIELSEN, P. E. Integrating intelligent computer generated forces in distributed simulations: Tacair-soar in stow-97. In: *Proceedings of the 1998 Simulation Interoperability Workshop*. [S.l.: s.n.], 1998. Citado na página 29.

LAIRD, J. E.; ROSENBLOOM, P. The evolution of the soar cognitive architecture. *Mind matters: A tribute to Allen Newell*, Mahwah, NJ: Lawrence Erlbaum, p. 1–50, 1996. Citado na página 27.

LANGLEY, P.; LAIRD, J. E.; ROGERS, S. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, Elsevier, v. 10, n. 2, p. 141–160, 2009. Citado na página 28.

LARICCHIA, F. *Number of digital voice assistants in use worldwide from 2019 to 2024 (in billions)*. Statistica, 2022. Disponível em: <https://www.statista.com/statistics/973815/worldwide-digital-voice-assistant-in-use/>. Citado na página 20.

LAURENTIS, D. D.; CALLAWAY, R. K. *et al.* A system-of-systems perspective for public policy decisions. *Review of Policy Research*, Wiley Online Library, v. 21, n. 6, p. 829–837, 2004. Citado na página 46.

LEE, E. A. Cyber-physical systems-are computing foundations adequate. In: CITESEER. *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*. [S.l.], 2006. v. 2. Citado na página 50.

LEE, E. A. Computing needs time. *Communications of the ACM*, ACM, v. 52, n. 5, p. 70–79, 2009. Citado 2 vezes nas páginas 49 e 55.

LEE, E. A.; SESHIA, S. A. *Introduction to embedded systems: A cyber-physical systems approach*. [S.l.]: Lee & Seshia, 2011. Citado na página 48.

LEPRINCE-RINGUET, D. *The global chip shortage is a much bigger problem than everyone realised. and it will go on for longer, too*. 2021. Disponível em: <https://www.zdnet.com/article/the-global-chip-shortage-is-a-bigger-problem-than-everyone-realised-and-it-will-go-on-for-longer-too>. Citado na página 26.

LI, B.; QI, P.; LIU, B.; DI, S.; LIU, J.; PEI, J.; YI, J.; ZHOU, B. Trustworthy ai: From principles to practices. *ACM Computing Surveys*, ACM New York, NY, v. 55, n. 9, p. 1–46, 2023. Citado na página 20.

LIM, K. Y. H.; ZHENG, P.; CHEN, C.-H. A state-of-the-art survey of digital twin: techniques, engineering product lifecycle management and business innovation perspectives. *Journal of Intelligent Manufacturing*, Springer, p. 1–25, 2019. Citado na página 62.

LU, J.; ZHENG, X.; GHARAEI, A.; KALABOUKAS, K.; KIRITSIS, D. Cognitive twins for supporting decision-makings of internet of things systems. In: SPRINGER. *Proceedings of 5th International Conference on the Industry 4.0 Model for Advanced Manufacturing*. [S.l.], 2020. p. 105–115. Citado 4 vezes nas páginas , 67, 70 e 72.

LUKASIK, S. J. Systems, systems of systems, and the education of engineers. *AI EDAM*, Cambridge Univ Press, v. 12, n. 01, p. 55–60, 1998. Citado na página 41.

LUKOWICZ, P.; KIRSTEIN, T.; TROSTER, G. Wearable systems for health care applications. *Methods of Information in Medicine-Methodik der Information in der Medizin*, Stuttgart [etc.] FK Schattauer [etc.], v. 43, n. 3, p. 232–238, 2004. Citado na página 34.

MACDOUGALL, W. *Industrie 4.0: Smart Manufacturing for the Future*. [S.l.]: Germany Trade & Invest, 2014. Citado na página 52.

MAEIR, M. Architecting principles for system-of-systems. *Systems Engineering*, v. 1, n. 4, p. 267–284, 1998. Citado na página 40.

MAIER, M. W. Architecting principles for systems-of-systems. In: WILEY ONLINE LIBRARY. *INCOSE International Symposium*. [S.l.], 1996. v. 6, n. 1, p. 565–573. Citado na página 40.

MASLOW, A. H. *Motivation and personality*. [S.l.]: Prabhat Prakashan, 1981. Citado na página 24.

MELIS, W. J.; CHIZUWA, S.; KAMEYAMA, M. Evaluation of hierarchical temporal memory for a real world application. In: IEEE. *2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*. [S.l.], 2009. p. 144–147. Citado na página 29.

MENDEZ-VAZQUEZ, A.; HELAL, A.; COOK, D. Simulating events to generate synthetic data for pervasive spaces. In: CITESEER. *Workshop on Developing Shared Home Behavior Datasets to Advance HCI and Ubiquitous Computing Research*. [S.l.], 2009. Citado na página 117.

MICLEA, L.; SANISLAV, T. About dependability in cyber-physical systems. In: IEEE. *Design & Test Symposium (EWDTS), 2011 9th East-West*. [S.l.], 2011. p. 17–21. Citado na página 49.

MITCHELL, M.; HOFSTADTER, D. The Copycat Project: A Model of Mental Fluidity and Analogy-making. In: _____. *Fluid Concepts & Analogies: Computer Models of the Fundamental Mechanisms of Thought*. New York: BasicBooks, 1994. Citado na página 21.

MITOLA, J. *An Integrated Agent Architecture for Software Defined Radio*. 2000. Citado na página 55.

MO, Y.; KIM, T. H.-J.; BRANCIK, K.; DICKINSON, D.; LEE, H.; PERRIG, A.; SINOPOLI, B. Cyber–physical security of a smart grid infrastructure. *Proceedings of the IEEE*, IEEE, v. 100, n. 1, p. 195–209, 2012. Citado na página 53.

MORLEY, J. *Five maxims about emergent behavior in systems of systems*. 2006. Citado na página 46.

NAKASHIMA, H.; NODA, I. Dynamic subsumption architecture for programming intelligent agents. In: IEEE. *Proceedings International Conference on Multi Agent Systems (Cat. No. 98EX160)*. [S.l.], 1998. p. 190–197. Citado na página 103.

NEWELL, A.; ROSENBLOOM, P. S.; LAIRD, J. E. *Symbolic architectures for cognition*. [S.l.], 1989. Citado na página 27.

NEWELL, A.; SIMON, H. A. *GPS, a program that simulates human thought*. [S.l.], 1961. Citado na página 27.

NILSSON, D. The usage of unmanned aerial vehicles and their prospects in archaeology. 2013. Citado na página 53.

NSF, N. S. F. Cyber physical systems. 2013. Citado na página 49.

OSMAN, M. An evaluation of dual-process theories of reasoning. *Psychonomic Bulletin & Review*, v. 11, n. 6, p. 988–1010, Dec 2004. Disponível em: <https://doi.org/10.3758/BF03196730>. Citado na página 93.

PARAENSE, A. L.; RAIZER, K.; PAULA, S. M. de; ROHMER, E.; GUDWIN, R. R. The cognitive systems toolkit and the {CST} reference cognitive architecture. *Biologically Inspired Cognitive Architectures*, v. 17, p. 32 – 48, 2016. ISSN 2212-683X. Disponível em: <//www.sciencedirect.com/science/article/pii/S2212683X1630038X>. Citado 5 vezes nas páginas , 79, 80, 81 e 82.

PARAENSE, A. L. O. *A machine consciousness approach to urban traffic signal control= Uma abordagem de consciência de máquina ao controle de semáforos de tráfego urbano*. Tese (Doutorado) — Universidade Estadual de Campinas, 2016. Citado na página 80.

PERERA, C.; LIU, C. H.; JAYAWARDENA, S. The emerging internet of things marketplace from an industrial perspective: A survey. *IEEE Transactions on Emerging Topics in Computing*, IEEE, v. 3, n. 4, p. 585–598, 2015. Citado na página 33.

ROHMER, E.; SINGH, S. P.; FREESE, M. V-rep: A versatile and scalable robot simulation framework. In: IEEE. *2013 IEEE/RSJ international conference on intelligent robots and systems*. [S.l.], 2013. p. 1321–1326. Citado na página 114.

ROSE, K.; ELDRIDGE, S.; CHAPIN, L. The internet of things: An overview. 2015. Citado 4 vezes nas páginas 31, 36, 38 e 39.

ROŽANEC, J.; LU, J.; RUPNIK, J.; ŠKRJANC, M.; MLADENIC, D.; FORTUNA, B.; KIRITSIS, D. Actionable cognitive twins for decision making in manufacturing. arxiv 2021. *arXiv preprint arXiv:2103.12854*, 2021. Citado na página 77.

ROZANEC, J. M.; JINZHI, L.; KOSMERLJ, A.; KENDA, K.; DIMITRIS, K.; JOVANOSKI, V.; RUPNIK, J.; KARLOVCEC, M.; FORTUNA, B. Towards actionable cognitive digital twins for manufacturing. In: *SeDiT@ ESWC*. [S.l.: s.n.], 2020. Citado na página 77.

RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 4. ed. [S.l.]: Pearson Education, 2020. ISBN 0134610997. Citado 4 vezes nas páginas , 29, 30 e 31.

SAMSONOVICH, A. V. Toward a unified catalog of implemented cognitive architectures. *BICA*, v. 221, p. 195–244, 2010. Citado na página 29.

SASIDHARAN, S.; SOMOV, A.; BISWAS, A. R.; GIAFFREDA, R. Cognitive management framework for internet of things: - a prototype implementation. In: *WF-IoT*. [s.n.], 2014. p. 538–543. Disponível em: <http://dblp.uni-trier.de/db/conf/wf-iot/wf-iot2014.html#SasidharanSBG14>. Citado na página 58.

SAUSER, B.; BOARDMAN, J.; GOROD, A. 8 system of systems management. 2008. Citado 2 vezes nas páginas e 43.

SCHAAT, S.; DOBLHAMMER, K.; WENDT, A.; GELBARD, F.; HERRET, L.; BRUCKNER, D. A psychoanalytically-inspired motivational and emotional system for autonomous agents. In: IEEE. *IECON 2013-39th Annual Conference of the IEEE Industrial Electronics Society*. [S.l.], 2013. p. 6648–6653. Citado na página 29.

SCHWEFEL, H.-P. Kybernetische evolution als strategie der experimentellen forschung in der stromungstechnik. *Diploma thesis, Technical Univ. of Berlin*, 1965. Citado na página 105.

SHA, L.; GOPALAKRISHNAN, S.; LIU, X.; WANG, Q. Cyber-physical systems: A new frontier. In: *Machine Learning in Cyber Trust*. [S.l.]: Springer, 2009. p. 3–13. Citado 2 vezes nas páginas 49 e 55.

SHENHAR, A. A new systems engineering taxonomy. In: *Proceedings of the 4th International Symposium of the National Council on System Engineering, National Council on System Engineering*. [S.l.: s.n.], 1994. v. 2, p. 261–276. Citado na página 40.

SIMON, H. A.; NEWELL, A. Human problem solving: The state of the theory in 1970. *American Psychologist*, American Psychological Association, v. 26, n. 2, p. 145, 1971. Citado na página 27.

SLOANE, E. Understanding the emerging national healthcare it infrastructure. *24× 7 Magazine*, 2006. Citado na página 45.

SLOANE, E.; WAY, T.; GEHLOT, V.; LEVITIN, A.; BECK, R. Sose modeling and simulation approaches to evaluate security and performance limitations of a next generation national healthcare information network (nhin-2). In: IEEE. *2007 IEEE International Conference on System of Systems Engineering*. [S.l.], 2007. p. 1–6. Citado na página 45.

SLOMAN, A. Architecture-based conceptions of mind. In: *In the Scope of Logic, Methodology and Philosophy of Science*. [S.l.]: Springer, 2002. p. 403–427. Citado na página 28.

SOMERS, S.; OLTRAMARI, A.; LEBIERE, C. Cognitive twin: A cognitive approach to personalized assistants. In: *AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering (1)*. [S.l.: s.n.], 2020. Citado 4 vezes nas páginas 70, 72, 76 e 77.

SOMERS, S.; OLTRAMARI, A.; LEBIERE, C. Cognitive twin: A personal assistant embedded in a cognitive architecture. 2021. Citado na página 76.

STANKOVIC, J. A.; LEE, I.; MOK, A.; RAJKUMAR, R. Opportunities and obligations for physical computing systems. *Computer*, IEEE, v. 38, n. 11, p. 23–31, 2005. Citado na página 52.

STRUBELL, E.; GANESH, A.; MCCALLUM, A. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019. Citado na página 25.

SUN, R. Desiderata for cognitive architectures. *Philosophical Psychology*, Taylor & Francis, v. 17, n. 3, p. 341–373, 2004. Citado na página 28.

SUN, R. The importance of cognitive architectures: An analysis based on clarion. *Journal of Experimental & Theoretical Artificial Intelligence*, Taylor & Francis, v. 19, n. 2, p. 159–193, 2007. Citado na página 28.

SUN, R. *The CLARION Cognitive Architecture*. Oxford University Press, 2015. Disponível em: <https://doi.org/10.1093/oxfordhb/9780199842193.013.11>. Citado na página 79.

SUN, R.; ZHANG, X. Top-down versus bottom-up learning in skill acquisition. In: ROUTLEDGE. *Proceedings of the Twenty-Fourth Annual Conference of the Cognitive Science Society*. [S.l.], 2019. p. 861–866. Citado na página 29.

SWENEY, M. *Global shortage in computer chips 'reaches crisis point'*. the Guardian, 2021. Disponível em: <https://www.theguardian.com/business/2021/mar/21/global-shortage-in-computer-chips-reaches-crisis-point>. Citado na página 26.

TAY, L.; DIENER, E. Needs and subjective well-being around the world. *Journal of personality and social psychology*, American Psychological Association, v. 101, n. 2, p. 354, 2011. Citado na página 24.

THOMAS, R.; DASILVA, L.; MACKENZIE, A. Cognitive networks. In: *New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005. 2005 First IEEE International Symposium on*. [S.l.: s.n.], 2005. p. 352–360. Citado na página 55.

THORMUNDSSON, B. *Artificial intelligence (AI) market size worldwide in 2021 with a forecast until 2030*. Statistica, 2023. Disponível em: <https://www.statista.com/statistics/1365145/artificial-intelligence-market-size/>. Citado na página 20.

THORMUNDSSON, B. *Artificial intelligence (AI) worldwide - statistics & facts*. Statistica, 2024. Disponível em: <https://www.statista.com/topics/3104/artificial-intelligence-ai-worldwide/#topicOverview>. Citado na página 20.

TURING, A. M. Lecture to the london mathematical society on 20 february 1947. *MD COMPUTING*, SPRINGER VERLAG KG, v. 12, p. 390–390, 1995. Citado na página 23.

UDAY, P.; MARAIS, K. B. Resilience-based system importance measures for system-of-systems. *Procedia Computer Science*, Elsevier, v. 28, p. 257–264, 2014. Citado 3 vezes nas páginas , 47 e 48.

VETERE, G.; LENZERINI, M. Models for semantic interoperability in service-oriented architectures. *IBM Systems Journal*, IBM, v. 44, n. 4, p. 887–903, 2005. Citado na página 54.

VLACHEAS, P.; GIAFFREDA, R.; STAVROULAKI, V.; KELAIDONIS, D.; FOTEINOS, V.; POULIOS, G.; DEMESTICHAS, P.; SOMOV, A.; BISWAS, A. R.; MOESSNER, K. Enabling smart cities through a cognitive management framework for the internet of things. *IEEE Communications Magazine*, v. 51, n. 6, p. 102–111, June 2013. ISSN 0163-6804. Citado 2 vezes nas páginas e 59.

WALDO, J. Virtual organizations, pervasive computing, and an infrastructure for networking at the edge. *Information Systems Frontiers*, Springer, v. 4, n. 1, p. 9–18, 2002. Citado na página 31.

WANG, L.; TÖRNGREN, M.; ONORI, M. Current status and advancement of cyber-physical systems in manufacturing. *Journal of Manufacturing Systems*, Elsevier, v. 37, n. Part 2, p. 517–527, 2015. Citado 3 vezes nas páginas , 50 e 51.

WILBER, G. F. Boeing's sose approach to e-enabling commercial airlines. *System of Systems Engineering*, Wiley Online Library, p. 232–256, 2008. Citado 2 vezes nas páginas 46 e 47.

WONG, A. K. Y.; RAY, P.; PARAMESWARAN, N.; STRASSNER, J. Ontology mapping for the interoperability problem in network management. *IEEE Journal on selected areas in Communications*, IEEE, v. 23, n. 10, p. 2058–2068, 2005. Citado na página 54.

WU, Q.; DING, G.; XU, Y.; FENG, S.; DU, Z.; WANG, J.; LONG, K. Cognitive internet of things: a new paradigm beyond connection. *IEEE Internet of Things Journal*, IEEE, v. 1, n. 2, p. 129–143, 2014. Citado 2 vezes nas páginas 22 e 36.

XU, L. D.; HE, W.; LI, S. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, IEEE, v. 10, n. 4, p. 2233–2243, 2014. Citado 2 vezes nas páginas 35 e 36.

ZANELLA, A.; BUI, N.; CASTELLANI, A.; VANGELISTA, L.; ZORZI, M. Internet of things for smart cities. *IEEE Internet of Things Journal*, IEEE, v. 1, n. 1, p. 22–32, 2014. Citado na página 34.

ZHANG, N.; BAHSOON, R.; THEODOROPOULOS, G. Towards engineering cognitive digital twins with self-awareness. In: IEEE. *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. [S.l.], 2020. p. 3891–3891. Citado 7 vezes nas páginas , 62, 63, 72, 73, 74 e 75.

Apêndices

APÊNDICE A – Detalhes de Implementação do DCT

A.1 Criando e Executando um Aplicativo DCT

Um conceito importante para nosso trabalho é o conceito de um Aplicativo DCT. Um Aplicativo DCT, implementando uma arquitetura cognitiva distribuída, corresponde a um conjunto de *Nodes* DCT, executando de maneira integrada, ou seja, quando os Codelets de um *Node* se comunicam com Memórias de outros *Nodes*. Dessa forma, para se criar um Aplicativo DCT, é necessário definir um conjunto de dispositivos físicos ou virtuais (e.g. Containers Docker), onde cada *Node* será executado, e listar os endereços IP e portas em que esses recursos deverão ficar disponíveis.

Para implementar aplicativos DCT, o código do DCT foi disponibilizado em repositório público [1]. No repositório, encontram-se os códigos-fonte implementados em Python e Shell script. O principal diretório, nomeado **dct**, possui estrutura própria de módulos Python (com arquivo "setup", por exemplo) e pode ser diretamente instalado utilizando o comando *pip*, seguindo o método padrão de instalação a partir de fontes locais. Para isto é necessário:

1. Através de um terminal, navegar até o diretório raiz do pacote;
2. Executar o comando "`pip install .`" sem ou com privilégios de administrador, dependendo do tipo de instalação escolhida.

Será então possível importar em todo sistema o módulo recém instalado como `dct`. Em sistemas que suportam *pip* e este tipo de instalação de pacotes, é conveniente utilizar desta forma pois suprime-se a necessidade de fornecer o diretório **dct** junto aos *scripts* que irão utilizá-lo. O diretório **pythonCodelet** possui um exemplo de estrutura de um Codelet DCT: um script `codelet.py` onde existe uma implementação "personalizada" que estende uma classe pré-existente, um arquivo `fields.json`, com as informações necessárias ao Codelet e uma pasta `memories` com apenas um arquivo (`simple-memory.json`) representando uma Memória. O diretório **nodes/shell_python_node** possui dois arquivos, sendo o `nodeMaster.sh` — responsável pela observação e controle dos processos internos do *Node* — e um arquivo `param.ini`, com informações necessárias ao funcionamento do *Node*. O diretório **utils** contém alguns scripts que podem ser bastante úteis no

[1] https://github.com/wandgibaut/dct

processo de desenvolvimento, como o `add_simple_container.sh`, que recebe como parâmetros um nome, um caminho de diretório com um *Node* e um par ip/porta, iniciando um *Node* com essas informações. O arquivo `requirements.txt` está presente para o caso de uso sem a necessidade de instalação do DCT no sistema.

Para criar e executar um aplicativo, o usuário deve seguir os seguintes passos:

- **Definição dos tipos de Memória a serem utilizados**: para cada Memória a ser criada em um *Node*, é necessário especificar como os dados dessa memória serão armazenados: se utilizando um banco de dados, como o *MongoDB* ou o *Redis*, ou se esses dados serão armazenados em um arquivo *.json* local. Para o acesso a essa memória, por meio da rede, a mesma tanto pode ser acessada utilizando-se dos mecanismos diretos do banco de dados, por meio de uma URL de acesso, ou por meio da disponibilização do dado do arquivo utilizando-se sockets, via TCP, definindo-se o *IP/port* para o acesso. Assim, para o acesso à memória, temos as seguintes opções disponíveis: *MongoDB*, *Redis* ou *TCP*. No caso de Memórias que estejam em banco de dados *MongoDB*, o nome da memória deve apresentar um certo padrão: Memórias em bancos *MongoDB* devem ter como nome `<coleção>:<nome-da-entrada>` e a base deve possuir como nome `database-raw-memory`. Outros tipos de Memórias não apresentam essa restrição de nome, mas ainda é necessário apontar a url ou caminho correto. É válido ressaltar que, com exceção de bancos *Redis* internos ao *Node*, a responsabilidade de pôr os bancos de dados em operação (ou seja, a instalação do banco de dados e sua inicialização, de modo automático, pelo sistema) não compete ao DCT. Caso não exista tal entrada no banco de dados, o DCT criará uma entrada nova. O mesmo mecanismo ainda não está disponível para Memórias *JSON*, cabendo ao programador garantir que as mesmas existam;

- **Implementação dos Codelets**: é necessário implementar o código para cada um dos Codelets a serem utilizados em cada Node, incluindo-se os dados das Memórias a serem acessadas por ele. Cada memória acessada pelo Codelet, tanto na entrada como na saída, deverá ser especificada por meio de sua URL ou seu *IP/port* de acesso, no arquivo *fields.json*;

- **Organização hierárquica da estrutura do *Node***: dentro do *Node* todos os Codelets devem estar em uma pasta (chamada `codelets`) em diretórios próprios, nomeados de forma a coincidir com os nomes encontrados no arquivo `param.ini`. Cada diretório deve conter um script `codelet.py`. Na raiz do *Node* deve constar ainda o `server.py`. Assim, nenhuma alteração precisará ser feita no `nodeMaster.py`. É necessária a definição de uma variável de sistema, o *ROOT_NODE_DIR*, que deve apontar para o diretório raiz do *Node*;

- **Execução individual de cada Node**: é necessário executar individualmente cada um dos *Nodes*, utilizando-se o código do *Node Master*, que se encarregará de chamar o executável principal de cada Codelet interno e iniciar um monitoramento da integridade dos mesmos. Isso pode ser feito manualmente, ou utilizando alguma ferramenta automatizada como, por exemplo, o *docker-compose*, caso os *Nodes* devam ser executados em *containers* Docker.

É possível ainda enviar os arquivos relacionados aos Nodes para outras máquinas físicas e executá-los remotamente via ssh, caso a máquina em questão possua um *ssh daemon* em execução. Estes processos podem ser facilitados ao utilizar-se os scripts `send_over_ssh.sh` e `start_over_ssh.sh`, respectivamente.

A.2 Rotinas e Subrotinas Disponíveis no Node Master

Em termos práticos, o *Node Master* possui as seguintes rotinas e subrotinas:

- **check_integrity**: função que verifica periodicamente se cada um dos processos (PID) que deveriam estar executando estão de fato em execução
- **regenerate_process**: caso um processo não esteja executando (seja ele relativo a Codelets ou ao servidor) e deveria estar, essa função executa novamente a entidade relevante, gerando um novo PID que é registrado
- **eliminate_codelet**: encerra a execução de um Codelet específico, evitando que o *Node Master* execute-o novamente
- **run_new_codelet**: inicia a execução de um Codelet específico, adicionando o PID à lista de processos que deveriam executar
- **check_codelets**: verifica se um Codelet que deveria estar em execução não está executando, ou se algum Codelet que está em execução não deveria estar
- **containsElement**: verifica se um *array* contém um determinado elemento
- **init**: apesar de não definido formalmente como função, uma parte do *script* do *Node Master* é dedicada à execução e ao arranjo inicial dos recursos do Nó em questão
- **main**: também não definida formalmente como função, uma parte do *script* do *Node Master* é basicamente um *loop* onde se executam etapas de verificação da integridade dos componentes do Nó

A.3 Estrutura Interna de Arquivos

A.3.1 fields.json (Arquivo de Configuração)

A lista completa de parâmetros contidos no arquivo de configuração é:

- *name*: O nome do *Codelet*.
- *threshold*: o limiar acima do qual o *Codelet* é considerado ativo.
- *timestep*: o intervalo de tempo, em segundos, onde o *Codelet* deve ser executado.
- *inputs*: lista com objetos contendo informações de acesso às Memórias de entrada, tais como ip/porta e tipo.
- *outputs*: equivalente aos *inputs*, porém em relação às Memórias de saída.
- *broadcast*: equivalente aos *inputs*, porém em relação às Memórias que não fazem parte da estrutura topológica direta do sistema, mas que devem ser ouvidas mesmo assim.

A.3.2 param.ini (relativo ao Node)

O arquivo `param.ini` relativo — e necessário — ao *Node Master* considera a formatação padrão para arquivos de extensão `.ini` e deve conter as seguintes seções:

- **internal_codelets**: essa seção deve listar todos os *Codelets* que o *Node* possui internamente, estejam eles em execução ou não.
- **active_codelets**: essa seção deve conter todos os *Codelets* que devem executar ou estar em execução, no caso do *Node* já estar em funcionamento.
- **signals**: seção dedicada a sinais externos de controle. Deve constar o *suicide_note* e *death_threshold*. O primeiro (geralmente especificado como *false*) determina se um *Node* deve parar sua execução imediatamente, enquanto o segundo representa um limiar de votos vindos de outros *Nodes* após o qual o *Node* em questão deve encerrar sua execução. Ambos são mecanismos criados para evitar que o sistema torne-se impossível de ser desligado sem uso de medidas "extremas", como cortar a fonte de energia do dispositivo físico.
- **memory**: nesta seção a única indicação é se *redis* é setado como verdadeiro ou falso. Isso determinará se o *Node Master* colocará um banco de dados *Redis* em operação ou não.

A.3.3 Memórias

Uma Memória DCT pode ser implementada de diversas formas, como já mencionado. No entanto, para garantir o pleno funcionamento das partes, deve conter as seguintes informações:

- **name** (String): nome da Memória.
- **IP/port** (String): endereço onde a Memória pode ser alcançada. Esse campo é um caminho absoluto em caso de Memórias do tipo *local* e uma URL em outros casos.
- **type** (String): tipo da Memória, podendo ser: *local*, *tcp*, *redis* e *mongo*
- **I** (String): informação principal guardada na Memória. Por convenção, essa informação é do tipo *String*, mas ela pode ser uma serialização (formato JSON ou *pickle*) de qualquer outro tipo de informação
- **eval** (Double): informação auxiliar adicional que pode ser utilizada em caso de necessidade de tomada de decisão ou desambiguação. O *eval* pode, por exemplo, ser utilizado por um Codelet para definir qual Memória deve ser processada primeiro.
- **group** (list of Strings): lista com todos os grupos aos quais a Memória faz parte. Esse campo pode ser utilizado para escrever ou consumir informação em múltiplas Memórias diretamente relacionadas.

Além disso, tais informações devem estar em formato serializável via JSON (como um dicionário em python, por exemplo), dado que este é o método de escolha de comunicação entre *Nodes* DCT.

A.4 Funções e Métodos Suportados no Pacote DCT

A.4.1 Funções Implementadas no Pacote DCT em Python

Conforme mencionado, no presente trabalho, a única implementação do DCT é em python. Dentre as funcionalidades já implementadas estão:

- **get_memory_object_by_name**: retorna uma lista de Memórias a partir do seu nome. A lista contém todas as Memórias que possuem o nome passado como parâmetro.
- **get_memory_object_by_group**: retorna uma lista de Memórias a partir do seu grupo. A lista contém todas as Memórias que possuem o grupo passado como parâmetro.

- **get_memory_object**: método que, dado nome, ip/port e tipo de comunicação, retorna uma Memória. Esse método não exige a leitura do arquivo *fields.json*, portanto é mais rápido que os métodos anteriores. No entanto, exige-se maior atenção em sua utilização para evitar erros.

- **set_memory_object_by_name**: método que, dado um nome, um campo e um valor, altera o valor do campo passado como parâmetro em uma dada Memória.

- **set_memory_object_by_group**: método que, dado um grupo, um campo e um valor, altera o valor do campo passado como parâmetro em todas as Memórias que pertencem ao grupo passado como parâmetro.

- **set_memory_object**: método que, dado um nome, um ip/port, um tipo de comunicação, um campo e um valor, altera o valor do campo passado como parâmetro em uma dada Memória. Esse método não exige a leitura do arquivo *fields.json*, portanto é mais rápido que os métodos anteriores. No entanto, exige-se maior atenção em sua utilização para evitar erros.

- **add_memory_to_group**: método que, dado um nome e grupo, adiciona uma dada Memória ao grupo passado como parâmetro.

- **get_node_info**: retorna as informações de um Nó, dado um ip e uma porta.

- **get_codelet_info**: retorna as informações de um Codelet, dado um nome, um ip e uma porta.

A.4.2 API REST do DCT

Todo *Node* DCT deve implementar um servidor que atenda requisições REST para comunicar-se com os demais elementos dos sistemas e subsistemas aos quais está inserido. Até o momento, as seguintes requisições podem ser realizadas:

- **'<ip/port>/get_memory/<memory_name>'**: retorna uma lista de Memórias serializadas em formato *JSON* a partir do seu nome. A lista contém todas as Memórias que possuem o nome passado como parâmetro. Caso não haja Memórias com este nome, o servidor retornará uma mensagem com código 404.

- **'<ip/port>/set_memory/', methods=['POST']**: A partir de parâmetros passados no corpo da requisição, altera o valor de um campo de uma dada Memória. Caso não haja Memórias com este nome, o servidor retornará uma mensagem com código 404.

- **'<ip/port>/get_codelet_info/<codelet_name>'**: retorna as informações de um Codelet, dado um nome. Caso não haja Codelets com este nome, o servidor retornará uma mensagem com código 404.
- **'<ip/port>/get_node_info'**: retorna as informações do Nó, como número de Codelets, quais Mémorias são lidas e escritas por elementos internos deste Nó.
- **'<ip/port>/kill_codelet/<codelet_name>'**: para a execução de um Codelet, dado um nome.
- **'<ip/port>/run_codelet/<codelet_name>'**: inicia a execução de um Codelet, dado um nome.
- **'<ip/port>/configure_death/'**: reinicia a contagem de "votos" do Nó para a sua morte. Também atualiza o valor de limiar.
- **'<ip/port>/vote_kill/', methods=['POST']**: vota na morte de um Nó, dado um ip e uma porta. Recebe uma confirmação de voto.
- **'<ip/port>/die/', methods=['POST']**: adiciona um voto para a morte do Nó. Caso o número de votos seja maior que o limiar, o *Node Master* para sua execução.
- **'<ip/port>/die_now/'**: para a execução do Nó.

APÊNDICE B – Artigos gerados no contexto deste doutorado

- Building a Cognitive Twin Using a Distributed Cognitive System and an Evolution Strategy (submetido no Cognitive Systems Research)
- Extending the CST: The Distributed Cognitive Toolkit (GIBAUT; GUDWIN, 2020)
- Using Environment Exploration for Inverse Kinematics Learning (GIBAUT; GUDWIN, 2019)

APÊNDICE C – Outros artigos gerados no período deste doutorado

- Neurosymbolic AI and its Taxonomy: a Survey (submetido no ACM)
- Toward a Federated Model for Human Context Recognition on Edge Devices (GIBAUT *et al.*, 2022)
- A Double-Layer Subsumption Mechanism for Enforcing Sequential Behaviors in a Cognitive Architecture (GUDWIN *et al.*, 2021)
- The TROCA Project: An autonomous transportation robot controlled by a cognitive architecture (GUDWIN *et al.*, 2020)
- A Cognitive Architecture for a Transportation Robotic System (GUDWIN *et al.*, 2019)

APÊNDICE D – Patentes geradas no período deste doutorado

- Sequential Behavior for Intelligent Control in Subsumption-like Architecture (*United States application or PCT international application number PCT/EP2022/072962 - WO2023021091A1*, depositado em 17 de Agosto de 2022, concedido em 23 de Fevereiro de 2023)