UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Ramna Sidharta de Andrade Palma

Interoperabilidade em aplicações distribuídas:
Um estudo de DLTs como infraestrutura para hospedagem de aplicações web

Florianópolis
2021

Ramna Sidharta de Andrade Palma

Interoperabilidade em aplicações distribuídas:

Um estudo de DLTs como infraestrutura para hospedagem de aplicações web

Trabalho de Conclusão de Curso de Graduação em Ciência da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Ciência da Computação.
Orientador: Prof. Martín Augusto Gagliotti Vigil, Dr.
Coorientador: Prof. Jean Everson Martina, Dr.

Florianópolis

2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Palma, Ramna Sidharta
Interoperabilidade em aplicações distribuídas : Um estudo de DLTs como infraestrutura para hospedagem de aplicações web / Ramna Sidharta Palma ; orientador, Martín Augusto Gagliotti Vigil, coorientador, Jean Everson Martina, 2021.
p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2021.

Inclui referências.

1. Ciências da Computação. 2. Web3. 3. Tecnologia de livro-razão distribuído. 4. Interoperabilidade. 5. DApps. I. Augusto Gagliotti Vigil, Martín. II. Everson Martina, Jean. III. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. IV. Título.

Ramna Sidharta de Andrade Palma

Interoperabilidade em aplicações distribuídas:

Um estudo de DLTs como infraestrutura para hospedagem de aplicações web

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Ciências da Computação" e aprovado em sua forma final pelo Programa de Graduação em Ciências da Computação.

Florianópolis, 13 de setembro de 2021.

_____

Prof. Jean Everson Martina, Dr.
Coordenador do Curso

Banca Examinadora:

_____

Prof. Martín Augusto Gagliotti Vigil, Dr.
Orientador
Universidade Federal de Santa Catarina (Campus Araranguá)

_____

Prof. Jean Everson Martina, Dr.
Coorientador
Universidade Federal de Santa Catarina

_____

Lucas Palma, Me.
Avaliador
Universidade Federal de Santa Catarina

_____

Gustavo Zambonin, Me.
Avaliador
Universidade Federal de Santa Catarina

À minha avó, Elsita Thorstenberg de Andrade, e a todas as pessoas que empregam esforços para uma sociedade mais justa e esclarecida.

AGRADECIMENTOS

Agradeço à minha família, por nutrir-me intelectualmente e espiritualmente, e por sempre apoiar-me. A todos os colegas e amigos, que fizeram desta jornada mais alegre. À Miriã, pela lealdade, amor e inspiração. E ao Martín, professor que, não surpreendentemente, orientou este trabalho com excelência.

The ultimate goal of the Web is to support and improve our weblike existence in the world. We clump into families, associations, and companies. We develop trust across miles and distrust around the corner. What we believe, endorse, agree with, and depend on is representable and, increasingly, represented on the Web. We all have to ensure that the society we build with the Web is the sort we intend.

(BERNERS-LEE, 1999)

RESUMO

Em meio a uma centralização de serviços essenciais da web, graves vazamentos de dados e questões alarmantes com redes sociais, cresce a importância de alternativas ao modelo atual da web 2.0, e a plataformas de computação em nuvem, onde são hospedadas aplicações web modernas. A tecnologia de livro-razão distribuído (DLT) vem contribuindo para o desenvolvimento da web 3.0, permitindo soluções descentralizadas. Apesar disso, o modelo de blockchain, tipo de DLT mais difundido, apresenta sérias limitações, em relação à, por exemplo, privacidade, interoperabilidade e escalabilidade. Diante disto, este trabalho visa responder se plataformas DLT são uma alternativa à plataformas de computação em nuvem, servindo como infraestrutura para hospedagem de aplicações web modernas. Para tanto, foi determinado o foco em interoperabilidade, dado que este é um desafio tanto em blockchains quanto na web 2.0, onde os dados são ilhados por barreiras técnicas e organizacionais entre aplicações. Sendo assim, são escolhidas quatro plataformas DLT de propósito geral e determinados quatro aspectos de interoperabilidade, sob os quais as plataformas são analisadas, comparadas e ranqueadas. Conclui-se que as DLTs proveem uma interoperabilidade ainda maior do que na web atual, com algumas vantagens sobre plataformas de computação nuvem atuais, mas que as formas de DLTs interoperarem com a web ainda são laboriosas. Além disso, constata-se que o modelo da Holochain é mais flexível do que o de blockchain, o que facilita interoperabilidade e escalabilidade.

Palavras-chave: Web3. Tecnologia de Livro-razão Distribuído. Holochain. Polkadot. Ethereum. DFinity.

ABSTRACT

Amid a centralization of essential web services, serious data leaks, and alarming issues with social networks, the importance of alternatives to the current "web 2.0" model and to cloud computing platforms, which hosts modern web applications, is growing. Distributed Ledger Technology (DLT) has contributed to the evolution of the “web 3.0” model, allowing for decentralized solutions. Despite this, its most disseminated model (blockchain) has serious limitations, such as privacy, interoperability, and scalability. We aim to answer whether DLT platforms are an alternative to cloud computing platforms, serving as an infrastructure for hosting modern web applications. For this, we focus on interoperability, which is a challenge both in blockchains and in the "web 2.0", where databases are isolated by technical and organizational barriers between applications. Thus, we selected four general-purpose DLT platforms and determined four interoperability aspects, which were then used to analyze, compare, and rank the DLTs. We conclude that DLTs provide even greater interoperability than the current web paradigm, with some advantages over current cloud computing platforms, but the ways for DLTs to interoperate with the web are still laborious. In addition, the Holochain model is found to be more flexible than the blockchain, which facilitates interoperability and scalability.

Keywords: Web3. DLT. Holochain. Polkadot. Ethereum. DFinity.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface (interface de programação de aplicação)
dApp	Distributed application (aplicação distribuída)
DAO	Decentralized Autonomous Organization (organização autônoma descentralizada)
DHT	Distributed Hash Table (tabela de espalhamento distribuída)
DLT	Distributed Ledger Technology (tecnologia de livro-razão distribuído)
GRANDPA	GHOST-based Recursive Ancestor Deriving Prefix Agreement
hApp	Aplicação Holochain
P2P	Peer-to-peer (ponto à ponto)
RPC	Remote Procedure Call (chamada procedimento remoto)
WASM	Web-Assembly
XCMP	Cross-chain Message Passing

SUMÁRIO

1 INTRODUÇÃO

Historicamente, novos modelos de computação tedem a emergir a cada 10 ou 15 anos: em 1960 existiam os *mainframes*; na década seguinte surgiram os computadores pessoais; a internet chegou ao grande público apenas na década de 1990; e nos anos 2000 surgiram as primeiras plataformas de computação em nuvem (e.g. Amazon Web Service, Google App Engine), juntamente com os *smartphones*. Cada novo modelo tornou possível uma gama de aplicações. A exemplo, dispositivos celulares com acesso à internet, câmeras e sensores de GPS possibilitaram modelos de negócio e aplicações como Uber, Telegram, Spotify, Waze e Reddit. Plataformas de computação em nuvem facilitaram o desenvolvimento de aplicações e pequenos negócios, por terem virtualizado servidores, permitindo desenvolvedores e empreendedores ultrapassarem limitações físicas, reduzindo ou eliminando os obstáculos de provisão e gerenciamento de infraestrutura.

Com infraestrutura, plataformas e aplicações sendo fornecidas como serviço, as oportunidades são mais abundantes, e o desenvolvimento tecnológico e de empresas é, de maneira em geral, mais acelerado (BERMAN et al., 2012). A hospedagem em nuvem é hoje inclusive considerada uma característica de aplicações web (chamada de *cloud-hosted*). Neste trabalho, "aplicação web" refere-se a qualquer software aplicativo que depende da web para sua execução correta (GELLERSEN; GAEDKE, 1999). Outra definição útil é data pela Microsoft: é um sistema composto por um conjunto de funções que executam em uma plataforma de computação e que são acessíveis por usuários através de um navegador web com conexão à internet (BANSOD; HSUEH; WONG, 2005).

Apesar do avanço proporcionado pelas plataformas de computação em nuvem, grande parte do tráfego da internet hoje é centralizado em algumas corporações gigantescas. A fase inicial da web, referida por web 1.0 e caracterizada por websites estáticos e consumo de conteúdo por parte dos usuários, proporcionou uma democratização da informação. A web atual, referida por web 2.0, é caracterizada por ser social, com aplicações web interativas, onde os usuários colaboram, consomem, produzem e compartilham conteúdo. Estas aplicações são hospedadas maioritariamente em servidores centralizados, bem como os dados de seus usuários. Ou seja, dados pessoais e a maior parte da informação da humanidade estão centralizados, sob custódia de algumas empresas. Diante da influência de algoritmos de redes sociais em eleições de países, diversos escândalos de vazamento de dados e ataques cibernéticos, cresce a necessidade de regulamentações e a preocupação com privacidade e segurança de dados

(AUXIER et al., 2019) (NOORDYKE, 2019) (HERN, 2018) (WHITNEY, 2021). São exemplos de desafios da web atual (lê-se também “de aplicações web”):

- segurança - servidores centralizados são alvos de diversos tipos de ataques, além da necessidade de guardar ambientes físicos dos *data centers*;
- resiliência - se um componente importante de uma aplicação falha, o serviço pode apresentar instabilidade ou mesmo ficar indisponível, potencialmente afetando dezenas de milhões de usuários;
- interoperabilidade - as bases de dados são ilhadas por barreiras técnicas e organizacionais entre aplicações, e para elas operarem entre si é necessário integrações por API (*Application Programming Interface*), o que requer, muitas vezes, esforço hercúleo.

Diante deste contexto, há um intenso desenvolvimento da tecnologia de livro-razão distribuído (Distributed Ledger Technology ou DLT), em meio a um movimento tecnológico para a descentralização, que cresceu rapidamente na última década e faz parte da chamada web 3.0. Ele teve seu início em 2008, com o artigo "Bitcoin: A Peer-to-Peer Electronic Cash System" (NAKAMOTO, 2008).

A tecnologia blockchain, um tipo de DLT cujo livro razão é composto por blocos criptograficamente encadeados, cada um contendo um conjunto disjunto de transações, conceitualizada no artigo, permitiu a criação de uma aplicação descentralizada de pagamentos em dinheiro eletrônico (o Bitcoin), a qual funciona sem intervenção, dependência e controle de governos e grandes corporações, ou qualquer intermediário.

Com base no mesmo modelo de computação, foi criada a Ethereum (BUTERIN, 2014), que marcou uma evolução do modelo de blockchain, o unindo ao conceito de *smart-contracts* (scripts executados sob consenso por um conjunto de participantes de uma DLT), assim servindo como uma plataforma para a hospedagem de aplicações distribuídas. Blockchains passaram a ser utilizadas para uma variedade de finalidades, desde jogos (ORDANO et al., [s.d.]), até internet das coisas, inclusive aplicações web em geral. Apesar de tudo isso, há controvérsias quanto a sua adequabilidade para esses casos de uso: a tecnologia possui desafios de escalabilidade, privacidade, interoperabilidade (com aplicações da web atual ou entre blockchains), entre outros (WUST; GERVAIS, 2018) (JOSHI; HAN; WANG, 2018).

À vista dos desafios da web atual e de aplicações hospedadas em plataformas de computação em nuvem, e do desenvolvimento de DLTs, este trabalho visa responder a seguinte questão de pesquisa: **DLTs podem ser uma alternativa a plataformas de computação em nuvem, servindo para hospedagem de aplicações web modernas?** Dito isso, este trabalho é

um estudo do uso de DLTs como infraestrutura para hospedagem de aplicações web distribuídas. Como um começo, para responder esta pergunta, foi determinado o foco em interoperabilidade, dados os problemas de interoperabilidade da web 2.0 e de blockchains. Sendo assim, este trabalho seleciona DLTs de propósito geral, determina quatro aspectos de interoperabilidade e faz uma análise comparativa detalhadas. Por fim, as DLTs selecionadas são ranqueadas quanto aos pontos de vista de interoperabilidade exclusiva à DLTs e interoperabilidade com a web 2.0, e é apresentada uma discussão sobre o tema.

1.1 OBJETIVOS

1.1.1 Objetivo geral

Apresentar um estudo sobre a interoperabilidade em DLTs de propósito geral, situando o uso dessa tecnologia como alternativa para hospedagem da web.

1.1.2 Objetivos Específicos

i. Selecionar DLTs populares de propósito geral;
ii. Definir critérios de interoperabilidade para comparação das DLTs;
iii. Descrever seu funcionamento, analisá-las, compará-las e ranqueá-las quanto ao seu grau de interoperabilidade.

1.2 TRABALHOS RELACIONADOS

Num primeiro momento, buscou-se por artigos que relacionassem DLTs e aplicações web. Com relação ao objetivo deste trabalho, foi encontrado apenas um: (ALABDULWAHHAB, 2018) faz um estudo geral sobre os desafios da web atual, referida por web 2.0, e os contrasta com o uso de blockchain para o desenvolvimento de aplicações web 3.0. O artigo é abrangente, mas pode ser compreendido como uma contextualização deste trabalho.

Há alguns trabalhos comparativos de DLTs. “A Comparative Analysis of Distributed Ledger Technology Platforms” faz uma comparação com critérios tanto qualitativos como quantitativos, mas sem foco em interoperabilidade e sem foco no desenvolvimento de aplicações web (por exemplo, são incluídas algumas DLTs que não suportam *smart-contracts*) (CHOWDHURY et al., 2019). “A Comparative Analysis of Distributed Ledger Technologies for Smart Contract Development” faz também um trabalho comparativo, considera *smart-contracts*, mas não aborda interoperabilidade (BENAHMED et al., 2019). “A Derivation of Categories for Interoperability of Blockchain and Distributed Ledger Systems” visa interoperabilidade, mas apenas entre plataformas blockchain, e de forma ampla, considerando diferentes critérios como linguagem de *scripting*, privacidade, modelo de consenso, sistema de recompensa ou incentivo (ZEUCH et al., 2019). “From Blockchain to Hashgraph: Distributed Ledger Technologies in the Wild” explora DLTs de diferentes formatos (não apenas blockchain) em uma análise comparativa, mas não foca em interoperabilidade e no desenvolvimento de aplicações web, e por abordar vários critérios, o resultado da comparação é mais geral (como “sim” ou “não” para escalabilidade) (AKHTAR, 2019).

É relevante notar que há uma categoria mais específica de trabalhos, que explora interoperabilidade em detalhes (BELCHIOR et al., 2021) (KANNENGIESSER et al., 2020) (KOŠT’ÁL, 2020). Todavia, estes são focados somente em blockchain, em mais baixo nível e envolvem detalhes avançados de protocolos. Este trabalho não é sobre blockchain, mas sim aplicações distribuídas.

O presente trabalho se diferencia dos acima apresentados ao comparar não apenas blockchains, mas também plataformas DLTs de propósito geral, quanto a servir como infraestrutura para o desenvolvimento de aplicações web interoperáveis. Ainda, com o foco em aplicações, são traçados paralelos e comparações com o desenvolvimento na web atual.

1.3 ORGANIZAÇÃO DO TRABALHO

No Capítulo 2 é explicado como as DLTs e critérios de análise comparativa foram selecionados, bem como define estes. No Capítulo 3 são explicadas cada uma das DLTs selecionadas. No Capítulo 4 as DLTs são analisadas comparativamente quanto aos critérios. O capítulo se encerra com a Seção 4.5, dedicada ao ranqueamento das DLTs, representado em Quadro 1 e Quadro 3. O Capítulo 5 discute o desenvolvimento da tecnologia de DLTs. O Capítulo 6 apresenta as conclusões deste trabalho e sugere trabalhos futuros.

2 METODOLOGIA

Tendo em vista que o objetivo é uma análise detalhada, foi determinado um limite de quatro plataformas DLT. Para sua escolha utilizou-se o site CoinMarketCap[1], por ser um dos maiores sites de listagem e classificação de DLTs, que também provê informações gerais sobre os projetos e dados de mercado. Duas listas ordenadas por capitalização de mercado foram utilizadas, das quais cada uma forneceu duas DLTs, selecionando do topo, decrescentemente: a lista geral[2] (que resultou em Ethereum e Polkadot) e a lista por categoria de computação distribuída[3] (que resultou em DFinity e Holochain). A segunda lista foi escolhida em adição à primeira para evitar que o conjunto escolhido dependesse exclusivamente das plataformas com maior capitalização de mercado globalmente. Entre diversas categorias[4], optou-se por àquela de "computação distribuída" devido ao foco do trabalho em aplicações (web) em geral.

É importante sinalar que da lista global foram desconsideradas aquelas tecnologias especializadas em transações, aplicações e protocolos financeiros como Bitcoin, Ripple e Cardano. E da lista por plataforma de computação distribuída foram excluídas Filecoin e Bittorrent (que apareciam entre DFinity e Holochain), porque são específicas para armazenamento e compartilhamento de arquivos. Para esta filtragem, levou-se em conta a descrição das plataformas no que elas próprias se propõem a solucionar. O leitor deve considerar também que a seleção é relativa ao estado das listas na data deste trabalho.

Para avaliar e comparar o nível de interoperabilidade das plataformas, foram mapeados e determinados quatro aspectos de interoperabilidade, que são casos onde tem-se o objetivo de fazer com que dois serviços se comuniquem, sendo ao menos um deles uma aplicação distribuída, hospedada em DLT. A definição dos aspectos levou em conta as questões de interoperabilidade sugeridas em (ZEUCH; WÖHNERT; SKWAREK, 2019). A seguir são discriminados estes casos de interoperabilidade, representados como "A-B", onde a aplicação "A" utiliza uma funcionalidade oferecida pela aplicação "B". Uma aplicação é chamada "nativa" quando hospedada em DLT e "externa" caso contrário, isto é, não distribuída. Por exemplo, uma aplicação hospedada em um serviço de computação em nuvem.

---

[1] <https://coinmarketcap.com>, acessado em 10/08/2021.
[2] <https://coinmarketcap.com>, acessado em 10/08/2021.
[3] <https://coinmarketcap.com/pt-br/view/distributed-computing>, acessado em 10/08/2021.
[4] <https://coinmarketcap.com/cryptocurrency-category>, acessado em 10/08/2021.

a) Externa-nativa: concretiza-se em dois casos: (i) ao desenvolver uma aplicação qualquer externa a uma DLT (frequentemente referida como *off-chain* no contexto de blockchain), que faz uso da aplicação distribuída hospedada em uma DLT (aplicação nativa); e (ii) no desenvolvimento de uma interface de usuário para uma aplicação distribuída hospedada em uma DLT. No tocante ao caso (i), dado que aplicações web tradicionais (i.e., não hospedadas em DLT) são capazes de interoperar umas com as outras através de inúmeros métodos, como APIs REST, é fundamental saber o grau em que elas poderiam interoperar com aplicações nativas à DLT. Em relação ao caso (ii), interfaces de usuário (e.g., terminal, *web browser* ou aplicativo móvel) geralmente não são hospedadas em DLT e, portanto, requerem formas de integração com a aplicação hospedada na DLT (referida, às vezes, por *backend*). Neste caso, o conjunto de aplicação externa e aplicação nativa pode ser referido por *dApp*.
b) Nativa-nativa: operação entre aplicações hospedadas na mesma DLT. Similarmente ao caso (i) anterior, assim como uma aplicação web pode requisitar operações em outra, é importante compreender as facilidades e dificuldades de construir aplicações em DLT fazendo uso de outras já existentes. Mesmo que a comunicação entre aplicações nativas de uma DLT não ocorra necessariamente através da web usual (com servidores de DNS e HTTP), este aspecto auxilia na compreensão do funcionamento das plataformas e na comparação com o comportamento de aplicações web tradicionais.
c) Nativa-externa: refere-se a uma aplicação nativa à uma DLT que interopera com aplicações externas. Por exemplo, executando requisições para APIs HTTP, *webhooks* (mecanismo que permite uma aplicação saber de eventos ou mudanças que ocorrem em outra, em tempo real, sem necessidade de *polling*, isto é, constantemente efetuar consultas), armazenamento em nuvem. Blockchains focam em segurança e determinismo, portanto originalmente não suportam acesso a dados externos (e.g., requisições HTTP a aplicações web na nuvem). No entanto, os desenvolvedores de uma aplicação podem aceitar que haja uma dependência com um serviço centralizado. Nota-se que aqui não é considerada a comunicação entre blockchains, assunto coberto pelo tópico *d*.

O quarto aspecto de análise de interoperabilidade tem natureza ligeiramente distinta dos demais, pois considera exclusivamente aplicações em redes distintas:

d) Rede-rede: nível de interoperabilidade que uma DLT oferece para que suas aplicações nativas operem com outras aplicações nativas, hospedadas em outra DLT ou em rede diferente da mesma DLT.

Este aspecto mostra-se relevante pois conforme aplicações em redes distintas podem interoperar, há um maior grau de customização. Algumas blockchains modernas interconectam redes distintas (e.g., Polkadot), significando que aplicações podem estar sob condições de protocolos diferentes, e ainda assim operar entre si.

3 PLATAFORMAS DLT

Neste capítulo, são apresentadas e explicadas as quatro plataformas selecionadas. As redes de todas elas são naturalmente públicas, sendo possível definir restrições de acesso em alguns casos.

3.1 ETHEREUM

A Ethereum (BUTERIN, 2014) é uma DLT, cuja principal característica é o suporte a *smart-contracts*. Foi lançada em 2015, sendo a primeira DLT a hospedar aplicações genéricas. Seu algoritmo de consenso é do tipo prova de trabalho (ou *Proof-of-work*). Este algoritmo é baseado em mineradores e um tipo de desafio criptográfico (computacionalmente custoso). Mineradores são os validadores e produtores de blocos da rede, que competem para solucionar o desafio criptográfico primeiro. O bloco "minerado" que chega à maior parte da rede primeiro é adicionado à blockchain, e seu minerador ganha uma recompensa.

Assim como um nó da rede Ethereum possui todo o histórico de transações, ele possui também todos os *smart-contracts* hospedados na rede, e todas as suas transições de estado, que ocorrem de acordo com sua execução. Existem outros tipos de nós mais leves, mas isto não impacta diretamente em interoperabilidade. Aplicações, ou *smart-contracts*, são compiladas em *bytecode* para a EVM (Ethereum Virtual Machine). Solidity é a linguagem principal e original, baseada em JavaScript, embora hoje existam outras linguagens (e.g., mais simples, com menos funcionalidades ou de mais baixo nível), como Yul ou Vyper.

A execução de aplicações se dá por transações entre contas, sendo que cada *smart-contract* é associado a uma conta especial, gerenciada por código, referida por "conta de contrato". Uma conta emissora, que requisita a execução de um *smart-contract*, é também do tipo "conta de contrato" ou do tipo "conta externa", controlada por aqueles com a chave privada (e.g., um usuário de aplicação). Transações para uma "conta de contrato" alimentam o *smart-contract* com unidades de computação, que servem como pagamento para a execução e armazenamento, referido por *gas*. Ou seja, unidades de *gas* são consumidas conforme as operações de um *smart-contract*. Mineradores da prova de trabalho são os responsáveis pela computação. Qualquer entidade pode participar da rede ou utilizá-la.

3.2 DFINITY

A DFinity ou DFinity Foundation é uma organização sem fins lucrativos fundada em 2016 por Dominic Williams. Ela desenvolve o Internet Computer (IC), uma DLT. A proposta do IC é a de servir como uma plataforma única para o desenvolvimento e hospedagem de aplicações web dos mais variados tipos. Em outras palavras, uma plataforma de computação em nuvem (como Heroku, Azure ou AWS), só que descentralizada. Essencialmente, trata-se de uma plataforma de computação distribuída cujo modelo de computação é baseado em *smart-contracts*.

A infraestrutura do IC é provida por *data centers* independentes, cujos computadores participam como nós de uma blockchain, a qual tem como algoritmo de consenso um tipo de prova de autoridade (HANKE; MOVAHEDI; WILLIAMS, 2018). Os nós compõem diferentes sub-redes, referidas por *subnets*. Cada *subnet* hospeda uma parte das aplicações da rede inteira. Naturalmente, todos os nós de uma *subnet* replicam código e estado das mesmas aplicações, bem como as executam. A partir daqui o IC será referido por DFinity, para facilitar o entendimento ao longo do texto, onde há outras siglas, e porque a tecnologia é frequentemente assim citada.

As aplicações na DFinity são chamadas de *canisters*, que são como *smart-contracts*, no sentido de que múltiplos nós da blockchain possuem e executam um mesmo *canister*, e as transições de seu estado requerem consenso. *Canisters* são também módulos WebAssembly que armazenam código e estado. Hoje, há suporte principalmente para as linguagens Rust e Motoko (desenvolvida especialmente para a DFinity e baseada em programação orientada a agentes). Em teoria, qualquer linguagem com suporte a WebAssembly poderia ser utilizada para escrever *canisters* na DFinity.

Cada aplicação está associada a uma conta especial de desenvolvedor, a qual deve conter unidades de computação chamadas *cycles*, que são consumidas conforme a execução. Isto funciona como um pagamento pela execução. *Canisters* seguem o modelo de computação baseado em atores (HEWITT; BISHOP; STEIGER, 1973), isto é, representam um tipo de entidade que encapsula dados e comportamento, podendo: (i) comunicar-se com outros

canisters através de mensagens; (ii) modificar seu próprio estado em resposta a uma mensagem; (iii) criar outros atores. Existem dois tipos de chamadas para *canisters*:

a) *query* – consulta ao estado atual, ou chamada síncrona de função que não persiste mudanças de estado (desempenho é favorecido em relação a segurança);
b) *update* – execução de função que altera estado, portanto requer consenso e é assíncrona (segurança é favorecida em relação a desempenho).

3.3 HOLOCHAIN

A Holochain (HARRIS-BRAUN; LUCK; BROCK, 2018) é um framework para o desenvolvimento de aplicações P2P (*Peer-to-Peer*), com garantias de integridade e autenticidade dos dados. A rede P2P de uma aplicação, que conecta usuários, é baseada em uma DHT (Distributed Hash Table). É através da DHT que usuários são identificados e endereçados. Dados de uma aplicação também são identificados, bem como compartilhados, validados e replicados, através da DHT. Uma aplicação Holochain é chamada de *hApp*. O restante desta seção explica o modelo de computação da Holochain e, em seguida, seu funcionamento.

3.3.1 Modelo *agent-centric* & biomimética

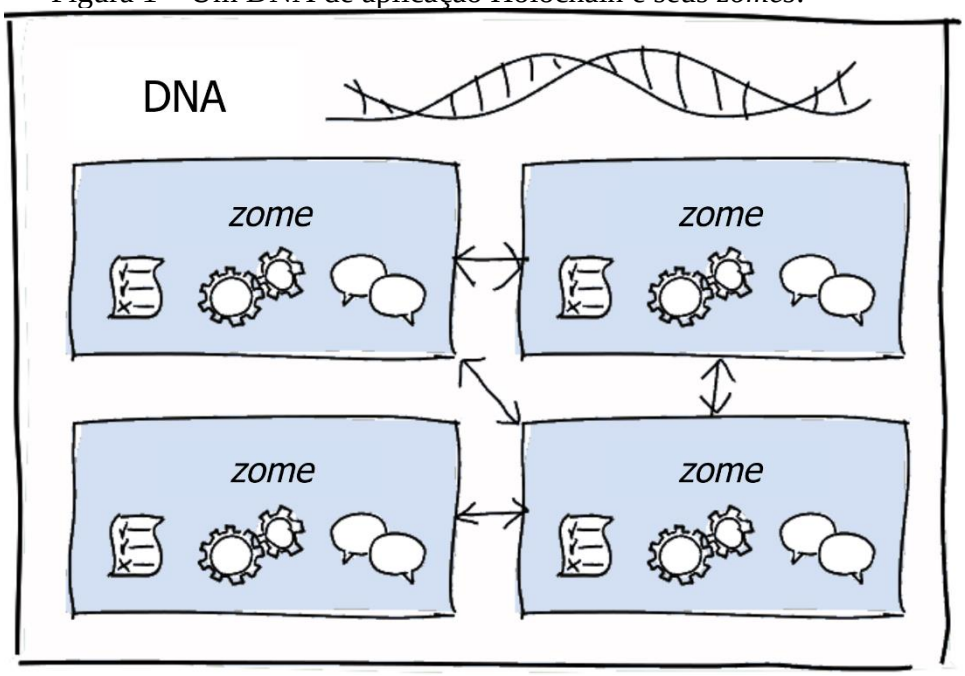
Para compreender esta tecnologia deve-se ter em mente dois de seus pilares. O primeiro é relativo ao seu modelo de computação, chamado de *agent-centric* (centrado no agente): um agente é a representação de um usuário na rede. Toda ação na rede é disparada por um agente, e a interpretação dos dados é relativa ao seu ponto de vista. Cada usuário executa uma aplicação em seu próprio dispositivo, e seu agente cria e armazena apenas os próprios dados, interagindo diretamente (ponto-a-ponto) com outros agentes. Isto será esclarecido mais adiante. O segundo pilar é a biomimética: a Holochain foi desenvolvida baseada em estruturas biológicas e em padrões da natureza. O código de uma aplicação chama-se DNA (ácido desoxirribonucleico), ilustrado na Figura 1. O DNA define tipos de dados, regras de validação para estes dados e a lógica da aplicação em si, e é composto por módulos, chamados de *zomes* (ou “cromossomos”). *Zomes* definem funções que farão parte da API pública de um *hApp*. A combinação de um DNA e do agente que o executa é referida por célula (*cell*).

Todos os nós que executam o mesmo DNA conectam-se a uma mesma DHT – neste sentido, um DNA pode ser compreendido como análogo a um *smart-contract*, dado que faz

todos os nós operarem sob as mesmas regras (adiante são explicadas as garantias de segurança e integridade). Em outras palavras, todas as células de mesmo DNA fazem parte de um mesmo sistema ou organismo. Uma célula tem vida própria e suas ações (produção de dados) independe das ações das outras células, ou seja, não há sincronização ou acordo entre todas as células do sistema acerca de uma verdade absoluta sobre os dados. Na seção seguinte, são descritos os componentes técnicos deste funcionamento.

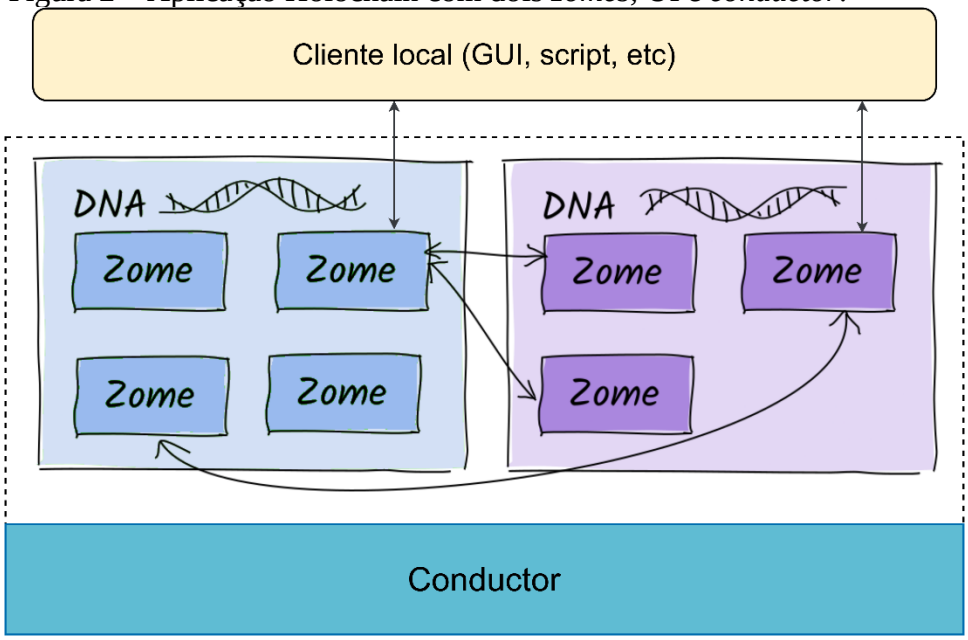
Conforme ilustra a Figura 2, o serviço sobre o qual estão hospedadas todas as células de um usuário e que executa em seu dispositivo, é chamado *conductor*. Ele expõe as APIs de DNAs expostas para clientes locais, além de ser responsável pela execução, armazenamento e comunicação entre *zomes* de um mesmo DNA, instâncias de aplicações de um mesmo usuário, e um agente e a rede externa.

Figura 1 – Um DNA de aplicação Holochain e seus *zomes*.



Fonte: adaptado de “Holochain – Application Architecture”.

Figura 2 – Aplicação Holochain com dois *zomes*, UI e *conductor*.



Fonte: adaptado de “Holochain – Application Architecture”.

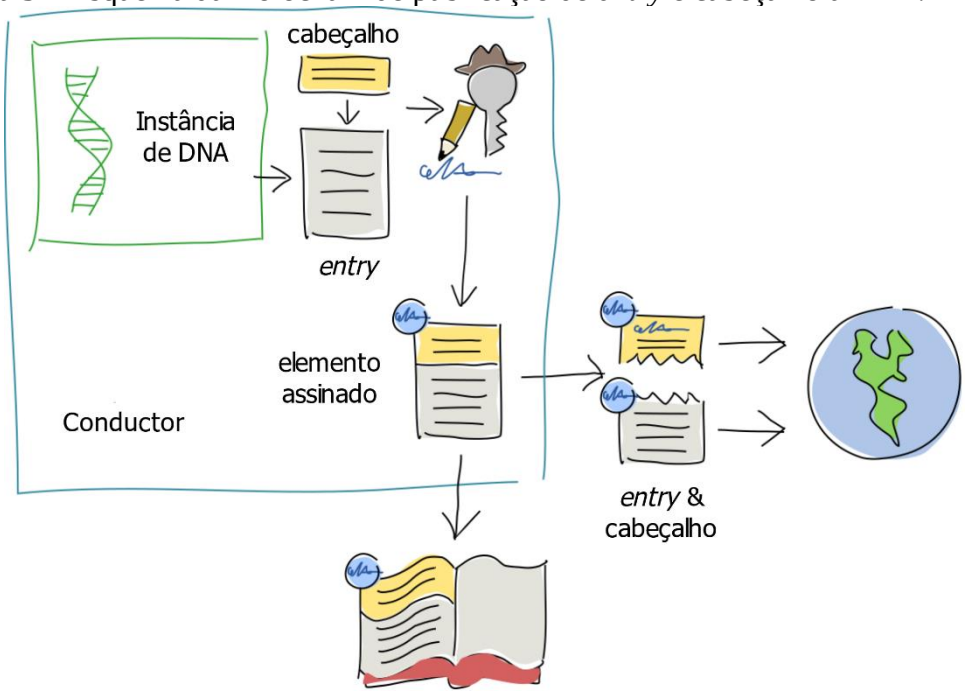
3.3.2 Source chain & integridade dos dados

Cada agente possui um registro cronológico local de suas ações, em uma estrutura de dados do tipo “lista de hash” (chamada de *source chain*). Dados armazenados na *source chain* são chamados de elementos. Eles são apenas adicionados, mas nunca excluídos ou modificados. Um elemento é composto por uma *entry* e um cabeçalho, e é sempre assinado digitalmente. A *entry* contém os dados em si, enquanto o cabeçalho contém o hash do elemento anterior na *source chain*, o instante atual e o tipo da *entry*. Em suma, um elemento adicionado à *source chain* representa o registro de uma ação do usuário. O tipo de uma *entry* dá semântica aos dados, permitindo sua interpretação (como o esquema de uma tabela de banco de dados). Cada tipo pode estar atrelado a funções de validação definidas pelos desenvolvedores de aplicação, as quais servem para a verificação da corretude dos dados recebidos por um agente.

Para o compartilhamento de dados entre usuários, elementos são publicadas na DHT, conforme ilustra a Figura 3, sendo identificadas por seu hash. O livro na parte de baixo representa a *source chain*. O círculo azul adicionado ao elemento é a assinatura digital do agente. À direita, há a representação da publicação do cabeçalho e *entry* na rede (DHT). No caso de uma *entry* privada, apenas o cabeçalho é publicado.

Cada agente possui um endereço na DHT, e aqueles que estão na vizinhança do hash do elemento publicado serão responsáveis por sua validação e redundância – eles mantêm uma cópia, além de comunicarem-se com outros agentes por meio de troca de mensagens (protocolo de *gossip*), contribuindo para a resiliência e disponibilidade da rede. Ao receber dados de um usuário, um agente pode tanto identificar sua invalidade através da execução das funções de validação do DNA, bem como através da checagem da integridade da *source chain* do autor. Se um comportamento malicioso é identificado, os agentes da aplicação podem avisar uns aos outros através do protocolo de *gossip*, executando uma resposta em conjunto, como bloquear o agente malicioso, o ignorando. Este mecanismo é chamado de *immune system* (ou "sistema imunológico"): dado um "corpo estranho", as células do sistema agem em conjunto para neutralizá-lo.

Figura 3 – Esquema da Holochain de publicação de *entry* e cabeçalho à DHT.



Fonte: adaptado de "Holochain – The Distributed Hash Table: a public data store".

3.4 POLKADOT

A Polkadot (WOOD, 2016) é uma rede heterogênea de múltiplas blockchains interoperáveis. É composta por dois conceitos centrais: *relay chain* e *parachain*. *Relay chain* é a blockchain central, com algoritmo de consenso do tipo *proof-of-stake* (KIAYIAS et al., 2017), responsável pela interconexão entre todas as blockchains do ecossistema. Uma *parachain* é uma blockchain especializada para casos de uso específicos que se conecta à *relay chain*, tendo seu próprio protocolo de validação, governança e funcionalidades. Por exemplo, uma *parachain* pode ser uma blockchain especializada em transações monetárias anônimas, *smart contracts* para DAOs (Decentralized Autonomous Organizations), oráculos, etc. A blockchain central executa alguns tipos específicos de transação, com o objetivo de coordenar todo o sistema. O funcionamento entre *relay chain* e *parachains* envolve principalmente duas entidades (ilustradas na Figura 4):

a) *validators* – (ou validadores) são nós da *relay chain* e os principais atores para a segurança da Polkadot, periodicamente e randomicamente atribuídos a uma *parachain*, para qual devem produzir, validar e ratificar blocos, os adicionando à *relay chain*. Como não é razoável esperar que cada validador faça isso para todas as *parachains*, a tarefa de produção de blocos é terceirizada para os *collators*. Assim, os validadores recebem provas de transições de estado dos *collators*;
b) *collators* – auxiliam os validadores na produção de blocos de uma parachain. Um *collator* é nó completo tanto de uma *parachain* como da *relay chain* e que coleta transações da primeira criando candidatos a blocos, dos quais são produzidas provas de transição de estado. Estes blocos, junto de suas provas, são enviadas aos validadores.

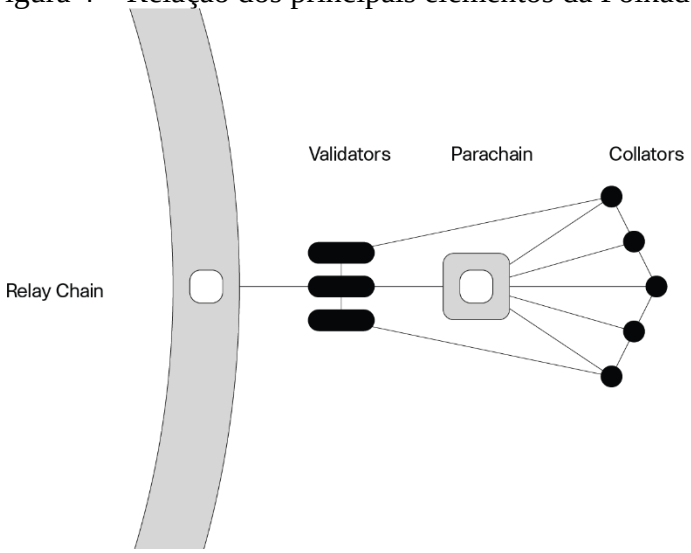
A *relay chain* é responsável pela segurança de cada *parachain*, pois seus validadores garantem a finalização dos blocos destas (i.e., garantem a irreversibilidade dos blocos). O algoritmo de finalidade utilizado é o GRANDPA (GHOST-based Recursive Ancestor Deriving Prefix Agreement) (STEWART, 2019). A operação com um único algoritmo de finalidade é o que permite a comunicação entre *parachains* , a qual dá-se por meio da *relay chain* através dos *collators*. Para este fim, é utilizando um protocolo chamado XCMP (Cross-Chain Message Passing)[5].

[5] <https://research.web3.foundation/en/latest/polkadot/XCMP/index.html>, acessado em 10/08/2021.

A veracidade ou integridade dos dados providos por um *collator* não é responsabilidade dos validatores, mas do ecossistema de cada Parachain. Ou seja, a *relay chain* apenas finaliza os blocos e não se responsabiliza se os *collators* são confiáveis ou não.

Parachains são geralmente baseadas no Substrate[6], um framework na linguagem Rust para desenvolvimento modular de blockchains. Com o Substrate desenvolvedores podem criar blockchains facilmente, selecionando módulos para funcionalidades específicas (e.g., suporte a *smart-contracts*, criptomoedas, votação ou identidade). Estes módulos são chamados *pallets*. Blockchains baseadas no Substrate não precisam necessariamente fazer parte da Polkadot (como uma *parachain*), podendo ser independentes. O componente que conecta uma blockchain baseada em Substrate à Polkadot chama-se Cumulus[7]. *Parachains* utilizam WebAssembly como alvo da compilação, bem como o Substrate.

Figura 4 – Relação dos principais elementos da Polkadot.



Fonte: documentação da Polkadot.

[6] <http://substrate.dev/>, acessado em 10/08/2021.
[7] < https://github.com/paritytech/cumulus>, acessado em 10/08/2021.

4 ANÁLISE E COMPARAÇÃO

Este capítulo apresenta uma análise e comparação das DLTs selecionadas quanto aos quatro níveis de interoperabilidade definidos no Capítulo 2, dispondo de uma seção para cada um deles. O Quadro 1 acompanha as seções e provê uma síntese das características das plataformas, servindo para consulta e visão geral de interoperabilidade. Finalmente, é fornecida uma conclusão da comparação na Seção 4.5, em forma de ranqueamento das plataformas. Relembra-se o leitor que os níveis de interoperabilidade são:

a) Externa-nativa: entre uma aplicação externa à uma DLT e uma aplicação nativa, sendo a interface de usuário da aplicação nativa um caso especial de aplicação externa;
b) Nativa-nativa: entre duas aplicações hospedadas em uma mesma DLT;
c) Nativa-externa: entre aplicação hospedada em DLT e aplicação externa, não hospedada em DLT;
d) Rede-rede: entre duas aplicações nativas a DLT, mas que não estejam em uma mesma rede, com as redes sendo de uma mesma DLT ou de DLTs distintas.

Quadro 1 – Resumo das características de interoperabilidade das DLTs Ethereum, DFinity, Polkadot e Holochain.

Tipo	Ethereum	DFinity	Holochain	Polkadot
Externa-nativa	Interface JSON-RPC sobre WebSocket ou HTTP. Há bibliotecas para diversas linguagens que abstraem as chamadas RPC. Há ferramentas de geração de código para que chamadas a aplicações nativas sejam como chamadas locais. Conexão a nó local ou através de provedores.	Interface HTTP. Possui a IDL Candid, que define interface de serviços, suportando inclusive parâmetros opcionais e funções e serviços como *first-class citizens*. Com o Candid há suporte integrado a geração de código, para que chamadas a aplicações nativas sejam como chamadas locais. É possível hospedar artefatos de front-end (“Add front-end assets - Internet Computer”, [s.d.]).	Interface JSON-RPC sobre WebSocket ou HTTP. Conexão a nó local.	Com a adição de um módulo (*pallet*), suporta interface JSON-RPC sobre WebSocket ou HTTP. Permite a implementação de RPCs customizadas. Há bibliotecas para diversas linguagens. Conexão a nó local ou através de provedores.

		Conexão remota através de provedores (*data centers* independentes).		
Nativa-nativa	Comunicação síncrona, direta ou indireta. Direta se a interface do contrato alvo for declarada no contrato chamador, indireta caso contrário. Chamada direta é facilitada com interoperabilidade total de tipos. Aplicações são endereçadas por endereço hexadecimal de contrato.	Comunicação assíncrona e direta por troca de mensagens. Chamada abstraída como chamada de função direta. Interface deve ser conhecida, havendo interoperabilidade total de tipos, mesmo para linguagens diferentes (e.g. Motoko e Rust), possibilitada através do Candid. Aplicações identificadas por endereço alfanumérico de *canister*.	Comunicação síncrona ou assíncrona, indireta. Ocorre através de funções auxiliares (call, remote_call). Interoperabilidade total de tipos se a chamada for entre *zomes*, apenas JSON se for entre DNAs. Aplicações identificadas por endereço hexadecimal de um DNA e chave pública do agente.	Suporte a *smart-contracts* se dá ao importar o *pallet* "contracts". Comunicação direta e síncrona. Interface deve ser conhecida, havendo interoperabilida de total de tipos. Aplicações identificadas por endereço hexadecialm de contrato.
Rede-rede	Não se aplica. Todas as aplicações na Ethereum que podem comunicar-se são hospedadas em uma única rede, onde elas operam sob as mesmas condições.	Não se aplica. A DFinity é composta por diferentes redes, chamadas *subnets* ou "sub redes", porém todas operam sob as mesmas condições. O fato da aplicação sendo integrada estar em uma *subnet* diferente é transparente à chamadora.	Interoperabili-dade total. Uma rede (autônoma) por aplicação. Cada rede é customizável e não há nenhum impacto direto para interoperabilidade "rede-rede". Chamadas idênticas àquelas entre aplicações ("nativa-nativa").	Interoperabilida de com algumas condições e restrições. Para redes baseadas em Substrate, há um módulo específico para interoperabilida de. Troca de mensagens assíncronas que passa da rede remetente pela rede central *relay chain* até a rede destino. Necessário que redes implementem algoritmo de finalidade GRANDPA.

				Ainda não suporta comunicação rede-rede a nível de contrato. Para redes não baseadas em Substrate, é necessário implementação completa de *bridging*, baseado em XCMP.
Nativa-externa	Não suportado. Esquema de "oráculos" são amplamente utilizados. Integração com aplicações externas facilitada por mecanismo de eventos, disparados por aplicações nativas.	Não suportado. Deduz-se que uma aplicação externa que executa consultas periódicas para identificar mudanças no estado de uma aplicação seja a alternativa. Artefatos de front-end podem ser providos nativamente (os quais poderiam executar requisições web normalmente).	Não suportado. Esquema de "oráculos" é uma alternativa. Integração com aplicações externas facilitada por mecanismo de eventos, disparados por aplicações nativas.	Limitado. A aplicação envia a requisição para uma aplicação especial (*off-chain worker*), que por sua vez executa o que é necessário, podendo submeter transações (assinadas ou não) de volta ao *runtime* principal *on-chain*. Suporta emissão de eventos a partir de contratos. Documentação insuficiente para avaliar este mecanismo.

Fonte: elaborado pelo autor

4.1 EXTERNA-NATIVA

A comunicação a partir de uma aplicação externa é sempre feita com um nó da rede, para qualquer DLT. A seguir discorre-se primeiramente sobre integração sem o uso de ferramentas específicas, ulteriormente considerando bibliotecas.

Todas as plataformas suportam uma interface JSON-RPC sobre HTTP e, com exceção da DFinity, também WebSockets. Para a Holochain, estas chamadas são feitas à uma instância local do nó da DLT (*conductor*), assim como para a Ethereum e Polkadot. Porém, para as duas últimas, há a opção de conexão à rede através de provedores ("Nodes as a service - ethereum.org") ("Providers - polkadot.js"). Provedores são serviços web tradicionais que oferecem acesso à blockchains. São hospedados em nuvem, provisionando e gerenciando nós de blockchains, de forma que desenvolvedores de aplicações externas possam facilmente conectar-se à uma DLT sem precisar empregar tempo e esforço em sua manutenção. Isso não necessariamente é desejável, pois dependendo da proporção de nós da rede provisionados dessa forma, há uma centralização do acesso e operação da DLT.

A conexão com aplicações nativas da DFinity é análoga àquela através de provedores. É sempre remota, pois os nós não são executados por qualquer pessoa ou entidade, e sim apenas por provedores ("*data centers* independentes") aprovados, que executam em hardware específico. A aplicação externa envia requisições HTTP para URLs do tipo "identificador-de-canister.ic0.app", que chegarão a um nó aleatório que serve a aplicação nativa desejada (um *canister*). Em teoria, as *subnets* possuem endereços conhecidos, podendo responder quais os endereços IP dos nós de um *canister* específico. Entretanto, a especificação da DFinity ("The Internet Computer Interface Specification - Internet Computer", 2020) e demais materiais disponibilizados[8] não deixam claro como funciona a resolução de um endereço IP a partir da URL terminada em "ic0.app".

Para todas as DLTs, a requisição do sistema externo deve especificar a aplicação e função alvos, bem como cada parâmetro e seus valores. Endereços de aplicação Polkadot são endereços de um *smart-contract* da blockchain (cadeia de caracteres hexadecimais), assim como endereços na Ethereum. Porém, contratos e identificadores de contas da Ethereum podem também ser endereçados por nomes, através do serviço ENS[9], que é análogo a um DNS. Esta forma é menos sujeita a erros: facilita a leitura, comunicação entre pessoas de um time, depuração, etc. Contudo, o ENS é um serviço implementado sobre a Ethereum e tem um custo. Já na DFinity, endereços são identificadores de c*anister* (cadeia de caracteres alfanuméricos).

[8] <https://dfinity.org/technicals>, acessado em 15/10/2021.
[9] <https://eips.ethereum.org/EIPS/eip-137>, acessado em 15/10/2021.

Na Holochain é utilizado o "*cell id*", composto pelo hash do *DNA* (aplicação) e a chave pública do agente. Além disso, especifica-se o nome do *Zome* (módulo) da função alvo.

Embora existam bibliotecas (oficiais ou não) que abstraem as chamadas JSON RPC para todas as plataformas, é relevante saber quais protocolos de comunicação os nós suportam, para os casos em que a aplicação externa sendo desenvolvida independa destas ferramentas. Por exemplo, quando existe alguma especificidade de linguagem, framework, biblioteca de requisições de rede, seja por compatibilidade com código existente, familiaridade dos desenvolvedores com determinadas ferramentas ou imaturidade das bibliotecas das DLTs. Nesses casos, pode ser preferível implementar uma ferramenta própria para a comunicação com o serviço distribuído, e neste aspecto, embora a DFinity seja a única que não suporta WebSockets, seu suporte a HTTP é ligeiramente superior às demais, pelo fato específico de aplicações nativas a ela poderem definir interfaces HTTP ("Introducing a new approach to handling HTTP requests and serving assets - Internet Computer") que suportam a execução de funções do tipo *query*. Vale ressaltar que ao dizer que as demais DLTs suportam HTTP, significa que um nó disponibiliza uma interface HTTP genérica, que recebe requisições e trata de traduzi-las a chamadas nativas. A DFinity também suporta esta forma.

Idealmente, o suporte por aplicações distribuídas a interfaces HTTP pode, além de oferecer as vantagens citadas anteriormente, facilitar a migração de um sistema em *cloud* para uma solução distribuída, já que sua API poderia se manter igual ou parecida, reduzindo o trabalho necessário para adaptar os códigos dos clientes. Canisters não suportam este grau de interoperabilidade pois, como já mencionado, ainda não suportam diferentes métodos HTTP, embora a discussão já tenha sido levantada, e algumas provas de conceito propostas[10]. Na verdade, esta funcionalidade foi adicionada com o intuito de melhorar o suporte a front-ends web de aplicações nativas (ao invés de aplicações externas quaisquer): aplicações na DFinity têm seus artefatos de front-end (arquivos JavaScript, HTML, CSS, imagens) hospedados diretamente na plataforma. Ou seja, ao invés de páginas Web serem obtidas de servidores de *cloud*, elas são disponibilizadas por um *canister*. Isto é possível através das requisições que são respondidas diretamente por *canisters*.

Ao considerar bibliotecas disponibilizadas pelas próprias DLTs, é importante considerar o nível de detalhes abstraídos para as chamadas JSON-RPC com respeito a modelo de programação e dados, interoperabilidade semântica, e a quantidade de opções de linguagens.

[10] <https://github.com/dfinity/agent-rs/pull/195>, acessado em 16/08/2021.

Todas as DLTs disponibilizam bibliotecas que auxiliam na comunicação com serviços nativos. Aquelas para Ethereum, Polkadot e Holochain possuem abstrações em nível semelhante enquanto que para a DFinity há um destaque, por possuir implementações de uma IDL (Interface Description Language). Como Ethereum e Polkadot são blockchains, a forma de utilização das bibliotecas é bastante parecida, contudo, para Ethereum há uma biblioteca em Java (Web3j) que fornece geração automática de código, de forma que o contrato seja representado em alto nível, como um objeto Java, eximindo programadores de lidar com conversão de dados, por exemplo. Para a Polkadot, as bibliotecas também permitem a criação de um elemento que represente um contrato, porém o mapeamento de dados fica a cargo do programador. Com a biblioteca para a Holochain (há apenas uma em JavaScript[11]), a questão dos dados é parecida, embora não haja uma representação da aplicação como um objeto ou interface. De forma geral, exceto pela geração de código da Web3J para a Ethereum e salvo que não há uma interação explícita e direta sobre HTTP, as integrações da Ethereum, Polkadot e Holochain são análogas ao desenvolvimento web tradicional, onde os clientes criam DTOs (Data Transfer Objects), especificam URLs e URIs etc. Entretanto, a implementação do Candid (“Candid Library for the Internet Computer”, 2020), IDL para os serviços DFinity, destaca esta plataforma.

Serviços na DFinity podem ser escritos em diferentes linguagens. O propósito principal do Candid é descrever interfaces de serviços de forma independente de linguagem. É possível descrever funções, parâmetros de entrada e saída, inclusive parâmetros opcionais, que podem ser adicionados sem a quebra de compatibilidade, tipos especiais (como structs). Além de dados, especificações nesta IDL também suportam a descrição de referências a outros serviços e funções como parâmetro e retorno. Suas implementações mapeiam automaticamente os tipos de um *canister* para os tipos correspondentes da linguagem cliente, não sendo necessário serializar ou deserializar tipos Candid abstratos manualmente. Em termos de desenvolvimento web, o resultado é que o cliente JavaScript de um serviço nativo executa chamadas como se estivesse operando com um objeto ou módulo JavaScript local. Este comportamento é semelhante àquele da Web3J, embora a DFinity proveja suporte nativo ao Candid. A representação local do serviço nativo é gerada automaticamente pelo compilador

[11] <https://github.com/holochain/holochain-conductor-api>, acessado em 16/08/2021.

Candid[12], cuja entrada é um arquivo “.did”, que por sua vez é criado manualmente, ou também automaticamente, por uma ferramenta[13].

4.2 NATIVA-NATIVA

Serviços web podem operar entre si através de webhooks, chamadas síncronas de API, troca de mensagens através de um serviço de mensageria, *etc.* Esta seção visa explicar as formas através das quais dois serviços nativos de uma plataforma podem interoperar, expondo suas vantagens e limitações. Algumas DLTs modernas são compostas por redes heterogêneas interligadas, como é o caso da Polkadot e Holochain. Este caso é arrazoado na seção referente ao aspecto "rede-rede". O aspecto nativo-nativo limita-se a aplicações em uma única rede.

A interação entre aplicações na Ethereum ou na Polkadot é muito parecida, seguindo os mesmos padrões de desenvolvimento de *smart-contracts* (referidos aqui por “contratos”). Cada contrato tem um conjunto de funções públicas e dados, que compõem sua interface. Um contrato pode interagir com outro sabendo seu endereço e interface (incluindo dados de entrada e saída). Caso a interface não seja conhecida, é possível fazer chamadas especificando a função alvo por seu nome. No contrato chamador, é possível declarar a interface do contrato alvo e então instanciá-la, através de seu endereço. Também é possível instanciar e hospedar um *smart-contract* inteiramente novo, a partir de outro, com um novo endereço e seu estado inicial. Através da instância fazem-se chamadas síncronas e determinísticas, que são executadas localmente no mesmo nó que processa o *smart-contract* chamador.

Chamadas entre contratos não geram novas transações, apenas chamadas iniciadas externamente (e.g., por um usuário ou software agindo em nome do usuário). Os efeitos de uma chamada são efetivados apenas após todos os nós da rede terem executado as mesmas operações, culminando na inclusão do novo estado em um novo bloco.

A Ethereum suporta um tipo especial de chamada entre contratos: com a operação DELEGATECALL é possível fazer com que o contrato sendo chamado execute em nome do usuário que disparou o contrato chamador. Em uma chamada normal o contrato sendo chamado por outro tem o primeiro como seu chamador, ao invés do usuário. Além disso, as variáveis de estado do primeiro contrato são compartilhadas com o segundo. Isto significa que ambos os

[12] <https://github.com/dfinity/candid/tree/master/tools/didc>, acessado em 13/08/2021.

[13] <https://sdk.dfinity.org/docs/developers-guide/cli-reference/dfx-parent.html>, acessado em 13/08/2021.

contratos devem possuir o mesmo layout de variáveis: mesmos tipos e mesma ordem de declaração. Em relação a uma interação entre serviços web tradicionais, o comportamento de DELEGATECALL seria análogo a solicitar que o serviço sendo chamado modifique o banco de dados da aplicação chamadora. É importante ressaltar que *smart-contracts* tanto na Ethereum quanto na Polkadot possuem limitações ao lidar com dados dinâmicos. Por exemplo, não é possível que um contrato retorne para outro um dado do tipo *string* ou uma estrutura de dados tipo mapa. Isto é uma restrição comum de *smart-contracts*, relacionada a imprevisibilidade de custos de execução.

Dado que o modelo de programação da DFinity é baseado em atores, um *canister* pode ser visto como um objeto especial, com memória isolada, comunicando-se por troca de mensagens assíncronas, passando dados binários de valores Candid codificados. O Candid é responsável por prover compatibilidade total de dados aos *canisters* envolvidos em uma comunicação, mesmo que eles sejam implementados em linguagens diferentes. Uma única mensagem é processada por vez, eliminando condições de corrida. Cada emissão de mensagem é associada a uma função de *callback* que processa o resultado da chamada, como naturalmente funcionaria um cliente não bloqueante de *web service*. Todavia, a comunicação entre micro serviços através de troca de mensagens, por exemplo, é mais complexa que isso, já que os resultados são totalmente independentes dos envios. Ademais, há uma dificuldade maior em garantir o formato dados.

Toda mensagem é emitida por um *canister* como uma chamada de função e todo processamento de mensagem é definido por uma função. Funções que alteram o estado de um ator requerem concordância entre as réplicas da DLT, portanto são efetivadas e retornam um resultado apenas após consenso. Isso é um processo bastante caro. Estas funções são chamadas de *update* (“Canisters and code - Internet Computer”, 2020). Uma função *query*, que não persiste mudanças de estado, possui latência bem inferior. Porém, não pode enviar mensagens para outros serviços, por restrições impostas pelo protocolo subjacente da DLT. Isto significa que, embora uma função *query* executada pelo usuário (externo) possa ser resolvida rapidamente, ela não pode comunicar-se com outros serviços. Um *canister* pode importar outro, e instanciá-lo, com um estado completamente novo, ou pode usar uma referência para um *canister* existente na rede, a partir de seu identificador (como na Ethereum e Polkadot) (“Language quick reference - Actor references - Internet Computer”).

Na Holochain existem quatro tipos de chamadas entre aplicações. Todos são *chamadas de procedimento remoto* (*RPC*): "chamada de cliente", "chamada entre *zomes*", "chamada *bridge*" e "chamada remota". A primeira foi descrita na seção anterior. A segunda, apesar de ocorrer dentro de uma mesma aplicação, será comentada a seguir pois é uma chamada indireta. As chamadas "bridge" e "remota" ocorrem entre aplicações e serão comentadas em breve. Recorda-se que toda funcionalidade de uma aplicação nativa é contida em um *DNA*, isto é, nos *zomes* ("cromossomos" ou "módulos") individuais de um *DNA*. Qualquer função pública dos *zomes* é exposta pelo *conductor* como parte da API pública de uma aplicação.

Ainda que *zomes* façam parte da mesma aplicação, uma "chamada entre *zomes*" não é direta, e sim através da função *hdk::call*[14]. Dados de parâmetro e resposta são codificados em JSON, havendo suporte para serialização e deserialização. É positivo que seja possível utilizar JSON entre aplicações, pela sua simplicidade e independência de linguagem. No entanto, isto é inusual para chamadas de uma mesma aplicação e adiciona complexidade.

Uma "chamada *bridge*" é igualmente efetuada através da função *hdk::call*. Dado que esta é uma chamada entre aplicações, vale ressaltar que ela ocorre em um mesmo nó, e que as *source chains* de cada aplicação são travadas durante a comunicação, facilitando a lógica da aplicação chamadora no que tange o nível de confiança acerca dos dados que lhe estão disponíveis. Ainda, é notável a facilidade e simplicidade arquitetural do compartilhamento de dados de um mesmo usuário entre aplicações. Como na Holochain ambas as aplicações executam localmente e em nome do usuário, não há barreiras para a troca de dados. Em aplicações web tradicionais onde há barreiras organizacionais, o análogo necessitaria da implementação de um protocolo como o OAuth 2.0 ("The OAuth 2.0 Authorization Framework", 2012). Em uma aplicação web ou aplicação em blockchain, guarda-se os dados de todos os usuários, adicionando complexidade, sem contar que para a operação ser possível ou efetuada, é necessário conexão com a internet. Como o leitor pode indagar, esta arquitetura da Holochain facilita também questões de privacidade.

Chamadas do tipo "chamada remota" ocorrem quando a instância da aplicação de um agente executa uma função da mesma aplicação em outro agente. De fato, não se trata de uma chamada entre aplicações, mas sim entre instâncias de uma mesma aplicação. Isto posto, como o paradigma P2P da Holochain é peculiar, decidiu-se por incluir este caso sob o aspecto de interoperabilidade "entre" aplicações nativas. O agente sendo chamado executa em nome do

[14] <https://docs.rs/hdk/0.0.42-alpha4/hdk/api/fn.call.html>, acessado em 17/08/2021.

agente chamador. Isto é semanticamente correlato à DELEGATECALL dos contratos da Ethereum. Essa funcionalidade oferece algumas vantagens: troca de dados privados à *source chain* da aplicação sendo chamada; compartilhamento de dados de aplicações terceiras, disponíveis no nó da aplicação sendo chamada, mas não no nó da aplicação chamadora; troca de informações ponta a ponta, sem registros públicos.

Adicionalmente, a Holochain suporta a comunicação assíncrona entre agentes por meio de sinais (ou "eventos"). Uma “chamada remota” é síncrona e mais lenta. Ao invés dela, se um agente não se interessa pela resposta de uma chamada, ele pode emitir um sinal para um ou mais agentes e continuar sua execução. Este mecanismo também é utilizado para a comunicação com aplicações externas, como explicado na próxima seção.

4.3 NATIVA-EXTERNA

Nenhuma das DLTs suporta interoperabilidade com aplicações externas diretamente a partir de aplicações nativas. Em outras palavras, aplicações nativas não são capazes de fazer chamadas diretas a aplicações externas. Todavia, cada uma possui funcionalidades específicas que podem auxiliar nesta comunicação. De forma geral, o conceito de oráculos é central no contexto das quatro plataformas.

Para que operações externas sejam disparadas a partir de uma aplicação na Ethereum faz-se uso do mecanismo de *events* (“Contracts — Solidity 0.8.6 documentation”). Um *event*, na linguagem Solidity, é um conjunto de dados que pode ser emitido a partir de uma função de *smart-contract*, indicando a uma aplicação externa a ocorrência de uma operação. Isto pode disparar uma ação implementada por uma aplicação externa intermediária, interessada no evento, que por sua vez executa a ação na aplicação externa alvo (ou qualquer outra operação off-chain). Este padrão é similar ao padrão “barramento de eventos” ou “barramento de mensagens” (HOHPE; WOOLF, p. 137, 2003), comum na integração de aplicações web, que permite que aplicações interessadas em eventos específicos os consumam do barramento de forma assíncrona, sem acoplamento entre produtor e consumidor. A rede Ethereum age como o barramento e aplicações externas podem filtrar eventos. Redes de oráculos como a Chainlink (ELLIS; JUELS; NAZAROV, 2017) utilizam este mecanismo. Um *event* em Solidity é enviado no formato JSON, portanto é de fácil manipulação. Além disso há diversas bibliotecas para a

Ethereum que auxiliam a escuta de eventos. Eventos não podem ser utilizados para comunicação entre *smart-contracts*.

A Polkadot facilita a construção e integração de oráculos com o que é chamado de *off-chain worker* (*OCW*) (“Off-Chain Features - Substrate Developer Hub”). Trata-se de uma aplicação que executa junto a um nó de *parachain*, mas cujos dados não são passíveis de consenso. Um *OCW* executa em um ambiente WASM (Web-Assembly) isolado e é disparado a partir de uma chamada de função[15] do nó propriamente dito. *Off-chain workers* podem gerar números aleatórios, executar requisições a aplicações externas, executar operações demoradas demais para *smart-contracts* ou caras demais se executadas por *smart-contracts* (e.g. criptografar ou descriptografar). Ainda, podem submeter transações assinadas ou não assinadas de volta ao *runtime* do nó. Lógica *on-chain* e OCWs têm acesso a um banco de dados em comum, ao qual a primeira possui apenas acesso de escrita. Essa unidade de armazenamento pode também ser utilizada para comunicação entre diferentes threads de um OCW, e tem sua operação com lógica *on-chain* facilitada, já que pode facilmente acessar dados *on-chain* e seus códigos são declarados juntos. Todas essas funcionalidades facilitam o desenvolvimento de um mecanismo de oráculos, sendo importante salientar que a verificação dos resultados de OCWs fica por conta dos desenvolvedores (e.g. simplesmente “confiar”, checar assinatura ou votação). Sumariamente, o funcionamento é parecido com o da Ethereum. No entanto o mecanismo de *off-chain workers*, pelos elementos descritos acima, oferece um grau maior de interoperabilidade. De fato, há um acoplamento maior. A linguagem ink!, para o desenvolvimento de *smart-contracts* em Rust, amplamente utilizada na Polkadot, suporta a emissão de eventos de forma similar ao Solidity[16], mas as documentações disponíveis não deixam claro como aplicações externas podem escutá-los, nem o formato de dados transmitido. Também, não está claro se *smart-contracts* podem disparar *off-chain workers*.

O mecanismo que permite aplicações em Holochain dispararem ações em aplicações externas é o mesmo que permite a comunicação assíncrona entre agentes, explicado na seção anterior. Seu funcionamento é bastante parecido ao da Ethereum: os dados são transmitidos em JSON; a aplicação conectada por WebSockets ao nó Holochain inscreve-se para receber eventos na porta de um DNA específico.

---

[15] <https://substrate.dev/rustdocs/latest/frame_support/traits/trait.OffchainWorker.html>, acessado em 17/08/2021.

[16] <https://paritytech.github.io/ink-docs/macros-attributes/contract/#events>, acessado em 17/08/2021.

A DFinity não faz menção a oráculos ou mecanismos para interoperabilidade de canisters com aplicações externas. Uma vez que os nós da plataforma estão localizados em *data centers* independentes, e que canisters são implantados exclusivamente como módulos WASM, não há a possibilidade de hospedar aplicações que escutam eventos através dos nós, como é o caso para Ethereum, Polkadot e Holochain. À vista disto, deduz-se que para haver uma interação entre aplicação nativa e externa, iniciada por acontecimentos internos à DFinity, seria necessária uma aplicação externa intermediária que fizesse consultas periódicas à aplicação nativa, para identificar mudanças de estado, finalmente disparando ações à aplicação externa alvo. Para facilitar esta integração, diminuindo a complexidade da aplicação nativa, poder-se-ia implementar um canister especial, que registre "tarefas" a serem executadas externamente em uma estrutura de dados tipo fila, por exemplo. Assim, o componente intermediário faria consultas apenas neste *canister* de tarefas.

4.4 REDE-REDE

Esta análise de interoperabilidade é aplicável apenas à Holochain e Polokadot. Todas as aplicações Ethereum que podem se comunicar operam em uma única rede. Evidentemente, existem diferentes instâncias da rede Ethereum. No entanto, o universo de comunicação de uma aplicação hospedada em uma instância é limitado a sua própria rede. A DFinity funciona em subdivisões, chamadas de *subnets*, mas estas são partes da mesma rede, não havendo diferença de protocolo para aplicações em diferentes *subnets* (i.e., a DLT é homogênea). O detalhe de *canisters* estarem em *subnets* distintas é transparente para sua comunicação.

Esta seção explicará, respectivamente, a interoperabilidade da Polkadot com blockchains externas, interoperabilidade entre *parachains* e, por fim, interoperabilidade entre redes Holochain.

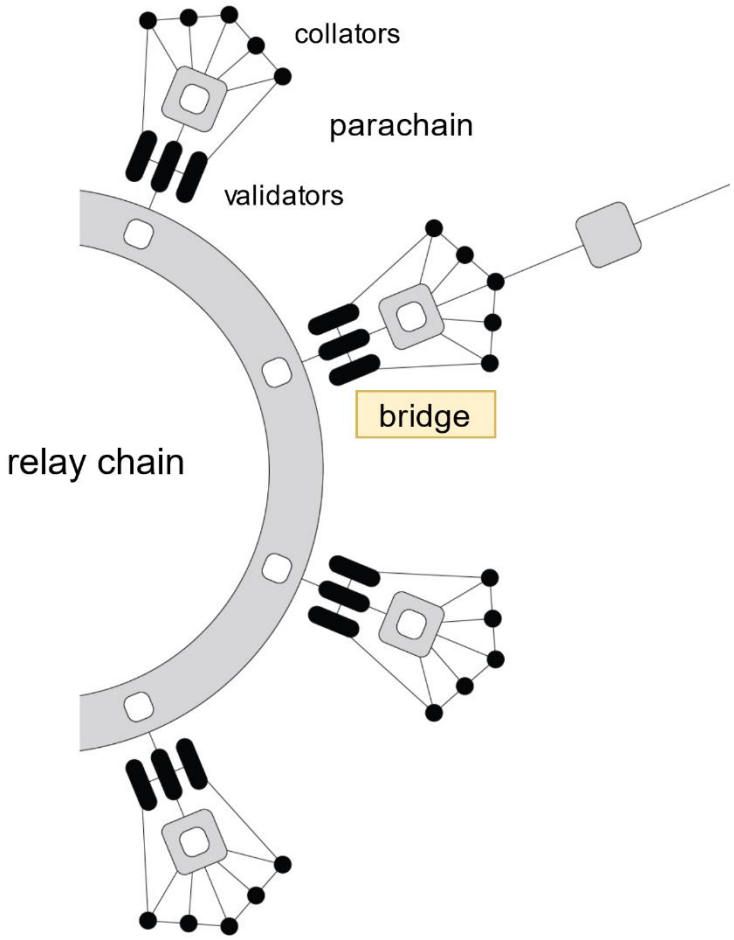
A Polkadot facilita a interoperabilidade entre *parachains* e uma blockchain externa através de *bridges* ("Bridges - Polkadot Wiki"). Uma *bridge* é um sistema que permite a interoperabilidade entre duas blockchains. No caso da Polkadot, uma *bridge* é um tipo especial de *parachain* (a partir daqui, será referida apenas por *bridge*), vide Figura 5. É importante destacar que *bridges* são apenas uma possibilidade de adicionar interoperabilidade entre a Polkadot e uma outra DLT (blockchain). Assim dizendo, há necessidade de serem construídas

pelos desenvolvedores de *parachains*, apesar de que a arquitetura da Polkadot e o Substrate facilitam este trabalho. Como dito na Seção 3.4, *parachains* precisam implementar o algoritmo de finalidade GRANDPA, e com uma *bridge* isto não é diferente (não importando o algoritmo para produção de blocos). *Parachains* baseadas no Substrate podem implementar o GRANDPA com o auxílio de um *pallet*[17]. Uma *bridge* possui diversos componentes, dos quais os principais são *header sync* e *message delivery:*

- *header sync:* cliente enxuto da rede fonte executando na rede destino, adicionado como um *pallet*. Serve para aplicações alto nível da rede destino visualizarem dados da rede chamadora;
- *message delivery*: um *pallet* construído a partir do pallet *header sync*. Responsável por aceitar mensagens de usuários na rede fonte, que por sua vez devem ser entregues à cadeia destino.

[17] <https://substrate.dev/docs/en/knowledgebase/runtime/frame#grandpa>, acessado em 10/08/2021.

Figura 5 – *Bridge* em forma de *parachain* que conecta a Polkadot a uma blockchain externa.



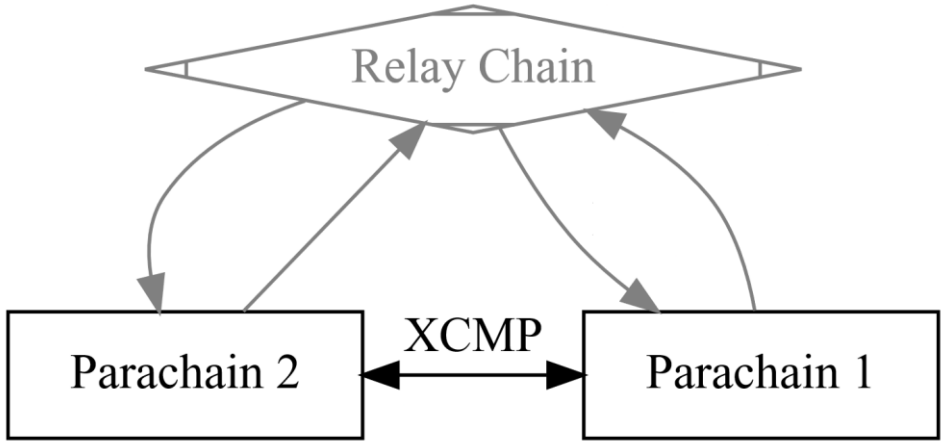
Fonte: adaptada de “Polkadot – How does Polkadot work?”.

A troca de mensagens com o pallet de *message delivery* se dá por canais que garantem ordenamento. Cada canal é isolado e pode ser processado independentemente. Diferentes canais permitem diferentes configurações de validação de mensagens e podem estar associados com aplicações específicas. A grosso modo, pode-se comparar este mecanismo à comunicação por filas de mensagens que, diferentemente da comunicação direta entre serviços web, permite garantia de entrega, melhor tolerância a falhas, comunicação assíncrona, entre outras coisas. Exemplos de serviços de filas de mensagens da Web são: RabbitMQ, AWS SQS, IBM MQ. Enquanto alguns dos elementos de tolerância a falhas e resiliência da comunicação por filas não é estranho à blockchain, em relação a garantia de entrega, por exemplo, se uma aplicação Holochain sendo chamada não está disponível por alguma razão, a chamadora precisa lidar com a falha. Além disso, com relação à interoperabilidade de protocolos de mensageria para

aplicações web, na Polkadot há uma padronização. Uma aplicação web tradicional deve conhecer o protocolo do serviço de filas de mensagens da aplicação destino, mas na Polkadot o protocolo é único e implementado através de um *pallet*, que, lembrando, pode ser incluído no *runtime* de qualquer *parachain* baseada no framework Substrate. Quanto à interoperabilidade "rede-rede" a nível de *smart-contracts*, até o momento da escrita deste trabalho não há esta possibilidade ("How to call other smart contracts on another parachain? - ink! documentation") (inclusive entre *parachains*). Um exemplo de *bridge* sendo construída para a Polkadot (com a Ethereum) é a Snowflake[18], com a qual supostamente será possível fazer chamadas entre um *smart-contract* hospedado em *parachain* e outro na Ethereum.

A interoperabilidade entre *parachains* se dá com troca de mensagens por meio da *relay chain*, sob o protocolo XCMP, como mencionado na Seção 3.4. *Parachains* fonte e destino precisam previamente concordar com sua ligação, e então é aberto um canal unidirecional (como com *bridges*), através do qual ocorre a comunicação. Cada *parachain* possui filas de entrada e saída, populadas e consumidas pelos *collators* (responsáveis pelo roteamento). Um *collator* consome uma mensagem da fila de entrada após enviar um bloco para um validador da *relay chain*. Há um limite de tamanho para as mensagens. Um par de *parachains* só pode ter dois canais (um para recebimento e outro para envio de mensagens).

Figura 6 – Comunicação entre *parachains* com XCMP.



Fonte: adaptada de "The Polkadot Parachain Host Implementer's Guide – Messaging Overview".

Como previamente explanado, cada aplicação em Holochain tem sua própria rede *P2P*. Sendo assim, elas naturalmente são flexíveis quanto a controle de acesso, regras de validação,

[18] <https://snowbridge-docs.snowfork.com/>, acessado em 10/08/2021.

parâmetros de rede como número de cópias para resiliência. Com a DLT sendo projetada desta forma, não há limitações intrínsecas para interoperabilidade entre redes ou entre aplicações. Isto significa que a Holochain permite alto nível de customização e as aplicações não perdem em nada em relação à interoperabilidade entre si, ou seja, a forma de operação entre elas é idêntica à como explicado na Seção 4.2, já que a interface entre elas continua sendo o *conductor* e nada mais. Lembrando, o design é centrado no usuário. Se o usuário é agente dos dois DNAs a interoperarem, então a chamada é uma "chamada *bridge*". Se o usuário é agente de apenas um dos DNAs, então a chamada é uma "chamada remota", para um agente que seja cliente do outra DNA. Notadamente, este último caso implica em um ponto único de falha. Por um lado, isto não é muito distinto da comunicação cliente-servidor ou entre serviços web. Por outro, os pontos de comunicação tratam-se de dispositivos de usuário, os quais são espera-se que estejam disponíveis a todo momento. Para lidar com isto, uma aplicação pode possuir mais de um endereço para "chamada remota" ou ainda prever que haja agentes específicos, os quais espera-se que sua disponibilidade seja maior que a de agentes comuns.

4.5 RANQUEAMENTO

Nesta seção, apresenta-se uma síntese da análise e comparação feitas no capítulo. O conteúdo de cada uma das seções anteriores culmina nos quadros Quadro 2 e Quadro 3, onde as quatro plataformas são ranqueadas em ordem decrescente de grau de interoperabilidade. O ordenamento ocorreu de modo qualitativo, refletindo a opinião do autor quanto aos diferentes graus de interoperabilidade.

O primeiro ranqueamento refere-se à interoperabilidade das DLTs com a web. O segundo atém-se à interoperabilidade exclusiva ao universo de DLTs. Eles foram derivados da combinação dois a dois dos quatro aspectos estudados: "externa-nativa" e "nativa-externa" para o Quadro 2 e "rede-rede" e "nativa-nativa" para o Quadro 3. A razão para isto é que os dois primeiros envolvem justamente interoperabilidade com a web, enquanto os dois últimos referem-se somente à DLTs.

Do ponto de vista da evolução da web e das justificativas para a descentralização, o desenvolvimento de aplicações em DLT (que utilizam aplicações legadas caso necessário) é mais favorável do que aplicações centralizadas que fazem uso de aplicações em DLT. Logo, no

que tange o Ranqueamento 1, o fato da DFinity não prever interoperabilidade com a web a faz ficar em último lugar.

Ao ler o Quadro 2, o motivo para a Holochain estar acima da Ethereum pode não estar claro. Deve-se ter em mente que aqui ferramentas externas e não integradas à DLT foram de menor importância para a classificação, pois isso envolve popularidade e não reflete características da plataforma. Por conseguinte, o mecanismo de emissão de eventos remotos próprio da Holochain a deixa acima da Ethereum, a despeito desta possuir ferramentas. Nota-se que os eventos da Holochain são, inclusive, uma vantagem para o Ranqueamento 2, já que podem ser utilizados para comunicação entre aplicações, ao contrário dos *events* da Ethereum.

Quadro 2 – Ranqueamento 1: por grau de interoperabilidade entre DLTs e a web.

1	Polkadot	Introduz o conceito de o*ff-chain worker*, aplicação integrada a um nó que permite processamento *off-chain* (e.g. requisições HTTP, criptografia); suporta emissão de eventos para aplicação externa local (mal documentado); há clientes em diferentes linguagens.
2	Holochain	Suporta emissão de eventos a aplicação externa local, mas também, indiretamente, a aplicações externas remotas, através de "eventos remotos".
3	Ethereum	Suporta emissão de eventos a aplicação externa local; ferramentas do ecossistema facilitam interoperabilidade: clientes em várias linguagens, diferentes opções de oráculo.
4	DFinity	Possui a IDL Candid, que suporta a representação de funções com parâmetros opcionais, e funções como parâmetro; hospeda front-ends; nenhuma informação oficial disponível em sua documentação relacionada a interoperabilidade com a web.

Fonte: elaborada pelo autor (2021)

Quadro 3 – Ranqueamento 2: por grau de interoperabilidade exclusiva ao universo de DLTs.

1	Holochain	Cada aplicação nativa tem sua rede distinta, a qual é nativamente interoperável com qualquer outra rede Holochain, ou seja, as aplicações são flexíveis e suas redes não são isoladas; aplicações podem se comunicar assíncrona ou sincronamente; não há *bridges* com blockchains.
2	Polkadot	Possibilita a comunicação entre *parachains* por meio de canais unidirecionais; facilita a construção de *bridges* com blockchains externas (e.g. Bitcoin, Ethereum).

3	DFinity	A IDL Candid suporta interfaces mais elaboradas, com parâmetros opcionais e funções como parâmetro. *Canisters* escritos em diferentes linguagens (e.g. Motoko, Rust e C) têm total interoperabilidade.
4	Ethereum	*Smart-contracts* possuem boa interoperabilidade entre si, já que todos são em Solidity e executam na mesma máquina. Há *bridges* com outras blockchains, construídas por terceiros.

Fonte: elaborada pelo autor (2021)

5 DISCUSSÃO

Este capítulo articula sobre a perspectiva de DLTs como alternativa à web centralizada, bem como faz considerações relacionando características das plataformas estudadas não só com interoperabilidade, mas também com escalabilidade, por esta última ser um tema importante (pela dimensão de aplicações web *cloud-hosted*) e de intensa pesquisa recentemente. Finalmente, são descritas limitações deste trabalho.

A interoperabilidade entre aplicações em uma DLT é maior do que entre aplicações web, considerando que em DLTs (públicas) todas as aplicações são, a princípio, abertas, bem como seus códigos. Além disso, para quaisquer duas aplicações em DLT, a maneira de interagir é idêntica, diferentemente de como funciona em aplicações web usuais, onde há diferentes formas de autenticação, APIs que não seguem a especificação correta do protocolo (e.g. métodos HTTP), sem contar a revogação de APIs, código proprietário fechado, indisponibilidade, entre outros motivos. Como visto neste trabalho, *smart-contracts*, *canister* e *zomes* (e DNAs) podem interagir entre si livremente, e suas interfaces são públicas. Pode-se tanto usar o código de uma aplicação existente e replicá-lo, implantando-o em DLT como componente de uma nova aplicação, bem como utilizar a aplicação já implantada. Isto torna possível uma web muito mais aberta e colaborativa, onde aplicações podem mais facilmente reusar e compartilhar funcionalidades e dados.

Ademais, em comparação a plataformas de computação em nuvem, onde são hospedadas as aplicações de grande escala da web hoje, DLTs poderão diminuir mais ainda o tempo de desenvolvimento de produtos, com um efeito análogo ao que a computação em nuvem trouxe nos últimos 10 ou 15 anos, pois os desenvolvedores focam na lógica de negócio, sem empregarem esforços em *firewalls*, configuração e gerenciamento de persistência, balanceamento de carga, resiliência, tolerância a falhas, descoberta de serviços etc. Plataformas de computação em nuvem facilitam essas tarefas, mas em DLTs elas não são necessárias.

Em contraponto, aplicações centralizadas possuem uma flexibilidade maior, por exemplo, em termos de banco de dados, linguagem de programação, interface web, e são escaláveis. Considerando estas vantagens, sua interoperabilidade não é ruim. Sendo assim, dadas as vantagens em relação aos problemas de centralização vistos hoje, e tendo em vista as características de DLTs citadas acima, se estas oferecessem certo grau de flexibilidade e interoperabilidade, sua adoção seria facilitada, para que aplicações distribuídas compusessem uma maior parte da web.

A Ethereum foi a primeira DLT a suportar aplicações variadas, através de *smart-contracts*, servindo de modelo para todas as plataformas posteriores baseadas também em *smart-contracts*. Por sua rede ser única e homogênea, há um alto grau de interoperabilidade: qualquer aplicação está disponível para qualquer outra, com as interações ocorrendo na mesma máquina. Contudo, a grosso modo, todos os computadores possuem todas as aplicações existentes, replicam todos os dados e repetem as mesmas operações, apresentando desafios a escalabilidade. A abordagem da Polkadot é a de interconexão de blockchains especializadas (*parachains*), que introduz complexidade na interoperabilidade, mas permite maior flexibilidade e escalabilidade, pois cada rede tem suas especificidades e os nós não hospedam todas as aplicações. A DFinity usa outro tratamento, funcionando como uma nuvem, mas decentralizada, segura e transparente, já que as máquinas são nós de uma blockchain, e providas por *data centers* independentes. Ou seja, seu funcionamento é, supostamente, tal que sendo executado por máquinas especializadas, rápido o suficiente para suportar aplicações variadas e que escalam. Não é heterogênea como a Polkadot, e sim homogênea como a Ethereum, mas com particionamento de aplicações em *subnets*. Em outras palavras, o particionamento da Polkadot é em redes distintas, e o da DFinity é em sub-redes. Sendo assim, a interoperabilidade não necessita de *bridges*, mas há certa complexidade para garantir a comunicação entre *subnets*. O design da Holochain é totalmente diferente, supostamente escalável pois cada nó de usuário (e.g. notebook, smartphone) possui apenas as aplicações do próprio usuário e alguns dados de seus vizinhos na DHT; é interoperável pois a interação entre duas aplicações se dá, geralmente, por comunicação pelo próprio nó do usuário. Pela perspectiva *agent-centric*, "duas aplicações interoperando" significa "duas instâncias de aplicações do usuário interoperando".

É possível perceber uma relação entre escalabilidade e interoperabilidade. Dito isto, pontua-se que a escalabilidade é um fator crítico para muitas aplicações web que não foi abordado neste trabalho. As considerações do parágrafo precedente são gerais, com base no que pôde ser observado da arquitetura das plataformas. Há também o fator de incentivos econômicos relacionado ao funcionamento de DLTs, que é fundamental para sua sustentabilidade e diretamente relacionado ao custo para a hospedagem e operação de aplicações. Escalabilidade, fatores econômicos, custo, usabilidade e privacidade são todas questões que possuem suas especificidades e devem ser tratadas judiciosamente. Estes assuntos

devem ser considerados ao definir o quanto uma DLT serve como alternativa a uma plataforma de computação em nuvem.

6 CONSIDERAÇÕES FINAIS & TRABALHOS FUTUROS

Os objetivos foram inteiramente contemplados, visto que foram determinados quatro aspectos de interoperabilidade que permitiram uma análise que explorou diferentes características das plataformas selecionadas. Ainda, foram providos dois ranqueamentos, os quais sintetizam a análise e a comparação. Foram identificadas funcionalidades que se destacaram, tanto para o desenvolvimento de aplicações inteiramente distribuídas, como para a comunicação com aplicações web tradicionais: requisições HTTP servidas por aplicações nativas a DFinity, bem como seu suporte a front-ends; *off-chain workers* da Polkadot e a característica modular das *parachains*, que facilitam a interoperabilidade entre blockchains especializadas. Foi visto também que o paradigma da Holochain centrado no agente favorece muito a interoperabilidade em DLT, permitindo aplicações menos isoladas, que mesmo com características variadas ainda podem interoperar, sem a necessidade de implementações complexas como *bridges* blockchain. Adicionalmente, foi visto que seu modelo é leve, impondo menos barreiras para que aplicações sejam executadas diretamente a partir dos dispositivos dos usuários. Este modelo é compatível com soluções que envolvam privacidade.

Haja vista a constatação de uma carência de detalhes nas documentações das DLTs, em relação a algumas funcionalidades, este trabalho contribui para sua compreensão, posto que as plataformas foram exploradas comparativamente, também com paralelos ao desenvolvimento de aplicações web tradicionais.

A tecnologia blockchain continua evoluindo, de forma a melhor adotar aplicações cuja segurança é uma das maiores preocupações, e cuja comunicação com outras aplicações é menos frequente (sendo *bridges*, como as da Polkadot, uma boa opção). Porém, blockchains ainda são estritas demais para suportarem uma vasta gama de aplicações web. Neste aspecto, o modelo da Holochain mostra-se promissor para a hospedagem de aplicações variadas da web atual. De qualquer modo, a tecnologia de *distributed ledgers* está demonstrando-se promissora como alternativa a plataformas de computação em nuvem, dado que, como discutido, plataformas DLT oferecem diversas vantagens ao desenvolver aplicações, podem oferecer alto grau de interoperabilidade entre aplicações distribuídas, e têm apresentado soluções para maior flexibilidade e escalabilidade.

Como trabalhos futuros, sugere-se que sejam abordados os custos da hospedagem e execução de aplicações. O custo é um fator determinante para a viabilidade de um negócio, e plataformas de computação em nuvem geralmente possuem baixo custo. Pode-se também explorar questões de armazenamento de dados. Aplicações web modernas podem utilizar quantidades massivas de dados, e o armazenamento é atrelado ao custo, bem como à escalabilidade. Em DLTs, ele também se correlaciona com interoperabilidade e nível de centralização, já que não é rara a utilização de serviços externos para armazenamento, os quais podem ser centralizados. Além disso, há a preocupação com a privacidade em relação a estes dados. Tais questões podem ser estudadas (inclusive de forma prática, com implementações de provas de conceito em diferentes DLTs, por exemplo). Ainda, trabalhos similares a este poderiam analisar o grau de suporte de cada DLT em relação a atualizações de software (adição de funcionalidades, correção de bugs, melhoramentos), que são fundamentais para aplicações web.

REFERÊNCIAS

Adding the RPC API extension - Add a Pallet - Substrate Developer Hub. Disponível em: <https://substrate.dev/docs/en/tutorials/add-contracts-pallet#adding-the-rpc-api-extension>. Acesso em: 27 ago. 2021.

AKHTAR, Z. From Blockchain to Hashgraph: Distributed Ledger Technologies in the Wild. **Proceedings - 2019 International Conference on Electrical, Electronics and Computer Engineering, UPCON 2019**, 1 nov. 2019.

ALABDULWAHHAB, F. A. Web 3.0: The Decentralized Web Blockchain networks and Protocol Innovation. **1st International Conference on Computer Applications and Information Security, ICCAIS 2018**, 20 ago. 2018.

BELCHIOR, R. et al. A Survey on Blockchain Interoperability: Past, Present, and Future Trends. 28 maio 2021.

BENAHMED, S. et al. A Comparative Analysis of Distributed Ledger Technologies for Smart Contract Development. **IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC**, v. 2019- September, 1 set. 2019.

Bridges - Polkadot Wiki. Disponível em: <https://wiki.polkadot.network/docs/learn-bridges>. Acesso em: 27 ago. 2021.

Candid Library for the Internet Computer. DFinity, , 2020. Disponível em: <https://github.com/dfinity/candid>. Acesso em: 26 ago. 2021

Canisters and code - Internet Computer. Disponível em: <https://sdk.dfinity.org/docs/developers-guide/concepts/canisters-code.html>. Acesso em: 26 ago. 2021.

CHOWDHURY, M. J. M. et al. A comparative analysis of distributed ledger technology platforms. **IEEE Access**, v. 7, p. 167930–167943, 2019.

Contracts — Solidity 0.8.6 documentation. Disponível em: <https://docs.soliditylang.org/en/v0.8.6/contracts.html?highlight=event#events>. Acesso em: 27 ago. 2021.

ELLIS, S.; JUELS, A.; NAZAROV, S. ChainLink: A Decentralized Oracle Network. **whitepaper**, 2017.

GELLERSEN, H. W.; GAEDKE, M. Object-oriented Web application development. **IEEE Internet Computing**, v. 3, n. 1, p. 60–68, jan. 1999.

HANKE, T.; MOVAHEDI, M.; WILLIAMS, D. DFINITY Technology Overview Series, Consensus System. **ACM Reference format**, v. 16, 11 maio 2018.

HARRIS-BRAUN, E.; LUCK, N.; BROCK, A. Holochain: scalable agent-centric distributed computing. **whitepaper**, 2018.

HEWITT, C.; BISHOP, P.; STEIGER, R. A universal modular ACTOR formalism for artificial intelligence. **Proceedings of the 3rd international joint conference on Artificial Intelligence**, p. 235–245, 1973.

HOHPE, G.; WOOLF, B. **Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions**. 1. ed. [s.l.] Addison-Wesley Professional; 1st edition (October 10, 2003), 2003.

How to call other smart contracts on another parachain? - ink! documentation. Disponível em: <https://paritytech.github.io/ink-docs/faq/#how-to-call-other-smart-contracts-on-another-parachain>. Acesso em: 26 ago. 2021.

Introducing a new approach to handling HTTP requests and serving assets - Internet Computer. Disponível em: <https://sdk.dfinity.org/docs/http-middleware.html#_enabling_canisters_to_respond_to_http_requests>. Acesso em: 26 ago. 2021.

JOSHI, A. P.; HAN, M.; WANG, Y. A survey on security and privacy issues of blockchain technology. **Mathematical Foundations of Computing. 2018, Volume 1, Pages 121-147**, v. 1, n. 2, p. 121, 3 maio 2018.

KANNENGIESSER, N. et al. Bridges Between Islands: Cross-Chain Technology for Distributed Ledger Technology. **53rd Hawaii International Conference on System Sciences**, p. 5298–5307, 2020.

KIAYIAS, A. et al. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 10401 LNCS, p. 357–388, 2017.

KOŠT'ÁL, K. Multi-Chain Architecture for Blockchain Networks. **Information Sciences and Technologies Bulletin of the ACM Slovakia**, v. 12, p. 8–14, 2020.

Language quick reference - Actor references - Internet Computer. Disponível em: <https://sdk.dfinity.org/docs/language-guide/language-manual.html#exp-actor>. Acesso em: 26 ago. 2021.

Nodes as a service - ethereum.org. Disponível em: <https://ethereum.org/en/developers/docs/nodes-and-clients/nodes-as-a-service/#Introduction>. Acesso em: 26 ago. 2021.

Off-Chain Features - Substrate Developer Hub. Disponível em: <https://substrate.dev/docs/en/knowledgebase/learn-substrate/off-chain-features#off-chain-workers>. Acesso em: 27 ago. 2021.

Providers - polkadot.js. Disponível em: <https://polkadot.js.org/docs/api/start/create/#providers>. Acesso em: 26 ago. 2021.

STEWART, A. Poster: GRANDPA Finality Gadget. **Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security**, 2019.

The Internet Computer Interface Specification - HTTPS Interface - Internet Computer. Disponível em: <https://sdk.dfinity.org/docs/interface-spec/index.html#http-interface>. Acesso em: 26 ago. 2021.

The Internet Computer Interface Specification - Internet Computer. Disponível em: <https://sdk.dfinity.org/docs/interface-spec/index.html>. Acesso em: 26 ago. 2021.

The OAuth 2.0 Authorization Framework. Disponível em: <https://datatracker.ietf.org/doc/html/rfc6749>. Acesso em: 27 ago. 2021.

WOOD, G. POLKADOT: VISION FOR A HETEROGENEOUS MULTI-CHAIN FRAMEWORK. **whitepaper**, 2016.

WUST, K.; GERVAIS, A. Do you need a blockchain? **Proceedings - 2018 Crypto Valley Conference on Blockchain Technology, CVCBT 2018**, p. 45–54, 5 nov. 2018.

ZEUCH, K.; WÖHNERT, K. H.; SKWAREK, V. Derivation of Categories for Interoperability of Blockchain-and Distributed Ledger Systems. **Gesellschaft für Informatik e.V.**, p. 153–166, 2019.

APÊNDICE B – ARTIGO DO TCC

Interoperabilidade em aplicações distribuídas
Um estudo de DLTs como infraestrutura para hospedagem de aplicações web

Ramna Sidharta de Andrade Palma[1], **Martín Augusto Gagliotti Vigil**[1]

[1]Instituto de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Santa Catarina – RS – Brazil

ramna.thors@gmail.com, martin.vigil@ufsc.br

Resumo. *A tecnologia de livro-razão distribuído (DLT) vem contribuindo para o desenvolvimento da web 3.0, permitindo soluções descentralizadas. Apesar disso, o modelo de blockchain, tipo de DLT mais difundido, apresenta sérias limitações. Diante disto, este trabalho visa responder se plataformas DLT são uma alternativa à plataformas de computação em nuvem, servindo como infraestrutura para hospedagem de aplicações web modernas. Para tanto, foi determinado o foco em interoperabilidade, dado que este é um desafio tanto em blockchains quanto na web 2.0. Sendo assim, são escolhidas quatro plataformas DLT de propósito geral e determinados quatro aspectos de interoperabilidade, sob os quais as plataformas são analisadas, comparadas e ranqueadas. Conclui-se que as DLTs proveem uma interoperabilidade ainda maior do que na web atual, com algumas vantagens sobre plataformas de computação nuvem atuais, mas que as formas de DLTs interoperarem com a web ainda são laboriosas. Além disso, constata-se que o modelo da Holochain é mais flexível do que o de blockchain, o que facilita interoperabilidade e escalabilidade.*

Abstract. *Distributed Ledger Technology (DLT) has contributed to the evolution of the "web 3.0" model, allowing for decentralized solutions. Despite this, its most disseminated model (blockchain) has serious limitations. We aim to answer whether DLT platforms are an alternative to cloud computing platforms, serving as an infrastructure for hosting modern web applications. For this, we focus on interoperability, which is a challenge both in blockchains and in the "web 2.0". Thus, we selected four general-purpose DLT platforms and determined four interoperability aspects, which were then used to analyze, compare, and rank the DLTs. We conclude that DLTs provide even greater interoperability than the current web paradigm, with some advantages over current cloud computing platforms, but the ways for DLTs to interoperate with the web are still laborious. In addition, the Holochain model is found to be more flexible than the blockchain, which facilitates interoperability and scalability.*

1. Introdução

Com infraestrutura, plataformas e aplicações sendo fornecidas como serviço, as oportunidades são mais abundantes, e o desenvolvimento tecnológico e de empresas é, de maneira em geral, mais acelerado (BERMAN et al., 2012). A hospedagem em nuvem é hoje inclusive considerada uma característica de aplicações web (chamada de cloud-hosted). Apesar do avanço proporcionado pelas plataformas de computação em nuvem,

grande parte do tráfego da internet hoje é centralizado em algumas corporações gigantescas. As aplicações são hospedadas maioritariamente em servidores centralizados, bem como os dados de seus usuários. Diante da influência de algoritmos de redes sociais em eleições de países, diversos escândalos de vazamento de dados e ataques cibernéticos, cresce a necessidade de regulamentações e a preocupação com privacidade e segurança de dados 16 (AUXIER et al., 2019) (NOORDYKE, 2019) (HERN, 2018) (WHITNEY, 2021). Segurança, resiliência e interoperabilidade são exemplos de desafios da web atual e de aplicações web.

A tecnologia de livro-razão distribuído (DLT) vem contribuindo para o desenvolvimento da web 3.0, permitindo soluções descentralizadas. O modelo de blockchain é bastante difundido, mas apresenta sérias limitações, em relação à, por exemplo, privacidade, interoperabilidade e escalabilidade. Diante disto, este trabalho visa responder se plataformas DLT são uma alternativa à plataformas de computação em nuvem, servindo como infraestrutura para hospedagem de aplicações web modernas. Para tanto, foi determinado o foco em interoperabilidade, dado que este é um desafio tanto em blockchains quanto na web 2.0, onde os dados são ilhados por barreiras técnicas e organizacionais entre aplicações. Sendo assim, foram escolhidas as DLTs Ethereum, Polkadot, DFinity e Holochain, a partir das listagens geral e por "plataformas de computação distribuída" do site CoinMarketCap. A seleção levou em conta plataformas de arquiteturas diferentes, e de propósito geral. Estas DLTs foram então analisadas comparativamente quanto a quatro aspectos de interoperabilidade.

2. Trabalhos relacionados

Há alguns trabalhos comparativos de DLTs. "A Comparative Analysis of Distributed Ledger Technology Platforms" faz uma comparação com critérios tanto qualitativos como quantitativos, mas sem foco em interoperabilidade e sem foco no desenvolvimento de aplicações web (por exemplo, são incluídas algumas DLTs que não suportam smart-contracts) (CHOWDHURY et al., 2019). "A Comparative Analysis of Distributed Ledger Technologies for Smart Contract Development" faz também um trabalho comparativo, considera smart-contracts, mas não aborda interoperabilidade (BENAHMED et al., 2019). "A Derivation of Categories for Interoperability of Blockchain and Distributed Ledger Systems" visa interoperabilidade, mas apenas entre plataformas blockchain, e de forma ampla, considerando diferentes critérios como linguagem de scripting, privacidade, modelo de consenso, sistema de recompensa ou incentivo (ZEUCH et al., 2019). "From Blockchain to Hashgraph: Distributed Ledger Technologies in the Wild" explora DLTs de diferentes formatos (não apenas blockchain) em uma análise comparativa, mas não foca em interoperabilidade e no desenvolvimento de aplicações web, e por abordar vários critérios, o resultado da comparação é mais geral (como "sim" ou "não" para escalabilidade) (AKHTAR, 2019).

É relevante notar que há uma categoria mais específica de trabalhos, que explora interoperabilidade em detalhes (BELCHIOR et al., 2021) (KANNENGIESSER et al., 2020) (KOŠT'ÁL, 2020). Todavia, estes são focados somente em blockchain, em mais baixo nível e envolvem detalhes avançados de protocolos. Este trabalho não é sobre blockchain, mas sim aplicações distribuídas.

O presente trabalho se diferencia dos acima apresentados ao comparar não apenas

blockchains, mas também plataformas DLTs de propósito geral, quanto a servir como infraestrutura para o desenvolvimento de aplicações web interoperáveis. Ainda, com o foco em aplicações, são traçados paralelos e comparações com o desenvolvimento na web atual.

3. Aspectos comparativos de interoperabilidade

Foram mapeados determinados quatro aspectos de interoperabilidade, que são casos onde tem-se o objetivo de fazer com que dois serviços se comuniquem, sendo ao menos um deles uma aplicação distribuída, hospedada em DLT. A seguir são discriminados estes casos de interoperabilidade, representados como "A-B", onde a aplicação "A" utiliza uma funcionalidade oferecida pela aplicação "B". Uma aplicação é chamada "nativa" quando hospedada em DLT e "externa" caso contrário, isto é, não distribuída. Por exemplo, uma aplicação hospedada em um serviço de computação em nuvem.

1. Externa-nativa: entre uma aplicação externa à uma DLT e uma aplicação nativa, sendo a interface de usuário da aplicação nativa um caso especial de aplicação externa;
2. Nativa-nativa: entre duas aplicações hospedadas em uma mesma DLT;
3. Nativa-externa: entre aplicação hospedada em DLT e aplicação externa, não hospedada em DLT;
4. Rede-rede: entre duas aplicações nativas a DLT, mas que não estejam em uma mesma rede, com as redes sendo de uma mesma DLT ou de DLTs distintas.

4. Plataformas DLT

4.1. Ethereum

A Ethereum (BUTERIN, 2014) é uma DLT, cuja principal característica é o suporte a smart-contracts. Foi lançada em 2015, sendo a primeira DLT a hospedar aplicações genéricas. Seu algoritmo de consenso é do tipo prova de trabalho (ou Proof-of-work). Solidity é a linguagem principal e original, baseada em JavaScript, embora hoje existam outras linguagens (e.g., mais simples, com menos funcionalidades ou de mais baixo nível), como Yul ou Vyper.

Assim como um nó da rede Ethereum possui todo o histórico de transações, ele possui também todos os smart-contracts hospedados na rede, e todas as suas transições de estado, que ocorrem de acordo com sua execução.

4.2. DFinity

A DFinity ou DFinity Foundation é uma organização sem fins lucrativos fundada em 2016 por Dominic Williams. Ela desenvolve o Internet Computer (IC), uma DLT. A proposta do IC é a de servir como uma plataforma única para o desenvolvimento e hospedagem de aplicações web dos mais variados tipos. Em outras palavras, uma plataforma de computação em nuvem (como Heroku, Azure ou AWS), só que descentralizada. Essencialmente, trata-se de uma plataforma de computação distribuída cujo modelo de computação é baseado em smart-contracts. A infraestrutura do IC é provida por data centers independentes, cujos computadores participam como nós de uma blockchain, a qual tem como algoritmo de consenso um tipo de prova de autoridade (HANKE; MOVAHEDI; WILLIAMS, 2018). Os nós compõem diferentes sub-redes referidas por subnets.

Cada subnet hospeda uma parte das aplicações da rede inteira. Naturalmente, todos os nós de uma subnet replicam código e estado das mesmas aplicações, bem como as executam. A partir daqui o IC será referido por DFinity, para facilitar o entendimento ao longo do texto, onde há outras siglas, e porque a tecnologia é frequentemente assim citada.

As aplicações na DFinity são chamadas de canisters, que são como smart-contracts, no sentido de que múltiplos nós da blockchain possuem e executam um mesmo canister, e as transições de seu estado requerem consenso. Canisters são também módulos WebAssembly que armazenam código e estado. Canisters seguem o modelo de computação baseado em atores (HEWITT; BISHOP; STEIGER, 1973), isto é, representam um tipo de entidade que encapsula dados e comportamento, podendo: (i) comunicar-se com outros canisters através de mensagens; (ii) modificar seu próprio estado em resposta a uma mensagem; (iii) criar outros atores.

4.3. Holochain

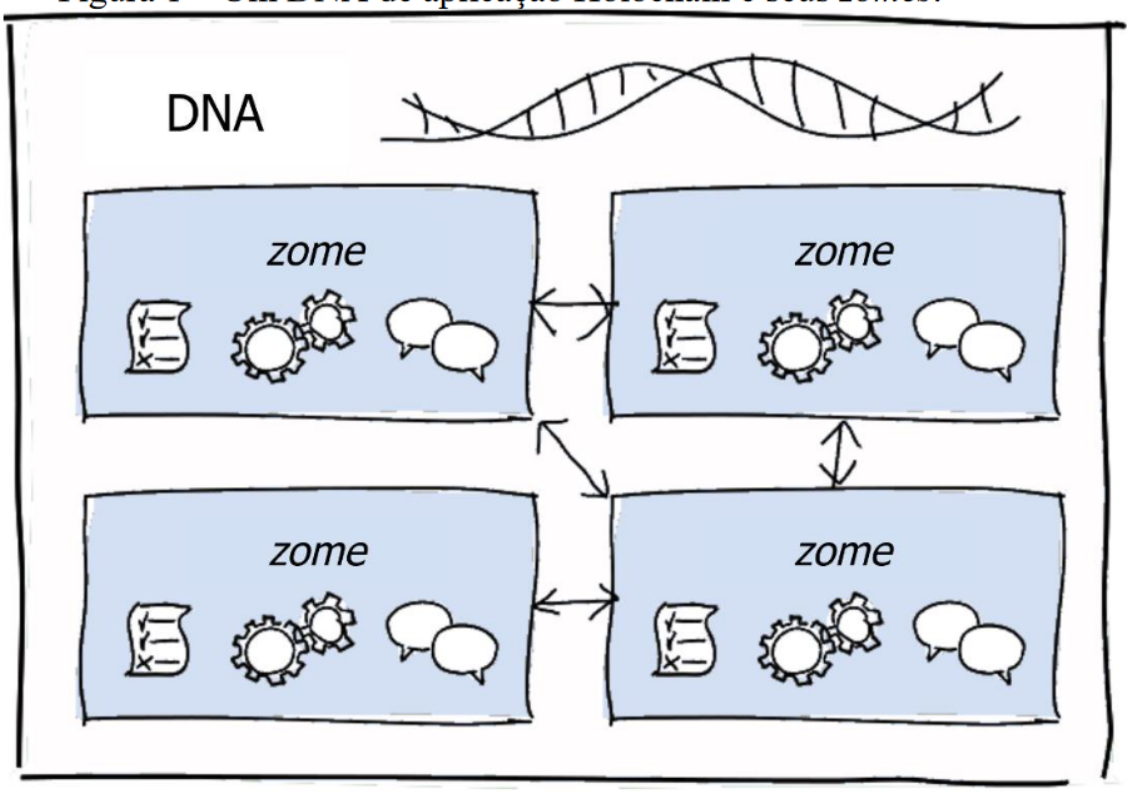
A Holochain (HARRIS-BRAUN; LUCK; BROCK, 2018) é um framework para o desenvolvimento de aplicações P2P (Peer-to-Peer), com garantias de integridade e autenticidade dos dados. A rede P2P de uma aplicação, que conecta usuários, é baseada em uma DHT (Distributed Hash Table). É através da DHT que usuários são identificados e endereçados. Dados de uma aplicação também são identificados, bem como compartilhados, validados e replicados, através da DHT. Uma aplicação Holochain é chamada de hApp.

Esta plataforma é agent-centric, o que significa que toda ação na rede é disparada por um agente, e a interpretação dos dados é relativa ao seu ponto de vista. Cada usuário executa uma aplicação em seu próprio dispositivo, e seu agente cria e armazena apenas os próprios dados, interagindo diretamente (ponto-a-ponto) com outros agentes. Isto será esclarecido mais adiante. A Holochain baseia-se também em biomimética. O código de uma aplicação chama-se DNA, ilustrado na Figura 1. O DNA define tipos de dados, regras de validação para estes dados e a lógica da aplicação em si, e é composto por módulos, chamados de zomes (ou "cromossomos"). Zomes definem funções que farão parte da API pública de um hApp. A combinação de um DNA e do agente que o executa é referida por célula (cell). Todos os nós que executam o meso DNA conectam-se a uma mesma DHT, através da qual dados são compartilhados e validados.

4.4. Polkadot

A Polkadot (WOOD, 2016) é uma rede heterogênea de múltiplas blockchains interoperáveis. É composta por dois conceitos centrais: relay chain e parachain. Relay chain é a blockchain central, com algoritmo de consenso do tipo proof-of-stake (KIAYIAS et al., 2017), responsável pela interconexão entre todas as blockchains do ecossistema. Uma parachain é uma blockchain especializada para casos de uso específicos que se conecta à relay chain, tendo seu próprio protocolo de validação, governança e funcionalidades. Por exemplo, uma parachain pode ser uma blockchain especializada em transações monetárias anônimas, smart contracts para DAOs (Decentralized Autonomous Organizations), oráculos, etc. A blockchain central executa alguns tipos específicos de transação, com o objetivo de coordenar todo o sistema. O funcionamento entre relay chain e parachains envolve principalmente duas entidades (ilustradas na Figura 4).

Figura 1 – Um DNA de aplicação Holochain e seus *zomes*.



Fonte: adaptado de "Holochain – Application Architecture".

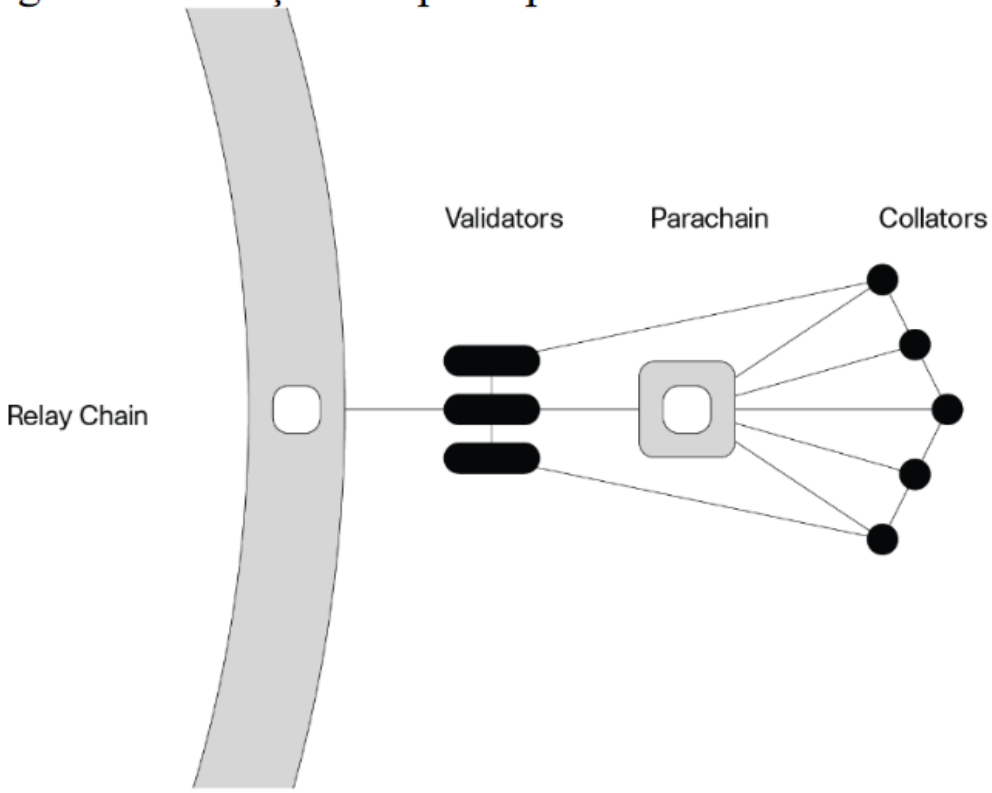
Validators (ou validadores) são nós da relay chain e os principais atores para a segurança da Polkadot, periodicamente e randomicamente atribuídos a uma parachain, para qual devem produzir, validar e ratificar blocos, os adicionando à relay chain. Como não é razoável esperar que cada validador faça isso para todas as parachains, a tarefa de produção de blocos é terceirizada para os collators. Assim, os validadores recebem provas de transições de estado dos collators; estes auxiliam os validadores na produção de blocos de uma parachain. Um collator é nó completo tanto de uma parachain como da relay chain e que coleta transações da primeira criando candidatos a blocos, dos quais são produzidas provas de transição de estado. Estes blocos, junto de suas provas, são enviadas aos validadores.

5. Análise e comparação

5.1. Externa-nativa

Para a Holochain, estas chamadas são feitas à uma instância local do nó da DLT (conductor), assim como para a Ethereum e Polkadot. Porém, para as duas últimas, há a opção de conexão à rede através de provedores ("Nodes as a service - ethereum.org") ("Providers - polkadot.js"). Provedores são serviços web tradicionais que oferecem acesso à blockchains. São hospedados em nuvem, provisionando e gerenciando nós de blockchains, de forma que desenvolvedores de aplicações externas possam facilmente conectar-se à uma DLT sem precisar empregar tempo e esforço em sua manutenção. Dependendo da proporção de nós da rede provisionados dessa forma, há uma centralização

Figura 4 – Relação dos principais elementos da Polkadot.



Fonte: documentação da Polkadot.

do acesso e operação da DLT. A conexão com aplicações nativas da DFinity é análoga àquela através de provedores. É sempre remota, pois os nós não são executados por qualquer pessoa ou entidade, e sim apenas por provedores (“data centers independentes”) aprovados, que executam em hardware específico. A aplicação externa envia requisições HTTP que chegarão a um nó aleatório que serve a aplicação nativa desejada (um canister). Em teoria, as subnets possuem endereços conhecidos, podendo responder quais os endereços IP dos nós de um canister específico.

A DFinity é a única que não suporta WebSockets, mas seu suporte a HTTP é ligeiramente superior às demais, pelo fato específico de aplicações nativas a ela poderem definir interfaces HTTP (“Introducing a new approach to handling HTTP requests and serving assets - Internet Computer”) que suportam a execução de funções do tipo query. A DFinity também suporta esta forma. Idealmente, o suporte por aplicações distribuídas a interfaces HTTP pode, além de oferecer as vantagens citadas anteriormente, facilitar a migração de um sistema em cloud para uma solução distribuída, já que sua API poderia se manter igual ou parecida, reduzindo o trabalho necessário para adaptar os códigos dos clientes. Canisters não suportam este grau de interoperabilidade pois, como já mencionado, ainda não suportam diferentes métodos HTTP, embora a discussão já tenha sido levantada, e algumas provas de conceito propostas. Ao invés de páginas Web serem obtidas de servidores de cloud, elas são disponibilizadas por um canister. Isto é possível

através das requisições que são respondidas diretamente por canisters.

Todas as DLTs disponibilizam bibliotecas que auxiliam na comunicação com serviços nativos. Aquelas para Ethereum, Polkadot e Holochain possuem abstrações em nível semelhante enquanto que para a DFinity há um destaque, por possuir implementações de uma IDL (Interface Description Language). Com o Candid, serviços na DFinity podem ser escritos em diferentes linguagens. O propósito principal do Candid é descrever interfaces de serviços de forma independente de linguagem. É possível descrever funções, parâmetros de entrada e saída, inclusive parâmetros opcionais, que podem ser adicionados sem a quebra de compatibilidade, tipos especiais (como structs). Além de dados, especificações nesta IDL também suportam a descrição de referências a outros serviços e funções como parâmetro e retorno.

5.2. Nativa-nativa

A interação entre aplicações na Ethereum ou na Polkadot é muito parecida, seguindo os mesmos padrões de desenvolvimento de smart-contracts (referidos aqui por "contratos"). Cada contrato tem um conjunto de funções públicas e dados, que compõem sua interface. Um contrato pode interagir com outro sabendo seu endereço e interface (incluindo dados de entrada e saída). Caso a interface não seja conhecida, é possível fazer chamadas especificando a função alvo por seu nome. No contrato chamador, é possível declarar a interface do contrato alvo e então instanciá-la, através de seu endereço. Também é possível instanciar e hospedar um smart-contract inteiramente novo, a partir de outro, com um novo endereço e seu estado inicial. Através da instância fazem-se chamadas síncronas e determinísticas, que são executadas localmente no mesmo nó que processa o smart-contract chamador.

Dado que o modelo de programação da DFinity é baseado em atores, um canister pode ser visto como um objeto especial, com memória isolada, comunicando-se por troca de mensagens assíncronas, passando dados binários de valores Candid codificados. O Candid é responsável por prover compatibilidade total de dados aos canisters envolvidos em uma comunicação, mesmo que eles sejam implementados em linguagens diferentes. Uma única mensagem é processada por vez, eliminando condições de corrida. Cada emissão de mensagem é associada a uma função de callback que processa o resultado da chamada, como naturalmente funcionaria um cliente não bloqueante de web service. Todavia, a comunicação entre micro serviços através de troca de mensagens, por exemplo, é mais complexa que isso, já que os resultados são totalmente independentes dos envios. Ademais, há uma dificuldade maior em garantir o formato dados.

Na Holochain existem quatro tipos de chamadas entre aplicações. Todos são chamadas de procedimento remoto (RPC): "chamada bridge" e "chamada remota". Uma "chamada bridge" é efetuada através da função hdk::call. Dado que esta é uma chamada entre aplicações, vale ressaltar que ela ocorre em um mesmo nó, e que as source chains de cada aplicação são travadas durante a comunicação, facilitando a lógica da aplicação chamadora no que tange o nível de confiança acerca dos dados que lhe estão disponíveis. Ainda, é notável a facilidade e simplicidade arquitetural do compartilhamento de dados de um mesmo usuário entre aplicações. Como na Holochain ambas as aplicações executam localmente e em nome do usuário, não há barreiras para a troca de dados. Em aplicações web tradicionais onde há barreiras organizacionais, o análogo necessitaria

da implementação de um protocolo como o OAuth 2.0 (“The OAuth 2.0 Authorization Framework”, 2012). A ”chamada remota” ocorrem quando a instância da aplicação de um agente executa uma função da mesma aplicação em outro agente. Essa funcionalidade oferece algumas vantagens: troca de dados privados à source chain da aplicação sendo chamada; compartilhamento de dados de aplicações terceiras, disponíveis no nó da aplicação sendo chamada, mas não no nó da aplicação chamadora; troca de informações ponta a ponta, sem registros públicos.

5.3. Nativa-externa

Para que operações externas sejam disparadas a partir de uma aplicação na Ethereum faz-se uso do mecanismo de events (“Contracts — Solidity 0.8.6 documentation”). Um event, na linguagem Solidity, é um conjunto de dados que pode ser emitido a partir de uma função de smart-contract, indicando a uma aplicação externa a ocorrência de uma operação. Isto pode disparar uma ação implementada por uma aplicação externa intermediária, interessada no evento, que por sua vez executa a ação na aplicação externa alvo (ou qualquer outra operação off-chain). Este padrão é similar ao padrão “barramento de eventos” ou “barramento de mensagens” (HOHPE; WOOLF, p. 137, 2003), comum na integração de aplicações web, que permite que aplicações interessadas em eventos específicos os consumam do barramento de forma assíncrona, sem acoplamento entre produtor e consumidor.

A Polkadot facilita a construção e integração de oráculos com o que é chamado de off- chain worker (OCW) (“Off-Chain Features - Substrate Developer Hub”). Trata-se de uma aplicação que executa junto a um nó de parachain, mas cujos dados não são passíveis de consenso. Um OCW executa em um ambiente WASM (Web-Assembly) isolado e é disparado a partir de uma chamada de função15 do nó propriamente dito. Off-chain workers podem gerar números aleatórios, executar requisições a aplicações externas, executar operações demoradas demais para smart-contracts ou caras demais se executadas por smart-contracts (e.g. criptografar ou descriptografar). Ainda, podem submeter transações assinadas ou não assinadas de volta ao runtime do nó.

A DFinity não faz menção a esse tipo de interoperabilidade. O mecanismo que permite aplicações em Holochain dispararem ações em aplicações externas é o mesmo que permite a comunicação assíncrona entre agentes, explicado na seção anterior. Seu funcionamento é bastante parecido ao da Ethereum: os dados são transmitidos em JSON; a aplicação conectada por WebSockets ao nó Holochain inscreve-se para receber eventos na porta de um DNA específico.

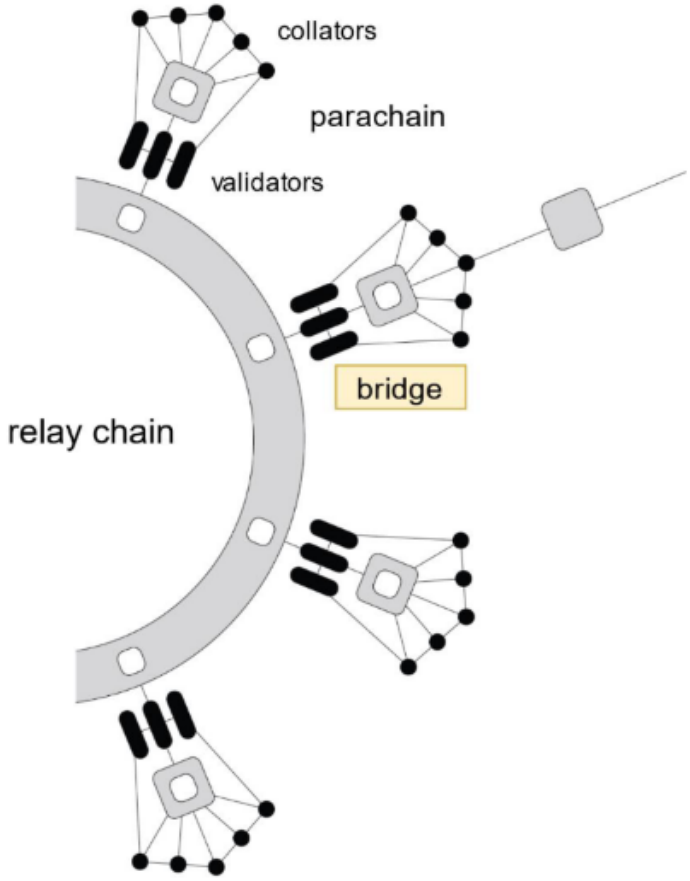
5.4. Rede-rede

Esta análise aplica-se apenas à Holochain e Polkadot, já que Ethereum e DFinity não redes heterogêneas.

A Polkadot facilita a interoperabilidade entre parachains e uma blockchain externa através de bridges (“Bridges - Polkadot Wiki”). Uma bridge é um sistema que permite a interoperabilidade entre duas blockchains. No caso da Polkadot, uma bridge é um tipo especial de parachain (a partir daqui, será referida apenas por bridge), vide Figura 5. Parachains precisam implementar o algoritmo de finalidade GRANDPA, e com uma bridge isto não é diferente (não importando o algoritmo para produção de blocos).

Parachains baseadas no Substrate podem implementar o GRANDPA com o auxílio de um pallet.

Figura 5 – *Bridge* em forma de *parachain* que conecta a Polkadot a uma blockchain externa.



Fonte: adaptada de "Polkadot – How does Polkadot work?".

Cada aplicação em Holochain tem sua própria rede P2P. Sendo assim, elas naturalmente são flexíveis quanto a controle de acesso, regras de validação, parâmetros de rede como número de cópias para resiliência. Com a DLT sendo projetada desta forma, não há limitações intrínsecas para interoperabilidade entre redes ou entre aplicações. Isto significa que a Holochain permite alto nível de customização e as aplicações não perdem em nada em relação à interoperabilidade entre si, ou seja, a forma de operação entre elas é idêntica à como explicado na Seção 4.2, já que a interface entre elas continua sendo o conductor e nada mais.

5.5. Ranqueamento

Nesta seção, apresenta-se uma síntese da análise e comparação feitas no capítulo. O conteúdo de cada uma das seções anteriores culmina nos quadros Quadro 2 e Quadro 3, onde as quatro plataformas são ranqueadas em ordem decrescente de grau de interoperabilidade. O ordenamento ocorreu de modo qualitativo, refletindo a opinião do autor quanto aos diferentes graus de interoperabilidade. O primeiro ranqueamento refere-se à

interoperabilidade das DLTs com a web. O segundo atém-se à interoperabilidade exclusiva ao universo de DLTs. Eles foram derivados da combinação dois a dois dos quatro aspectos estudados: "externa-nativa" e "nativa-externa" para o Quadro 2 e "rede-rede" e "nativa-nativa" para o Quadro 3. A razão para isto é que os dois primeiros envolvem justamente interoperabilidade com a web, enquanto os dois últimos referem-se somente à DLTs.

Quadro 2 – Ranqueamento 1: por grau de interoperabilidade entre DLTs e a web.

1	Polkadot	Introduz o conceito de *off-chain worker*, aplicação integrada a um nó que permite processamento *off-chain* (e.g. requisições HTTP, criptografia); suporta emissão de eventos para aplicação externa local (mal documentado); há clientes em diferentes linguagens.
2	Holochain	Suporta emissão de eventos a aplicação externa local, mas também, indiretamente, a aplicações externas remotas, através de "eventos remotos".
3	Ethereum	Suporta emissão de eventos a aplicação externa local; ferramentas do ecossistema facilitam interoperabilidade: clientes em várias linguagens, diferentes opções de oráculo.
4	DFinity	Possui a IDL Candid, que suporta a representação de funções com parâmetros opcionais, e funções como parâmetro; hospeda front-ends; nenhuma informação oficial disponível em sua documentação relacionada a interoperabilidade com a web.

Fonte: elaborada pelo autor (2021)

6. Considerações finais & trabalhos futuros

Os objetivos foram inteiramente contemplados, visto que foram determinados quatro aspectos de interoperabilidade que permitiram uma análise que explorou diferentes características das plataformas selecionadas. Ainda, foram providos dois ranqueamentos, os quais sintetizam a análise e a comparação. Foram identificadas funcionalidades que se destacaram, tanto para o desenvolvimento de aplicações inteiramente distribuídas, como para a comunicação com aplicações web tradicionais: requisições HTTP servidas por aplicações nativas a DFinity, bem como seu suporte a front-ends; off-chain workers da Polkadot e a característica modular das parachains, que facilitam a interoperabilidade entre blockchains especializadas. Foi visto também que o paradigma da Holochain centrado no agente favorece muito a interoperabilidade em DLT, permitindo aplicações menos isoladas, que mesmo com características variadas ainda podem interoperar, sem a necessidade de implementações complexas como bridges blockchain.

A tecnologia blockchain continua evoluindo, de forma a melhor adotar aplicações cuja segurança é uma das maiores preocupações, e cuja comunicação com outras aplicações é menos frequente (sendo bridges, como as da Polkadot, uma boa opção). Porém, blockchains ainda são estritas demais para suportarem uma vasta gama de aplicações web. Neste aspecto, o modelo da Holochain mostra-se promissor para a

Quadro 3 – Ranqueamento 2: por grau de interoperabilidade exclusiva ao universo de DLTs.

1	Holochain	Cada aplicação nativa tem sua rede distinta, a qual é nativamente interoperável com qualquer outra rede Holochain, ou seja, as aplicações são flexíveis e suas redes não são isoladas; aplicações podem se comunicar assíncrona ou sincronamente; não há *bridges* com blockchains.
2	Polkadot	Possibilita a comunicação entre *parachains* por meio de canais unidirecionais; facilita a construção de *bridges* com blockchains externas (e.g. Bitcoin, Ethereum).
3	DFinity	A IDL Candid suporta interfaces mais elaboradas, com parâmetros opcionais e funções como parâmetro. *Canisters* escritos em diferentes linguagens (e.g. Motoko, Rust e C) têm total interoperabilidade.
4	Ethereum	*Smart-contracts* possuem boa interoperabilidade entre si, já que todos são em Solidity e executam na mesma máquina. Há *bridges* com outras blockchains, construídas por terceiros.

Fonte: elaborada pelo autor (2021)

hospedagem de aplicações variadas da web atual. De qualquer modo, a tecnologia de distributed ledgers está demonstrando-se promissora como alternativa a plataformas de computação em nuvem, dado que, como discutido, plataformas DLT oferecem diversas vantagens ao desenvolver aplicações, podem oferecer alto grau de interoperabilidade entre aplicações distribuídas, e têm apresentado soluções para maior flexibilidade e escalabilidade.

7. Referências

AKHTAR, Z. From Blockchain to Hashgraph: Distributed Ledger Technologies in the Wild. Proceedings - 2019 International Conference on Electrical, Electronics and Computer Engineering, UPCON 2019, 1 nov. 2019.

ALABDULWAHHAB, F. A. Web 3.0: The Decentralized Web Blockchain networks and Protocol Innovation. 1st International Conference on Computer Applications and Information Security, ICCAIS 2018, 20 ago. 2018.

BELCHIOR, R. et al. A Survey on Blockchain Interoperability: Past, Present, and Future Trends. 28 maio 2021.

BENAHMED, S. et al. A Comparative Analysis of Distributed Ledger Technologies for Smart Contract Development. IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC, v. 2019- September, 1 set. 2019.

Candid Library for the Internet Computer. DFinity, , 2020. Disponível em: ¡https://github.com/dfinity/candid¿. Acesso em: 26 ago. 2021.

CHOWDHURY, M. J. M. et al. A comparative analysis of distributed ledger tech-

nology platforms. IEEE Access, v. 7, p. 167930–167943, 2019.

Contracts — Solidity 0.8.6 documentation. Disponível em: ¡https://docs.soliditylang.org/en/v0.8.6/contracts.html?highlight=eventevents¿. Acesso em: 27 ago. 2021.

ELLIS, S.; JUELS, A.; NAZAROV, S. ChainLink: A Decentralized Oracle Network. whitepaper, 2017.

GELLERSEN, H. W.; GAEDKE, M. Object-oriented Web application development. IEEE Internet Computing, v. 3, n. 1, p. 60–68, jan. 1999.

HANKE, T.; MOVAHEDI, M.; WILLIAMS, D. DFINITY Technology Overview Series, Consensus System. ACM Reference format, v. 16, 11 maio 2018.

HARRIS-BRAUN, E.; LUCK, N.; BROCK, A. Holochain: scalable agent-centric distributed computing. whitepaper, 2018.

HEWITT, C.; BISHOP, P.; STEIGER, R. A universal modular ACTOR formalism for artificial intelligence. Proceedings of the 3rd international joint conference on Artificial Intelligence, p. 235–245, 1973.

HOHPE, G.; WOOLF, B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. 1. ed. [s.l.] Addison-Wesley Professional; 1st edition (October 10, 2003), 2003.

Introducing a new approach to handling HTTP requests and serving assets - Internet Computer. Disponível em: ¡https://sdk.dfinity.org/docs/http-middleware.html_enabling_canisters_to_respond_to_http_requests¿. Acesso em: 26 ago. 2021.

JOSHI, A. P.; HAN, M.; WANG, Y. A survey on security and privacy issues of blockchain technology. Mathematical Foundations of Computing. 2018, Volume 1, Pages 121-147, v. 1, n. 2, p. 121, 3 maio 2018.

KANNENGIESSER, N. et al. Bridges Between Islands: Cross-Chain Technology for Distributed Ledger Technology. 53rd Hawaii International Conference on System Sciences, p. 5298–5307, 2020.

KIAYIAS, A. et al. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), v. 10401 LNCS, p. 357–388, 2017.

KOŠT'ÁL, K. Multi-Chain Architecture for Blockchain Networks. Information Sciences and Technologies Bulletin of the ACM Slovakia, v. 12, p. 8–14, 2020.

Off-Chain Features - Substrate Developer Hub. Disponível em: ¡https://substrate.dev/docs/en/knowledgebase/learn-substrate/off-chain-featuresoff-chain-workers¿. Acesso em: 27 ago. 2021.

STEWART, A. Poster: GRANDPA Finality Gadget. Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019.

The Internet Computer Interface Specification - HTTPS Interface - Internet

Computer. Disponível em: ¡https://sdk.dfinity.org/docs/interface-spec/index.html#http-interface¿. Acesso em: 26 ago. 2021.

The OAuth 2.0 Authorization Framework. Disponível em: ¡https://datatracker.ietf.org/doc/html/rfc6749¿. Acesso em: 27 ago. 2021.

WOOD, G. POLKADOT: VISION FOR A HETEROGENEOUS MULTI-CHAIN FRAMEWORK. whitepaper, 2016.

WUST, K.; GERVAIS, A. Do you need a blockchain? Proceedings - 2018 Crypto Valley Conference on Blockchain Technology, CVCBT 2018, p. 45–54, 5 nov. 2018.

ZEUCH, K.; WÖHNERT, K. H.; SKWAREK, V. Derivation of Categories for Interoperability of Blockchain-and Distributed Ledger Systems. Gesellschaft für Informatik e.V., p. 153–166, 2019.