GERENCIAMENTO DE NEGOCIAÇÃO COM ARBITRAGEM EM *BLOCKCHAIN*

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS 01/2019

Discente: Cléber Macieski[1]

Orientador: Rafael Vieira Coelho[2]

Instituto Federal de Educação Ciência e Tecnologia do Rio Grande do Sul

Campus Farroupilha

Resumo: Criptomoedas tem se mostrado confiáveis e valiosas. São inúmeras as possibilidades de uso para a tecnologia *blockchain* e provavelmente existem muitas possibilidades a serem descobertas. Este artigo apresenta a proposta de um aplicativo com a finalidade de se tornar mais uma ferramenta disponível à sociedade visando simplificar, baratear e trazer segurança a transações comerciais, utilizando criptomoedas. A tecnologia discutida é inovadora e uma grande novidade, tendo sido apresentada ao mundo no ano de 2009, seu desenvolvimento baseado em sua comunidade de usuários, sob os preceitos *open source*. Em confluência a isso, a realização deste trabalho utilizou de ferramentas livremente disponíveis na internet, explorando este conjunto de tecnologias inovadoras assim como a base das mesmas, constituídas de outras tecnologias consolidadas e confiáveis, capazes de fornecer os subsídios necessários à solução de problemas de forma inédita, surpreendente e empolgante. Neste trabalho foi criado um *app mobile* e se buscou criar uma forma de transação *on-line* onde os participantes podem utilizar de um árbitro capaz de resolver conflitos oriundos da incapacidade de estimar confiança entre pares distantes e/ou desconhecidos, utilizando criptomoedas. Foi encontrada viabilidade no principal objetivo elencado, a efetivação de transações com criptomoedas sob a proteção de um árbitro com poderes para resolver conflitos que em uma transação direta representam um grande problema.

Palavras-chave: Criptomoedas. *Blockchain*. Multi-Assinatura.

1 INTRODUÇÃO

No inicio da civilização, sistemas de troca eram a solução utilizada para realizar o comércio, porém, devido às evoluções tecnológicas, a espécie humana passou a dispor de um excedente de bens, os quais possibilitaram um aumento populacional, num ciclo que se retroalimenta (NOGUEIRA, 2018). O comércio passou a ficar cada vez mais complexo e problemas surgiram. Por exemplo, para conseguir obter bens de comerciantes não interessados em nada diretamente produzido por determinado produtor, havia a necessidade da formação de cadeias de relações comerciais que se mostravam ineficientes. Além disso, a troca por vezes se mostrava ineficaz no fato de que o valor estimado de determinado bem em relação a outro se mostrava incorreto ou até falso.

Com base neste tipo de problema delimitou-se a necessidade de padronizar um valor para as coisas, um modelo de referência que servisse de intermediário nas relações de comércio e que inspirasse confiança aos envolvidos. Dessa forma foi concebido o conceito de moeda (ENCYCLOPAEDIA BRITANNICA, 2018), também chamada de dinheiro e definida como:

[1] cleber.macieski@gmail.com

[2] rafael.coelho@ifrs.edu.br

"Meio de troca, sob a forma de moedas ou notas, usado na aquisição de bens, na compra de serviços, de mão-de-obra, ou noutras transações financeiras, emitido pelo governo de cada país". (PRIBEREAM, 2008-2013). Moedas são utilizadas dentro um sistema monetário, que segundo Abreu e Coelho (2009 apud RATTI, 2001), é o conjunto das diversas moedas que circulam em um país, guardando entre si relações definidas de valor, de acordo com normas legais estabelecidas pelas autoridades monetárias.

No início de sua utilização, a moeda foi cunhada em metais como prata e ouro, onde identificaram-se outros problemas: (a) transações de grandes quantias se tornavam difíceis; (b) armazenamento de moeda; e (c) se tornou necessário a existência de um intermediário nas transações, que fornecesse a moeda padrão, o que aumentava o custo das transações. A participação dos bancos se consolidou. Eles eram utilizados para guardar o dinheiro dos indivíduos de maneira relativamente segura, auxiliar em transações e prover serviços financeiros a juros.

Porém a utilização de metais como moeda se mostrou ineficaz, culminando na criação do papel moeda. Tal tipo de moeda não possui valor por si só como os metais anteriormente utilizados, mas é produzida de maneira a ser de difícil cópia e era inicialmente lastreada em referência a um estoque de metal precioso existente no país. Estas medidas garantiam a escassez e, consequentemente, um valor relativo. Efetivamente, era possível trocar papel moeda por metal precioso nos bancos. Este padrão causou problemas: (a) Podia não suprir a demanda de dinheiro da sociedade, devido ao aumento da disponibilidade dos metais não estar estritamente relacionado com as necessidades financeiras da economia; (b) Um país talvez não conseguisse isolar sua economia da crise ou inflação do resto do mundo; (c) O processo de ajuste para um país com um *déficit* poderia ser longo e doloroso.

Assim sendo, a maioria dos países abandonou o lastro e passou a usar um sistema financeiro no qual o governo busca definir junto a um banco central a quantidade de dinheiro disponível (ROTHBARD, 1995). Esta abordagem traz os problemas atualmente existentes no sistema financeiro: (a) Custos extras na transação: a utilização de intermediários para possibilitar transações ou outras operações financeiras exige que as partes envolvidas paguem por serviços diversos. (b) Liberdade: a necessidade de intermediários limita a liberdade do indivíduo em realizar transações. (c) Inflação: a desvalorização da moeda de um país, causada por uma grande disponibilidade de moeda, que quando descontrolada aumenta o custo dos produtos para o consumidor. (d) Deflação: a valorização da moeda, devido à baixa disponibilidade de moeda na economia, que a longo prazo em um cenário de recessão, pode gerar por exemplo, desemprego. (e) Na economia globalizada atual, problemas financeiros das nações provocam reações em cadeia, afetando diversos países. Estas limitações do sistema monetário atual causam diversos problemas, como instabilidade econômica, pobreza, disfunções sociais e guerras.

Através das técnicas de criptografia e da computação distribuída foi desenvolvida a tecnologia *blockchain*. Esta tecnologia define um protocolo de comunicação eletrônico que possibilita a implementação das chamadas criptomoedas.

Entre as particularidades necessárias à implementação efetiva de uma criptomoeda ou moeda digital, está a capacidade de lidar com problema do gasto múltiplo de uma mesma unidade de moeda. A solução de tal problema veio ser criada através da criptografia utilizada na tecnologia *blockchain*.

A tecnologia *blockchain*, intimamente ligada as criptomoedas, pode ser considerada um livro-razão (do inglês, *ledger*), na qual transações são registradas, e validadas como únicas, utilizando de criptografia e de uma rede *peer-to-peer* para tal. As criptomoedas são transacionadas através de softwares chamados de carteiras (*wallets*, em inglês), tendo esses softwares funcionalidade análoga a carteiras comuns, mas podendo implementar contratos inteligentes (*smart contracts*) com capacidades ainda pouco exploradas.

Criptomoedas impactam diretamente no sistema monetário atual: (a) A capacidade de

distribuir, entre próprios indivíduos na rede, a possibilidade de realizar o trabalho de intermédio de transações, pode se mostrar, em diversos níveis, uma opção barata e eficaz na resolução do problema. (b) Utilizando a rede mundial de computadores, pode promover a liberdade a povos antes relegados a um sistema econômico restrito. (c) A auto regulação determinada na implementação do sistema pode acabar com os problemas gerados pela ingerência de governantes, atuar na estabilização do sistema econômico mundial. (d) Devido ao sistema ser baseado em um código aberto, a segurança do mesmo pode ser verificada por qualquer um, assim como qualquer pesquisador pode buscar melhorar a implementação do sistema. (e) A tecnologia de livro-razão distribuído possui capacidades ainda pouco exploradas, para auto processar transações, na qual regras podem ser persistidas dentro *blockchain*, e serem executadas de maneira automática.

O software desenvolvido neste trabalho utiliza tecnologia de criptomoedas que por sua vez se baseia na tecnologia de criptografia assimétrica, permitindo a geração de endereços de recebimento e envio de transações, garantindo a unicidade de cada transação na rede e mantendo a segurança na transmissão de dados. Este tipo de tecnologia de criptografia é largamente utilizado por diversas instituições do sistema monetário tradicional (ROUSE, 2016) e a implementação básica de suas funcionalidades encontram-se encapsuladas em *frameworks* de código aberto.

1.1 OBJETIVO GERAL

Utilizar da tecnologia *blockchain* para produzir um aplicativo para dispositivos móveis capaz de gerenciar transações, delegando arbitragem a um participante extra a transação. A implementação desta solução visará capacitar seus usuários a realizarem transações em dispositivos móveis, dando uma maior liberdade financeira aos indivíduos, reduzindo custos e aumentando a competitividade de pequenos negociantes no mercado. A simplificação da utilização de terceiros em arbitragem de conflitos busca tornar o processo menos custoso para as partes, mantendo um nível de segurança adequado para diversos tipos de negociações comuns.

1.2 OBJETIVOS ESPECÍFICOS

- Pesquisar e analisar as dificuldades técnicas em transações de m-n participantes no *blockchain Bitcoin*.
- Implementar uma solução para criptomoedas para dispositivos móveis.
- Explorar capacidades da tecnologia *blockchain*, principalmente para transferências complexas e criação/execução de contratos inteligentes.
- Realizar transações no *blockchain Bitcoin*.
- Realizar transações complexas (arbitradas) no *blockchain Bitcoin*.
- Utilizar preferencialmente *frameworks*, API's , softwares e ferramentas *open source*.
- Utilizar tecnologias de programação para dispositivos móveis.

2 A TECNOLOGIA NO SISTEMA MONETÁRIO

O uso de *smart contracts* e, mais especificamente, de arbitragem em transações ainda se mostra pouco explorado pelos usuários de criptomoedas, porém existem softwares que permitem a utilização do recurso. Electrum é a mais popular carteira Bitcoin, sendo rica em recursos e tento sua confiabilidade baseada no fato de ser *open source* (SEDGWICK, 2019). Ela suporta o gerenciamento de valores no esquema até 2-de-2, baseando-se em chaves públicas dos envolvidos para gerar o endereço de recebimento. A realização do envio permite a segunda assinatura

necessária para validação da transação seja realizada transferindo um arquivo para um pendrive, usando *QR codes* ou usando um servidor remoto com o chamado *CosignerPlugin* (ELECTRUM DOCS, 2019). Outra opção de carteira com suporte multi-assinaturas é a Armory, uma carteira também *open source*, para usuários avançados que suporta o gerenciamento de transações até 7-de-7, sendo considerada ideal para configurar uma carteira onde os fundos provavelmente serão guardados em um "armazenamento frio" por um longo período, e o acesso infrequente. Esta carteira está disponível apenas para *desktop* (SEDGWICK, 2019). Uma opção diferente é a aplicação Casa, que oferece um serviço de gerenciamento de fundos multi-assinaturas chamado KeyMaster, onde é possível efetivar um pagamento periódico para que eles gerenciem uma das chaves privadas da negociação multi-assinatura (SEDGWICK, 2019). Esta organização especifica esta modalidade de negócio como *Sovereignty-as-a-Service*, ou "soberania como um serviço", enfatizando a segurança, privacidade e a confiabilidade (WELCH, 2019). Na variante de carteiras inteiramente *on-line*, a Carbonwallet roda diretamente no navegador, alegando utilizar do sistema de multi-assinaturas a fim de que o usuário possa delegar o armazenamento dos valores aos servidores do serviço. A geração da chave ocorre no lado do cliente, via *javascript*, e utiliza-se do esquema multi-assinaturas e encriptação com senha para geração das chaves privadas do cliente e do serviço, combinando-as a fim de criar a carteira. As chaves mantidas no servidor do serviço são encriptadas com senha a fim de garantir a posse do cliente (CARBOMWALLET.COM, 2019).

Outra solução em desenvolvimento chama-se Bitrated, que planeja oferecer um serviço envolvendo arbitragem e uma camada de reputação a rede Bitcoin. O serviço destaca a criação de um *marketplace* a ser utilizado para arbitragem e construção de reputação, tanto para negociantes quanto para árbitros, disponibilizando uma API (atualmente em estado alfa) que proverá dados e integrará com ecossistemas de criptomoedas a fim de que seus usuários carreguem suas reputações (BITRATED, 2019).

3 MATERIAIS E MÉTODOS

3.1 CRIPTOGRAFIA

Criptografia é o estudo de técnicas matemáticas relacionadas a aspectos da segurança da informação como confidencialidade, integridade de dados, autenticação de entidades e autenticação de origem de dados (KATZ et al., 1996). A tecnologia de criptomoedas funciona amplamente baseada em métodos de cálculo de *hash* e de criptografia assimétricos (ANTONOPOULOS, 2017). As ferramentas de criptografia podem ser avaliadas conforme critérios como nível de segurança, funcionalidade, métodos de operação, performance e facilidade de implementação. O entendimento das atuais ferramentas de criptografia exige a compreensão do conceito de função, no sentido matemático, que no escopo da criptografia podem ser referidas como mapeamento ou transformação (KATZ et al., 1996).

Um dos recursos utilizados em criptografia são a chamadas funções de uma via (*One-way Functions*) definidas como: Uma função f de um conjunto X para um conjunto Y cujo $f(x)$ é "fácil"de computar para todo x pertencente a X mas para "essencialmente todos"elementos y pertencentes a $Im(f)$ é computacionalmente inviável encontrar qualquer x pertencente a X o qual $f(x) = y$. Dado $X = \{1, 2, 3, ..., 16\}$ e definido $f(x) = r_x$ para todo x pertencente a X onde r_x é o resto quando 3^x é dividido por 17 (ver tabela 1).

Dado um número entre 1 e 16, é relativamente fácil encontrar a imagem dele abaixo de f. De qualquer forma, dado um número como 7, sem a tabela 1, é difícil encontrar x dado que $f(x) = 7$. Claro que, se o número dado é 3, o valor de x será facilmente encontrado, x será 1. O ponto é que aplicar a função em um $x(f(x))$, para achar seu y, é fácil e relativamente muito mais fácil que dado somente a $f(x)$ e consequentemente seu resultado, encontrar o x que resolve para esse resultado (KATZ et al., 1996).

Tabela 1 – Funções de uma via

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$f(x)$	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1

Outro recurso utilizado largamente em criptomoedas são os chamados *hashs*, um subtipo de função de uma via, que são computacionalmente eficientes para mapear dados de tamanho arbitrário para dados de tamanho fixo. A ideia básica do cálculo de *hash* é que o valor do *hash* de determinado dado serve como uma representação compacta de um tamanho sempre igual deste dado. Para uma função de *hash* ter uso criptográfico, deve ser computacionalmente inviável encontrar dois valores de entrada para ela, que computam para um mesmo valor de saída *hash* (tal fato, se ocorre, é chamado de colisão). Um uso comum de funções de para cálculo de *hashs* se dá na verificação de integridade de dados ao longo do tempo, onde um *hash* gerado anteriormente é comparado com um novo cálculo de *hash* sobre os mesmos dados a fim de verificar que não ocorreram mudanças (KATZ et al., 1996).

Funções do tipo *trapdoor*, ou funções de uma via alçapão, são outro subtipo de função de uma via, na qual, apesar da dificuldade de encontrar x dado $f(x)$, é possível dar um segundo dado a ser computado (chamado informação de *trapdoor*), o qual torna o cálculo facilitado (KATZ et al., 1996).

A criptografia de chave pública utiliza de *one-way trapdoor functions*. O uso deste tipo de função permite a criação de duas chaves criptográficas distintas, com a propriedade que se usada uma destas chaves para encriptação, somente a outra poderá decriptar a cifra criada. Nem mesmo a chave que foi usada para criar a cifra pode descriptografar a mensagem. Um esquema de criptografia é chamado de esquema de encriptação de chave-pública se para cada par de encriptação/decriptação (e, d), uma chave e (a chave pública) fica publicamente disponível, enquanto a outra chave d (a chave privada) é mantida secreta. Para o esquema ser considerado seguro, é necessário que seja computacionalmente inviável computar d a partir de e. Como analogia é possível pensar em um cofre com uma combinação a qual somente Bob conhece. Se Bob deixar o cofre aberto, qualquer um pode colocar dados no cofre e trancá-lo, mas somente Bob poderá abri-lo. Nem mesmo quem trancou o cofre poderá fazê-lo (KATZ et al., 1996).

Através deste tipo de criptografia são criadas assinaturas digitais, que provêm um meio pelo qual uma entidade tem sua identidade ligada a uma informação (mesmo propósito de qualquer método de assinatura). Para tal é utilizada de criptografia assimétrica, determinando a autenticidade de um determinado dado, sem conhecer o dado em si, mas apenas aplicando uma função computacional criptográfica que gera um dado derivado. Este dado derivado é guardado e posteriormente utilizado para comparação e avaliação da autenticidade. Devido ao caráter publico da função a ser aplicada para gerar a assinatura, e a sua extrema dificuldade em reversão, é assumido que apenas o conhecedor do dado original pode gerar o dado derivado, comprovando assim sua autenticidade (DIFFIE; HELLMAN, 1976).

3.2 BLOCKCHAIN

Um *blockchain* é análogo a um livro contábil ou livro razão, contendo todas as transações já executadas em determinada rede. Ele está constantemente crescendo ao passo que os chamados mineradores adicionam novos blocos a ele (a cada 10 minutos aproximadamente) para registrar as transações mais recentes. A terminologia pode se tornar confusa devido as palavras Bitcoin e *blockchain* serem usada para se referir as três partes do conceito: a tecnologia base *blockchain*, o protocolo/cliente através do qual transações são efetivadas, e a atual criptomoeda. O *blockchain* é um banco de dados compartilhado e descentralizado, localizando-se nos nodos da

rede, atualizado por "mineradores", monitorado por todos, possuído e controlado por ninguém. Em termos monetários, a tecnologia *blockchain* soluciona o problema do gasto duplo combinando compartilhamento de arquivos *peer-to-peer* BitTorrent com criptografia de chave publica (SWAN, 2015).

A tecnologia *blockchain* tem capacidades não só no que se refere a transações de compra e venda, mas também engloba a capacidade de fornecer a estrutura necessária para execução de contratos inteligentes, que são transações com condições mais complexas embutidas, onde a infraestrutura da tecnologia é capaz de fazer o papel de intermediário e impor a execução das condições no tempo de execução do contrato. Um tipo elementar de contrato inteligente (*smart contract*) é a transação com multi-assinatura, na qual um intermediário é envolvido na transação prestando o serviço de árbitro, tendo seu poder garantido pela tecnologia (ANTONOPOULOS, 2017).

3.3 BITCOIN

Bitcoin é dinheiro digital. É uma moeda digital e um sistema de pagamento *on-line* no qual técnicas de encriptação são utilizadas para regular a geração de unidades de moeda e verificar a transferência de fundos, operando de maneira independente de um banco central (SWAN, 2015). Bitcoin constitui-se de uma coleção de conceitos e tecnologias que formaram a mais conhecida e valorizada criptomoeda do mercado. Unidades da moeda digital são utilizadas para guardar e transmitir valor pelos participantes da rede. Os seus usuários se comunicam através do protocolo Bitcoin, principalmente via internet, mas no entanto, outras redes podem ser utilizadas. O protocolo Bitcoin é disponibilizado de maneira *open source* e pode rodar em uma larga gama de dispositivos de computação, incluindo computadores portáteis e *smartphones*, fazendo esta tecnologia altamente acessível (ANTONOPOULOS, 2017).

Transações permitem aos usuários da rede de criptomoedas o gasto de valores representados pelas próprias transações (BITCOIN.ORG, 2019a). Em termos simples uma transação diz a rede que o possuidor de um valor em criptomoeda autorizou a transferência dele para um novo dono. Este, pode criar outra transação para transferi-la para um subsequente e assim por diante (ANTONOPOULOS, 2017). Uma transação contém diversos dados com funcionalidades diversas usadas em contextos diversos. Abaixo segue a uma descrição simplificada de uma transação em um contexto comum, do tipo *Pay-to-Public-Key-Hash* (que será definido mais adiante).

Alice, a compradora, encontra um anúncio de um *smartphone* usado, o qual Bob, o vendedor, dispõe a opção de receber o pagamento em Bitcoins. Eles marcam de se encontrar para realizar a transação. A fim de que Alice possa enviar o valor que ela já possui previamente, Bob deverá enviar o endereço no qual ele deseja receber os valores. Para Bob criar um endereço Bitcoin para recebimento ele gera um par de chaves pública e privada com o ECDSA (*Elliptic Curve Digital Signature Algorithm*) com a curva secp256k1, algoritmo padrão para essa etapa da operação na rede Bitcoin. Após, ele executa o cálculo do *hash* da chave pública (pubkey) deste par. Ele constrói o endereço com um número de versão, este *hash* da pubkey, e um *checksum* para verificação de erros de digitação, e codifica tudo em uma string base58.

Alice de posse desse endereço, o decodifica para acessar o *hash* da pubkey do Bob. Ela utiliza este *hash* para criar o chamado pubkey *script* da transação. Quem provar possuir a chave privada gerada juntamente com a chave pública disponível no pubkey *script* poderá gastar esta transação. Ela constrói o output da transação informando no campo *value*, a quantidade de unidades a ser enviada, e referenciando este pubkey *script* recém criado. O input da transação que Alice cria referência, através dos chamados txid's, *outputs* de uma ou mais transações anteriores, registradas no *blockchain*, a qual ela possui a chave privada correspondente ao *hash* da chave pública contido no pubkey *script*. Os *outputs* destas transações anteriores devem ter a soma do campo value resultante em no mínimo o valor do informado no output da transação que Alice

está criando. Este conjunto de referências é chamado *outpoint*. Para o *input*, Alice também cria o *signature script* ou scriptSigs. Este *script* contém a chave pública completa informada para endereçar os *scripts* aos quais ela está referenciando no campo *outpoint*. Ele também contém dados das transações origem assinados digitalmente com a chave privada correspondente a cada chave pública de cada output referenciado no *outpoint*. Alice então constrói o *input* da transação com o conjunto de referências a *outputs* de origem (campo *outpoint*), o *signature script* e *sequence numbers* (números de sequência).

Por fim, Alice divulga essa transação, que é validada pelos pares da rede, adicionada ao *blockchain*, e categorizada como UTXO (*Unspend Transaction Output*), pois ainda não existem inputs subsequentes apontando para o output desta transação (BITCOIN.ORG, 2019a).

Uma transação divulgada na rede só é incluída no *blockchain* após ser validada pelo processo de mineração. A validação de uma transação na rede se dá com base nas regras de consenso incutidas na mesma, aplicadas nos dados disponibilizados pela transação e divulgadas para todos os nós da rede (ANTONOPOULOS, 2017). A criptomoeda Bitcoin utiliza de uma linguagem de *script* para transações chamada simplesmente de "Script", que é baseada em um execução em pilha, de notação polonesa reversa, parecida com a linguagem Forth. Ela também é uma linguagem classificada como Touring incompleta e sem estado (ANTONOPOULOS, 2017).

Transações em Bitcoin são transmitidas entre pares na rede em um formato serializado, chamado de formato bruto. Diversas ferramentas utilizadas para manipular dados da rede utilizam a notação hexadecimal para exibi-las. Abaixo segue tabela com descrição de uma transação no chamado *transaction-level* (BITCOIN.ORG, 2019a):

Tabela 2 – Estrutura de uma transação.

Bytes	Nome	Tipo de Dados	Descrição
4	*version*	uint32_t	Versão da transação.
Variável	tx_in count	compactSize uint	Números de inputs da transação.
Variável	tx_in	txIn	Inputs da transação.
Variável	tx_out count	compactSize uint	Número de *outputs* da transação.
Variável	tx_out	txOut	Outputs da transação.
4	lock_time	uint32_t	Um tempo (*Unix epoch time*) ou número de bloco.

- *VERSION*: Campo com a função de possibilitar a implementação de novas regras sem invalidar anteriores.
- TX_IN_COUNT: Quantidade de *inputs* da transação.
- *TX_IN*: *Inputs* da transação, contendo três campos: o *outpoint*, um *signature script* e um *sequence number* (BITCOIN.ORG, 2019b).
- TX_OUT_COUNT: Quantidade de *outputs* da transação.
- TX_OUT: *Outputs* da transação, contendo dois campos: o campo *value* informando o valor da transação em unidades e um pubkey *script* (BITCOIN.ORG, 2019b).
- LOCK_TIME: O campo LOCK_TIME existe no *transaction-level* desde o começo da utilização da tecnologia. Tal campo define, a partir de que tempo aquela transação poderá ser gasta pelo possuidor (válida na rede). Tal campo adiciona a dimensão de tempo a transações, assemelhando-se com um cheque pré-datado, mas com garantia de não compensação prévia ao tempo estipulado. Valores abaixo de 500 milhões no campo, estipulam o bloco futuro do *blockchain* no qual a transação será válida, valores acima deste são interpretados como *Unix Epoch timestamp's* (segundos desde 1-Jan-1970). O valor zero não restringe a validação

imediata da transação. Essa configuração não tem uma grande limitação. O recebedor não tem garantia que aquela transação ainda não terá sido gasta até a data que ele pode fazê-lo, ou seja, o pagador poderá gastar e invalidar a transação antes de ser possível para o recebedor. A fim de implementar a garantia de validade de transação com LOCK_TIME, foi implementado posteriormente na rede (BIB-65) o CLTV (*Check Lock Time Verify*) que permite a validação da transação do pagador, mas não possibilita o gasto pelo recebedor até o tempo futuro definido (ANTONOPOULOS, 2017).

Transações processadas na rede Bitcoin podem enviar Bitcoins para novos donos com um *script Pay to Public Key Hash* ou P2PKH *script*. Estes *scripts* estão contidos na parte anteriormente citada, o *output* da transação. Estes *scripts* de travamento efetivamente travam a transação para um *hash* de chave pública, mais conhecido como endereço Bitcoin. Eles podem ser destravados (gastos), essencialmente, pela apresentação de uma chave pública e de uma assinatura digital correspondente a chave privada da chave pública utilizada para a criação do endereço Bitcoin travado (ANTONOPOULOS, 2017). Também existem os chamados *Pay to Script Hash*, ou P2SH. São *scripts* de um tipo de transação mais novo, introduzido em 2012, que simplifica a criação de uma transação complexa, através da substituição dos *scripts* de validação complexos por uma assinatura digital deles (ANTONOPOULOS, 2017).

O recurso de *scripts* de multi-assinatura define condição onde N chaves públicas são gravadas no script e pelo menos M dos controladores das chaves privadas do par devem fornecer assinaturas a fim de desbloquear os fundos representados na transação (ANTONOPOULOS, 2017).

A comunidade de desenvolvedores Bitcoin organiza as melhorias propostas nas chamadas BIP's (*Bitcoin Improvement Proposals*). Dentre as diversas propostas, a própria comunidade busca consenso para decidir quais serão integradas a rede porém, qualquer uma das propostas só é efetivamente colocada em ação se os próprios usuários da rede escolherem atualizar seus softwares para versões contendo-as, denotando sentido ao termo software de consenso (BITCOIN.ORG, 2019c). Dentre os BIP's mais interessantes, existe o BIP 32, que utiliza de um esquema de derivação de chaves capaz de, através de uma "semente", gerar chaves privadas válidas ou mesmo delegar a geração de chaves públicas a um terceiro, de maneira segura, mantendo em segredo a chave privada (DORIER, 2018).

3.4 C# E .NET

C# é uma linguagem de programação orientada a objetos, produzida pela Microsoft, visando o desenvolvimento para seu sistema operacional. Ela é uma linguagem independente, porém altamente relacionada ao chamado *framework* . Este define um ambiente que suporta o desenvolvimento e execução de aplicações altamente distribuídas, baseadas em componentes. Ele habilita diferentes linguagens de computador a trabalhar juntas e provê segurança, portabilidade de programas, e um modelo de programação comum inicialmente voltado a plataforma Windows. Para tal, ele define duas entidades de suma importância, o CLR (*Common Language Runtime*), responsável por gerenciar a execução de programas, e a .NET *Framework Class Library*, uma vasta biblioteca de classes que dispõe de diversos recursos prontos para utilização em um programa (SCHILDT, 2010).

Entre os recursos mais interessantes da linguagem, estão a programação assíncrona embutida (palavras chave *async* e *await*) capaz de facilitar muito a implementação de concorrência (MICROSOFT, 2019d), as expressões lambda, que são blocos de código que são tratados como objetos e podem ser passados como argumentos a métodos ou retornados em chamadas a métodos (MICROSOFT, 2019b), a integração de expressões de consulta muito parecida com SQL, chamada LINQ (*Language Integrated Query*) (MICROSOFT, 2019c).

Em combinação com a linguagem e o *framework*, existe a possibilidade de uso da ferramenta Visual Studio, uma poderosa IDE capaz de facilitar a realização de tarefas comuns de programação, dispondo da tecnologia IntelliCode, que usa inteligência artificial no auxilio da codificação (AUGUSTO, 2019).

3.5 XAMARIN

Xamarin é uma solução para desenvolvimento móvel multiplataforma. O Xamarin oferece ao desenvolvedor um caminho para criar uma aplicação multiplataforma, sem sacrificar o uso de componentes de interface de usuário nativos ou mesmo performance, enquanto permite a reutilização de código entre as plataformas. Isto é feito através da unificação do desenvolvimento em uma única linguagem de programação, o C#. Ele utilizado o Mono *runtime* que foi desenvolvido para trazer a linguagem C# para os sistemas operacionais Linux e Macintosh (DICKSON, 2013).

Para criação da interface de usuário do aplicativo utilizou-se de XAML (*eXtensible Aplication Markup Language*), linguagem de marcação baseada em XML, criada pela Microsoft como uma alternativa a código de programação para instanciação e inicialização de objetos, assim como organização dos mesmos em hierarquias pais-filhos. XAML tem sido adaptado para diversas tecnologias dentro do .NET *framework*, mas tem seu principal uso na definição de interfaces de usuário (MICROSOFT, 2019a).

O Xamarin faz parte da IDE Visual Studio e os termos de uso da IDE aplicam-se ao mesmo. Assim sendo, a licença individual garante que: "Se você for uma pessoa física trabalhando em seus próprios aplicativos, seja para vendê-los ou para qualquer outra finalidade, você poderá usar o software para desenvolver e testar esses aplicativos"(MICROSOFT, 2019) .

3.6 *FRAMEWORK* NBITCOIN

NBitcoin é uma biblioteca Bitcoin que utiliza o *framework*.NET. Ela é de código aberto e mantida pela comunidade e por seu criador, Nicolas Dorier. NBitcoin dispõe de ferramentas básicas para a implementação de softwares capazes de interagir com diferentes *blockchains* e facilita a construção de soluções para criptomoedas usando a linguagem de programação C#. Ela implementa a grande maioria dos BIP's Bitcoin (HE, 2018).

NBitcoin é a mais completa biblioteca Bitcoin para o *framework* .NET. Ela implementa todas as mais relevantes BIP's (*Bitcoin Improvement Proposals*) e também provê acesso de baixo nível as primitivas Bitcoin, possibilitando e facilitando a construção de aplicações Bitcoin baseadas nela (DORIER, 2019).

3.7 API'S E QBIT NINJA

Uma API (*Application Programing Interface*) ou Interface de Programação de Aplicação, é um conjunto de protocolos, rotina, funções e/ou comandos que programadores usam para desenvolver software ou facilitar a interação entre sistemas distintos (TECHOPEDIA, 2018). Existem inúmeras API's disponíveis para criptomoedas, cuja finalidade é fornecer informações sobre um ou mais *blockchains* no qual as criptomoedas se baseiam. Elas podem ser classificadas quanto a funcionalidades como: (a) integração com carteiras: capacidade de trabalhar com diversas carteiras. (b) suporte a transações: capacidade de enviar ou receber as mesmas. (c) preço. (d) capacidades especiais (LASTCALL, 2018).

Nesta seção citaram-se API's comerciais através da tabela 3.7 com diversas características e algumas tendo um custo para utilização. Para fins de desenvolvimento acadêmico, destacou-se a API QBit Ninja, que é uma API desenvolvida em código aberto, criada pelo mesmo autor do *framework* C# NBitcoin sendo dependente da classe `Indexer` do *framework* NBitcoin que por sua vez apoia-se no serviço de armazenamento Azure da Microsoft. Devido a mesma ser de

Tabela 3 – Tabela Comparativa de Algumas API's.:

API	Integração	Suporte a Transações	Habilidades Especiais	Preço
Digital Currency Tickers	Não	Não	Sim	Livre
Crypto Market Intraday Reference Rates	Não	Não	Sim	Livre
CoinAPI	Não	Não	Sim	Livre
ICOs	Não	Não	Sim	Planos Variados
Due Diligence	Não	Não	Sim	Livre
Global Bitcoin Price Index	Não	Não	Sim	Livre
Coinbase	Sim	Sim	Sim	Livre
CoinMarketCap	Não	Não	Sim	Livre
Nexchange	Sim	Sim	Sim	Taxas de Câmbio
GetBalance	Sim	Não	Não	Livre
BitcoinAverage	Não	Não	Sim	Planos Variados
Bitcointy	Não	Não	Sim	Livre
Bitcoin ATMs	Não	Não	Sim	Planos Variados

Fonte: (LASTCALL, 2018)

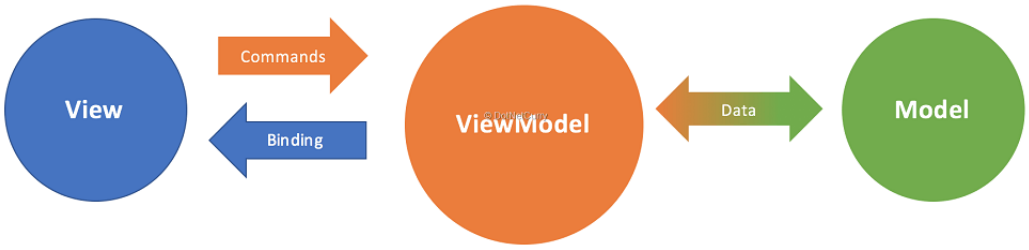
código aberto e de facilitada utilização e extensão, ela foi utilizada no desenvolvimento deste projeto. Ela funciona como um *web service* capaz de consultar o *blockchain* (DORIER, 2017).

4 O DESENVOLVIMENTO DO APLICATIVO

A fim de atingir os objetivos deste trabalho foi realizada ampla pesquisa *on-line* sobre sistema monetário, criptografia, a tecnologia *blockchain*, a mais conhecida implementação desta, a rede *peer-to-peer* Bitcoin, assim como a plataforma Xamarin, para programação de dispositivos móveis multiplataforma, juntamente a linguagem de programação C# e o *framework* .NET, utilizados pelas plataformas de desenvolvimento e *framework open source* NBitcoin, além da linguagem de marcação XAML, usada na interface gráfica do aplicativo, e da API QBit Ninja utilizadas para consulta do *blockchain Bitcoin* em sua rede de teste, a *TestNet*.

A interface gráfica do aplicativo foi baseada no padrão de *design Model-View-View-Model* (MVVM), ilustrado na Figura 1, o qual desacoplou a mesma das implementações do negócio. Cada classe do *namespace Views* traz um arquivo XAML descrevendo as características como botões e campos que existem na interface de usuário. As classes do gênero foram responsáveis por inicializar os rótulos e pontualmente, por tratar eventos restritos a interface gráfica, como cópias para a área de transferência.

Figura 1 – MVVM



Fonte: Versluis (2017).

As classes do *namespace ViewModels* implementaram a regra de negócio em conjunto com as classes `ExploradorBlockchain` e `Negociador` do *namespace Services*. Não foi necessária a delimitação de *Models*. Os elementos de interface gráfica das *Views* responsáveis por mostrar dados gerados pela lógica de negócio são ligados ás respectivas *ViewModels* através de *data bindings*, definidos nos arquivos XAML e gerenciados pelo *framework* através do levante de *PropertyChangedEventArgs* nos métodos assessores do dado.

Os eventos gerados pelo usuário através da interação com a interface são tratados através do envio de *commands*. Os *commands* funcionam em conjunto com os *bindings* visto que a fim de usar a interface *Command*, devem ser criados *bindings* do elemento da interface gráfica com uma propriedade do tipo *Command* da *ViewModel* correspondente. Do uso deste padrão de *design*, foi possível implementar de maneira facilitada a desativação dos botões da interface assincronamente durante o aguardo do retorno dos dados, além dos próprios comandos a serem executados pela interação com os diferentes elementos da interface.

A classe de serviço `Negociador` foi implementada usando o *framework* NBitcoin, se baseando em uma geração determinística de chaves privadas com base em uma chave "mestra" codificada nele. O aplicativo disponibiliza em sua tela inicial o saldo desta chave privada mestra utilizada como base para o desenvolvimento e testes. A geração de endereço de recebimento comum utiliza apenas a chave mestra para gerar um endereço válido. A geração de endereço de recebimento arbitrado utilizou de duas chaves públicas a mais, referentes aos outros envolvidos na transação, para a criação do *script multi-sign*.

Para envios de valores de forma normal, foi utilizado um endereço de destino combinado com o valor, onde foi construída uma `Transaction` com base em fundos disponíveis nas transações criadas com destino a chaves privadas codificadas no aplicativo. O gerenciamento de chaves privadas não monitora fundos disponíveis para gasto com as chaves derivadas.

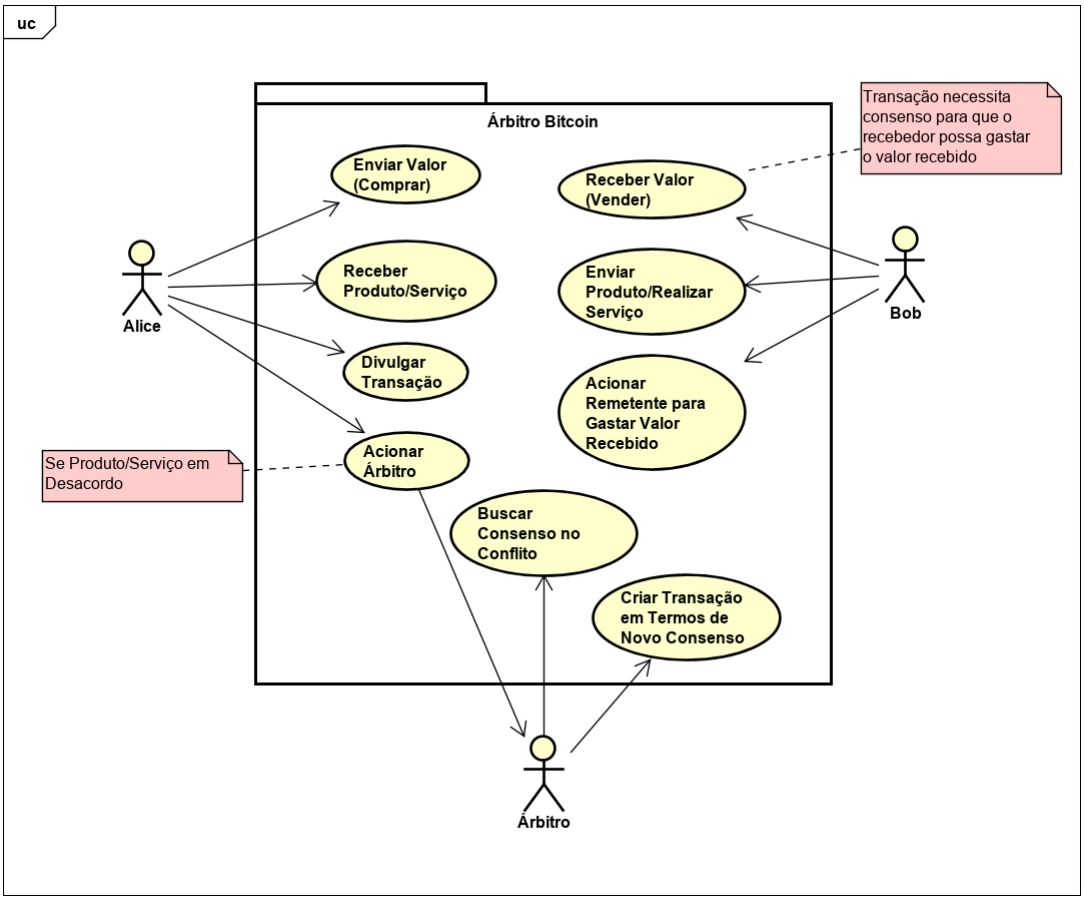
O envio de transação arbitrada requereu uma estruturação de um processo mais complexo devido à natureza de transações arbitradas necessitar de assinaturas geradas por pelo menos mais um dos três negociantes envolvidos. O aplicativo permite apenas a configuração de transação arbitrada 2 de 3, sendo assim, optou-se por planejar a exportação de arquivo serializado da classe `Transaction`, inicialmente valorado pelo gastador, para envio via meio eletrônico diverso para o segundo ente da transação importá-lo, verificar se os valores estão de acordo, assinar, serializar, exportar e retornar o arquivo para nova importação. Nesta importação ocorre validação dos valores e destinatários, assinatura final, e divulgação na rede. O processo de exportação inicial será realizado na tela de envio, através de *switch* para alteração da funcionalidade do botão Enviar para Exportar. A importação será realizada através da tela Arbitrar, a qual disponibilizará a visualização dos dados da transação e opção para assinatura final e divulgação, ou modificação de transação e nova exportação, caso seja necessário adequá-la a novas condições determinadas pelo consenso com o outro ente.

A classe de serviço ExploradorBlockchain usou a API QBitNinja a fim de realizar consultas no *blockchain* provendo dados para o processo de envio e foi responsável pela divulgação da transação na rede.

5 RESULTADOS E DISCUSSÃO

Através desta pesquisa foi produzido o aplicativo "Árbitro Bitcoin". Este aplicativo visou incorporar os objetivos deste trabalho através das tecnologias referenciadas, utilizando uma metodologia prática. Os casos de uso dele para uma negociação arbitrada encontram-se lustrados na Figura 2.

Figura 2 – Casos de Uso do Aplicativo



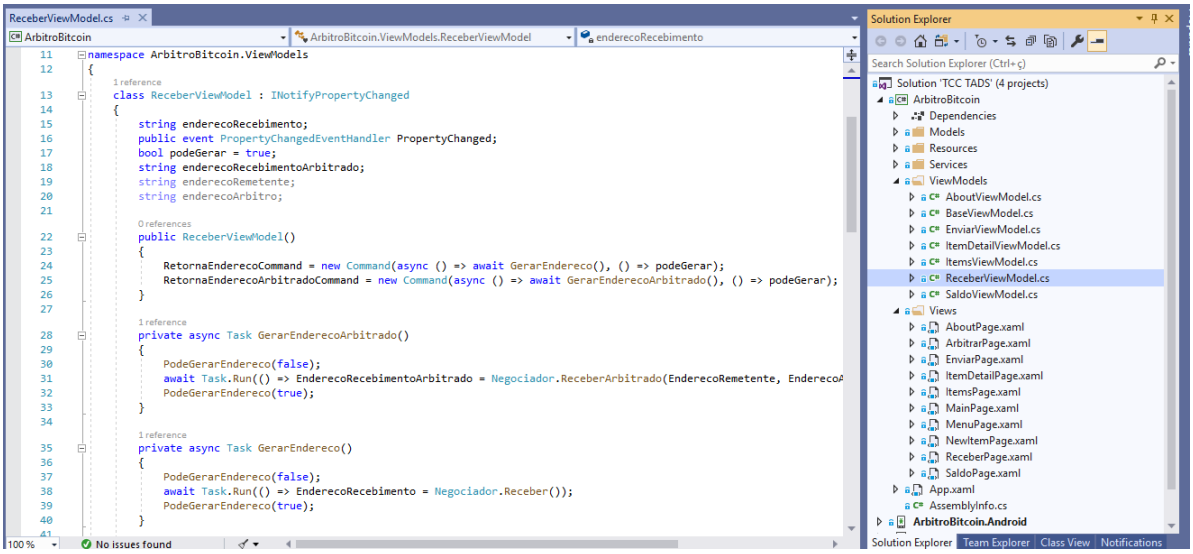
Fonte: o autor.

O software produzido tem as seguintes funcionalidades implementadas: (a) Consulta e exibição de saldo de um determinado endereço da rede; (b) Criação de endereço de recebimento; (c) Criação de endereço de recebimento arbitrado; (d) Criação de transações comuns para endereço; (e) Divulgação de transação. As seguintes funcionalidades não estão implementadas totalmente: (f) Criação de transações multi-assinaturas (2 de 3); (g) Importação, verificação, assinatura e exportação de transação serializada.

A Figura 3 mostra a estrutura de diretórios do projeto juntamente ao código produzido para implementação da funcionalidade de geração de endereços. No desenvolvimento do software foi utilizada a linguagem de programação C#, que baseia-se no paradigma de programação orientado a objetos, sendo fortemente tipada e permitindo a criação de softwares robustos e seguros executados no .NET *framework*, mantendo uma sintaxe parecida com as principais linguagens de programação do mercado (WAGNER et al., 2015). A utilização desta linguagem proveu um amplo arsenal de recursos prontos os quais foram utilizados, como a linguagem de consulta embutida (LINQ), expressões lambda, assessores de propriedades simplificados e as

funcionalidades de paralelismo embutidas na própria linguagem, através das palavras-chave *async* e *await*. Estes recursos de paralelismo permitiram rodar processos demorados, como consultas a API *web* e difusão de transações, em uma linha de processamento paralela, mantendo a interface do usuário responsiva.

Figura 3 – Código: Recebimento



Fonte: o autor.

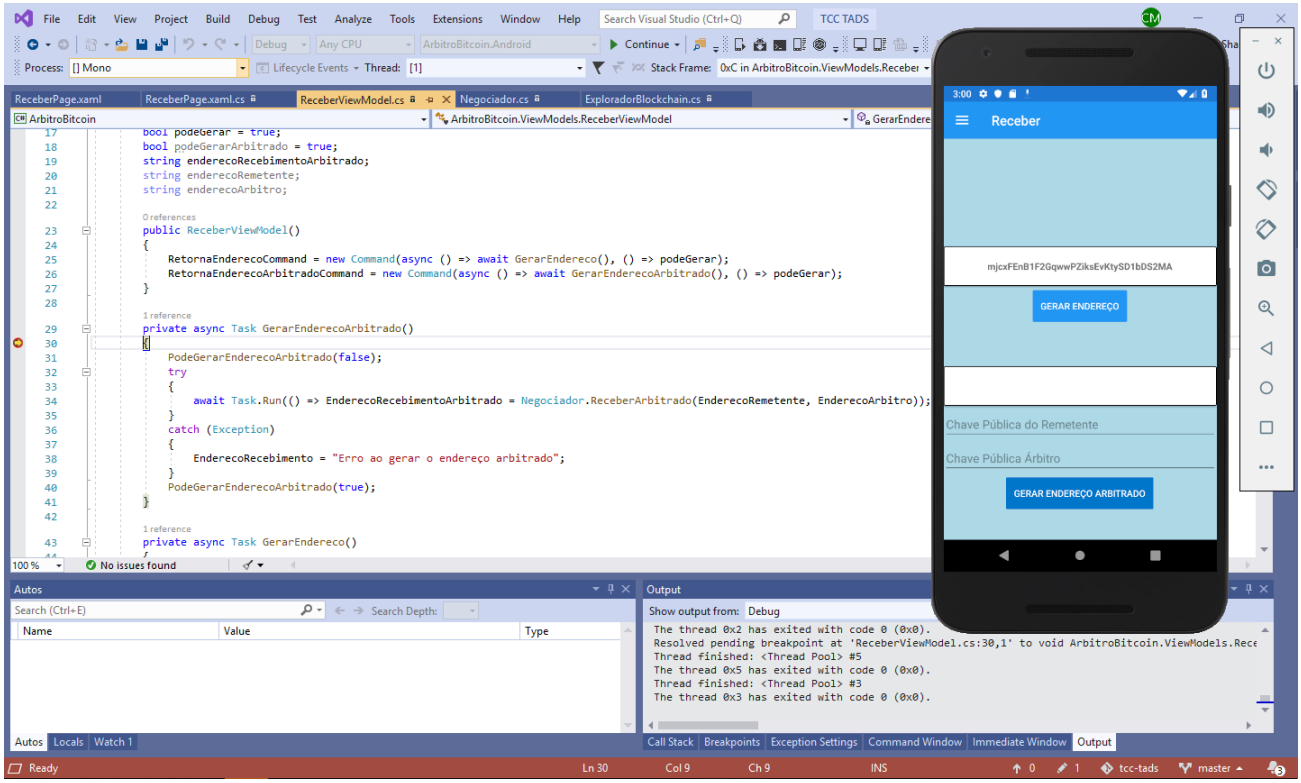
A plataforma de desenvolvimento Xamarin propiciou um desenvolvimento multi-plataforma rápido e robusto, dando uma estrutura clara e concisa ao projeto. Além disso, o ambiente de desenvolvimento integrado Visual Studio 2019 revelou-se altamente preparado para projetos do gênero, trazendo agilidade na construção inicial da estrutura básica projeto e auxiliando no controle de código fonte de maneira transparente e simples através do sistema de gerenciamento de código-fonte Git, utilizando do serviço de hospedagem Bitbucket. A IDE integra recurso de dispositivos virtuais Android (AVD - *Android Virtual Devices*) e integração com sistema operacional iOS, sendo possível a utilização dos dispositivos para *debug* diretamente na interface do Visual Studio exigindo porém, um hardware robusto para uma execução performática.

Devido a limitações de hardware no desenvolvimento em algumas máquinas utilizadas, optou-se por utilizar o *deploy* do projeto diretamente em dispositivo *Android* disponível, sendo essa modalidade igualmente compatível com recursos de *debug* disponíveis nas máquinas virtuais, sem sofrer com lentidão em função do hardware utilizado para compilação, *deploy* e *debug*. A Figura 4 mostra o processo e *debug* em conjunto com o dispositivo virtual Android disponibilizado na IDE Visual Studio 2019.

A interface de usuário declarada em XAML mostrou-se simples de criar, e a implementação do padrão de design *Model-View-View-Model* permitiu que o software fosse construído de maneira desacoplada e preparada para expansão. O recurso de *data bindings* embutido na plataforma permitiu uma implementação transparente da atualização de dados do *backend* do aplicativo com sua interface gráfica de usuário. Já utilização do padrão de projeto *Command* na manipulação de interações do usuário com botões e controles da interface, simplificou o disparo de eventos a serem tratados no *backend* assim como trouxe responsividade a interface gráfica, devido a capacidade embutida de gerenciar a ativação e desativação do controle de interface, enquanto se aguardada por resposta de *threads* executadas paralelamente.

A fim de analisar as informações da rede, foi utilizado serviço web da API QBit Ninja,

Figura 4 – *Debug* do Aplicativo



Fonte: o autor.

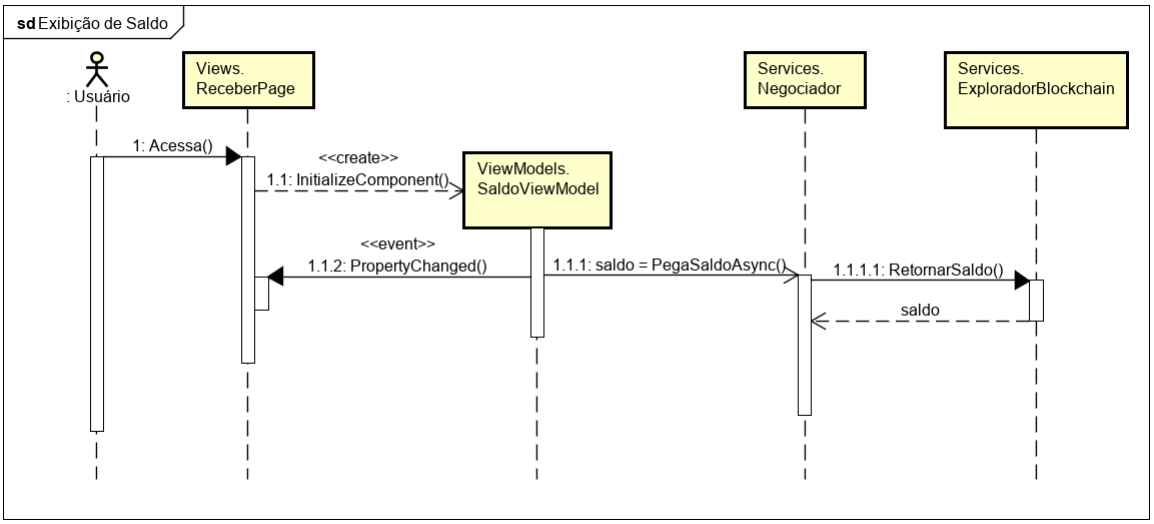
que é hospedada por terceiros que disponibiliza informações sobre blocos, transações e endereços no *blockchain* Bitcoin (DORIER, 2018), tanto em sua rede principal, a *MainNet*, quanto em na rede utilizada para testes, a *TestNet*. Consulta de saldo e divulgação de transação são dois exemplos da utilização da API. A rede de criptomoedas na qual o software se baseou foi a do Bitcoin.

As funcionalidades do aplicativo foram implementadas através do uso do *framework* NBitcoin, utilizado nas classes `Negociador` e `ExploradorBlockchain`. A classe `Negociador` define o método `Receber` e `ReceberArbitrado` e utiliza do método `RetornaPrivateKey` a fim de receber o `BitcoinSecret` da chave mestra vinculada a carteira e que define seu saldo. O gerenciamento de chaves privadas se mostrou não trivial. A geração determinística de chaves simplificou parte do processo de geração, porém evidenciou-se a necessidade de criação de estrutura a fim armazenar chaves e análises dos saldos disponíveis em cada uma, não tendo sido identificada estrutura com tal nível de abstração disponível no *framework*.

A Figura 5 mostra a sequência do processo de exibição de saldo na página inicial do aplicativo. Este processo consulta, através da API QBitNinja, os saldos relacionados a um endereço codificado no aplicativo o mostra na tela principal, assincronamente. O saldo consiste essencialmente de transações cujo o destino é o endereço especificado, e que não tenham referencia nos *outpoints* de nenhuma transação subsequente (o que tornaria o status do valor representado pela transação em gasto).

A geração do endereço de recebimento normal utiliza apenas uma chave privada gerada com base na chave mestra determinística, abstraída na classe `ExtKey` do *framework*. Já a criação do endereço de recebimento arbitrado necessita das chaves públicas do remetente, do destinatário

Figura 5 – Exibição de Saldo



Fonte: o autor.

e de um árbitro, diretamente. Ambos os tipos de endereços de recebimento ao fim do processo são gerados no formato *Pay to Public Key Hash* ou P2PKH. A geração de endereço arbitrado foi planejada de maneira a requerer 2 de 3 assinaturas digitais para que uma transação enviada para o endereço possa ser posteriormente gasta.

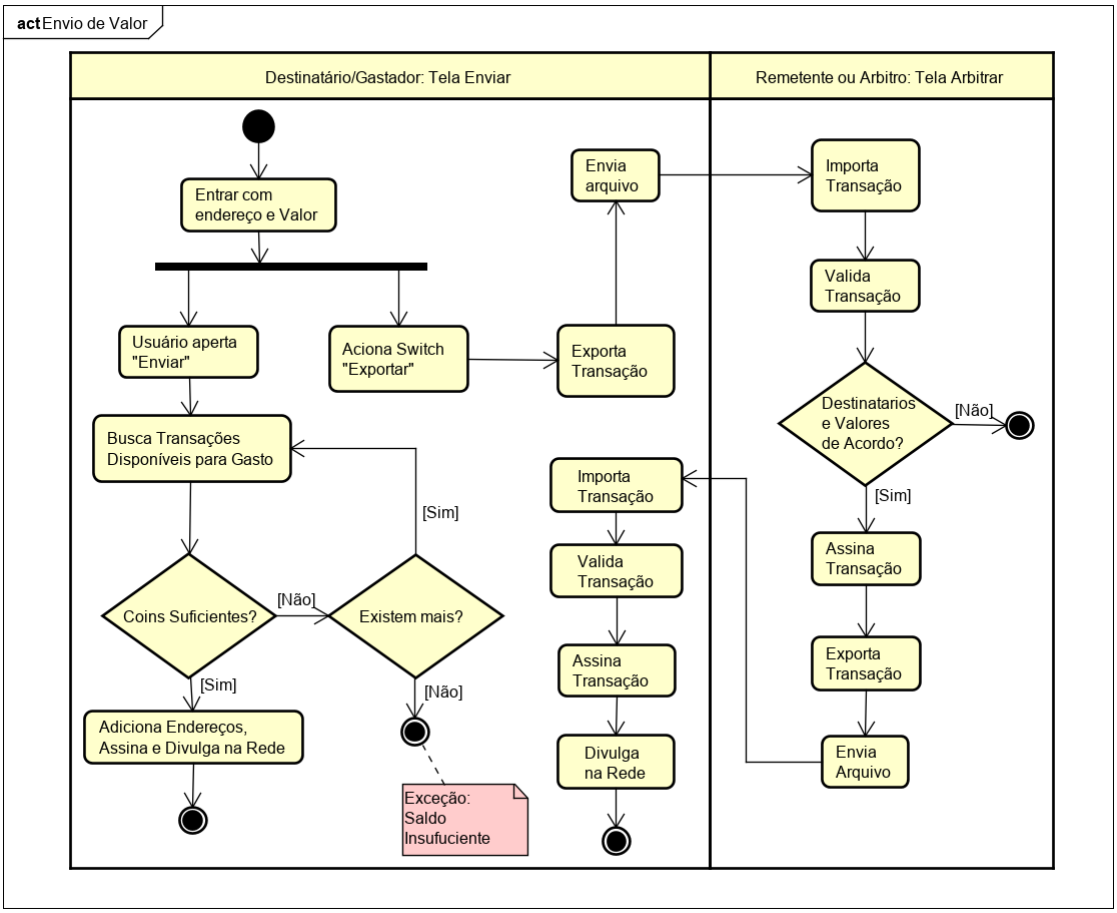
O envio de valores em Bitcoins é realizado pelo método `Enviar` da classe `Negociador`. O envio normal necessita do endereço do destinatário e do valor a ser enviado, onde então é criada a transação e divulgada na rede. Já o envio utilizando um árbitro necessita que o remetente da negociação crie a mesma com base nas chaves públicas do árbitro e do destinatário, e defina quantas assinaturas são necessárias para que a transação seja gasta posteriormente. A construção da transação para gastar um valor recebido na forma de multi-assinaturas requer o número de assinaturas conforme determinado em sua criação. Este projeto busca resolver essa questão através da serialização e exportação para arquivo do objeto da classe `Transaction`.

Quando o destinatário de uma transação arbitrada desejar gastar os valores recebidos nesta transação, a tela de envio permite a exportação de uma transação parcial. Esta transação conterá somente o destinatário e o valor. O arquivo com a transação será importado pelo aplicativo do remetente ou do árbitro envolvidos, na tela Arbitrar, sendo que os destinatários e o valor da transação serão expostos em tela, juntamente aos botões assinar e cancelar. Se realizada assinatura, será realizada nova exportação do arquivo para devolução ao requerente da assinatura onde na mesma tela Arbitrar ele realizará a importação, nova verificação de destinatários, valor e assinaturas realizadas, realizando ele a assinatura.

Identificado o atingimento de 2 de 3 assinaturas necessárias, a transação será divulgada na rede. Conforme citado anteriormente, não foi implementado um gerenciamento avançado de chaves privadas, sendo assim, não se faz possível escolher transação que requeira multi-assinaturas para gasto, sendo utilizada uma implementação estática nos testes. O cancelamento na tela Arbitrar removerá o arquivo importado da memória. Na Figura 6 são mostradas as atividades realizadas pelos dois tipos de processo de envio.

Caso o destinatário da transação não cumpra com sua parte na negociação, enviando um produto ou prestando um serviço em desacordo com o ofertado, o remetente da mesma pode entrar em contato com o árbitro para resolver a questão. O árbitro então pode: (a) Criar

Figura 6 – Envio de Valor



Fonte: o autor.

uma transação devolvendo os fundos para o remetente; (b) Criar uma transação pagando o destinatário conforme originalmente acordado; (c) Criar uma transação com percentuais dos valores para o remetente e para o destinatário, a fim de atingir um consenso.

Tendo o árbitro conseguido um consenso quanto aos valores com mais um dos três participantes da transação, ele utilizará a tela Arbitrar a fim de realizar o processo de exportação/importação com este participante, assim como a divulgação da transação.

A transação pode conter uma taxa destinada ao árbitro, pagando por seus serviços. Como o árbitro necessita da assinatura de mais um participante da transação, ele não poderá sequestrar ou desviar o valor da transação para algum endereço seu sem o consenso de pelo menos mais um participante da transação. Além desta taxa, a transação deve conter um excedente utilizado como taxa de mineração aos mineradores de rede e necessário para que a transação seja validada na rede. Esta taxa está fixada em 80 Bit ou 8000 satoshis.

6 CONSIDERAÇÕES FINAIS

O presente trabalho permitiu a utilização de ferramentas, metodologias e tecnologias inovadoras, utilizando amplamente de materiais de código aberto e software livre, assim como ferramentas licenciadas para uso em projetos de código livre, ou por usuários individuais. Também

propiciou aprendizado de linguagem de programação dotada de amplos recursos e *framework* robusto.

A opção por desenvolver o projeto utilizando a plataforma Xamarim mostrou-se eficaz e com curva de aprendizado rápida. A escolha foi realizada principalmente devido a utilização do *framework* NBitcoin, escrito e feito para C# e o *.NET framework*, e integrou-se de maneira transparente ao mesmo, e possibilitando seu uso pleno. Tal opção pode ser avaliada como uma característica positiva a um projeto deste gênero por permitir alcançar uma gama maior de dispositivos devido às capacidades multiplataforma embutidas na plataforma Xamarin, além da surpreendente receptividade dos termos de uso dos softwares proprietários a projetos de cunho individual e/ou livre.

O trabalho com código aberto se mostrou desafiador devido a uma documentação limitada do *framework* NBitcoin, exigindo uma abordagem exploratória, porém a base teórica sobre a tecnologia *blockchain* provida pela pesquisa foi decisiva na interpretação das abstrações e informações disponíveis a fim de produzir os resultados.

A implementação de transações comuns se mostrou muito facilitada pelo *framework* e API, onde a maior dificuldade do processo mostrou ser o gerenciamento da carteira de chaves privadas determinísticas. Este gerenciamento exige uma análise das transações com base em uma chave privada "mestra", sistematicamente colecionando dados relacionados a fim de prover ao usuário o controle total sobre os valores e representando parte crítica da implementação, mas secundária aos objetivos deste trabalho.

A realização de transações arbitradas revelou seu maior desafio na necessidade de realizar assinaturas parciais. Tais mecanismos exigem vasta gama de testes que tem sua complexidade elevada devido a exigir interações com múltiplos participantes e seus respectivos endereços em instâncias diferentes do aplicativo, exigindo um desenvolvimento planejado a fim de simplificar a realização de testes unitários.

Este trabalho gerou um aplicativo com capacidades de transação limitadas e em estágio inicial de desenvolvimento, porém permitiu a resolução do problema de pesquisa de maneira otimista, definindo uma abordagem através da qual mostra-se possível implementar em completude a realização de transações arbitradas que potencialmente podem ser utilizadas em negociações reais dando as pessoas uma opção mais barata e simples de obter segurança. A tecnologia *blockchain* está permitindo que toda uma nova gama de soluções emerja e dentre estas soluções, as criptomoedas se destacam, tendo em seus recursos uma ampla reserva de capacidades inexploradas. O desenvolvimento de aplicações utilizando a tecnologia *blockchain* que busquem solucionar a questão confiança e segurança em transações é viável e deve representar uma nova revolução, que trará maior poder a cada individuo da nossa sociedade, contribuindo para uma sociedade mais justa, segura e economicamente desenvolvida.

REFERÊNCIAS

ABREU, Y. V. de; COELHO, S. B. *EVOLUÇÃO HISTÓRICA DA MOEDA*: Estudo de caso: Brasil (1889 – 1989). 2009. ed. Universidade de Málaga, 2009. Disponível em: <http://www.eumed.net/libros-gratis/2009a/477/index.htm>. Acesso em: 26 mai. 2019. Citado na página 2.

ANTONOPOULOS, A. M. *Mastering Bitcoin*: Programming the open blockchain. 2. ed. O'Reilly Media, 2017. Disponível em: <https://github.com/bitcoinbook>. Acesso em: 29 nov. 2018. Citado 4 vezes nas páginas 4, 6, 7 e 8.

AUGUSTO, T. Microsoft lança visual studio 2019 para windows e mac. *Canaltech*, jun. 2019. Disponível em: <https://canaltech.com.br/software/microsoft-lanca-visual-studio-2019-para-windows-e-mac-136237/>. Acesso em: 05 jun. 2019. Citado na página 9.

BITCOIN.ORG. *Bitcoin Developer Documentation*. [S.l.], 2019. Disponível em: <https://bitcoin.org/en/developer-documentation>. Acesso em: 16 set. 2018. Citado 2 vezes nas páginas 6 e 7.

BITCOIN.ORG. *Bitcoin Developer Glossary*. 2019. Disponível em: <https://bitcoin.org/en/developer-glossary>. Acesso em: 24 fev. 2019. Citado na página 7.

BITCOIN.ORG. *Bitcoin Improvement Proposals Readme*. [S.l.], 2019. Disponível em: <https://github.com/bitcoin/bips>. Acesso em: 09 jun. 2019. Citado na página 8.

BITRATED. *BITCOIN TRUST PLATFORM*. [S.l.], 2019. Disponível em: <https://www.bitrated.com/>. Acesso em: 04 jun. 2019. Citado na página 4.

CARBOMWALLET.COM. *Multi Signature Online Cryptocurrency Wallet*. [S.l.], 2019. Disponível em: <https://carbonwallet.com>. Acesso em: 28 mai. 2019. Citado na página 4.

DICKSON, J. Xamarin mobile development. 2013. Citado na página 9.

DIFFIE, W.; HELLMAN, M. New directions in cryptography. *IEEE transactions on Information Theory*, IEEE, v. 22, n. 6, p. 644–654, 1976. Citado na página 5.

DORIER, N. *QBit Ninja*. [S.l.], 2017. Disponível em: <https://qbitninja.docs.apiary.io/#>. Acesso em: 17 mar. 2019. Citado na página 10.

DORIER, N. *Programming The Blockchain in C#*. 2. ed. [s.n.], 2018. Disponível em: <https://programmingblockchain.gitbook.io/programmingblockchain>. Acesso em: 29 nov. 2018. Citado 2 vezes nas páginas 8 e 14.

DORIER, N. Nbitcoin. *GitHub*, mai. 2019. Disponível em: <https://github.com/MetacoSA/NBitcoin>. Acesso em: 26 mai. 2019. Citado na página 9.

ELECTRUM DOCS. *Multisig Wallets*. [S.l.], 2019. Disponível em: <https://electrum.readthedocs.io/en/latest/multisig.html>. Acesso em: 28 mai. 2019. Citado na página 4.

ENCYCLOPAEDIA BRITANNICA. *Gold Standard*. Encyclopædia Britannica, inc, 2018. Disponível em: <https://www.britannica.com/topic/gold-standard>. Acesso em: 16 set. 2018. Citado na página 1.

HE, H. Nbitcoin: Introduction to nbitcoin. *C#Corner*, aug. 2018. Disponível em: <https://www.c-sharpcorner.com/article/introduction-to-nbitcoin/>. Acesso em: 31 mar. 2019. Citado na página 9.

KATZ, J. et al. *Handbook of applied cryptography*. [S.l.]: CRC press, 1996. Citado 2 vezes nas páginas 4 e 5.

LASTCALL. The top 13 bitcoin, blockchain & cryptocurrency apis. out. 2018. Disponível em: <https://blog.rapidapi.com/bitcoin-blockchain-cryptocurrency-apis/>. Acesso em: 17 mar. 2019. Citado 2 vezes nas páginas 9 e 10.

MICROSOFT. *Documentação do Xamarin*. [S.l.], 2019. Disponível em: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms>. Acesso em: 26 mai. 2019. Citado na página 9.

MICROSOFT. *Lambda expressions*. [S.l.], 2019. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/statements-expressions-operators/lambda-expressions>. Acesso em: 05 jun. 2019. Citado na página 8.

MICROSOFT. *Language Integrated Query (LINQ)*. [S.l.], 2019. Disponível em: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/>. Acesso em: 05 jun. 2019. Citado na página 8.

MICROSOFT. *The Task asynchronous programming model in C#*. [S.l.], 2019. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/concepts/async/>. Acesso em: 05 jun. 2019. Citado na página 8.

MICROSOFT. *TERMOS DE LICENÇA PARA SOFTWARE MICROSOFT*. [S.l.], 2019. Disponível em: <https://visualstudio.microsoft.com/pt-br/license-terms/mlt031819/>. Acesso em: 09 jun. 2019. Citado na página 9.

NOGUEIRA, M. História do comércio. *estudo prático*, 2018. Disponível em: <https://www.estudopratico.com.br/historia-do-comercio/>. Acesso em: 15 set. 2018. Citado na página 1.

PRIBEREAM. *Dinheiro*. 2008–2013. Disponível em: <https://www.priberam.pt/dlpo/Dinheiro>. Acesso em: 15 set. 2018. Citado na página 2.

ROTHBARD, M. Fractional reserve banking. *FEE - Foundation for Economic Education*, out. 1995. Disponível em: <https://fee.org/articles/fractional-reserve-banking-part-ii/>. Acesso em: 16 set. 2018. Citado na página 2.

ROUSE, M. asymmetric cryptography (public key cryptography). *SearchSecurity*, jun. 2016. Disponível em: <https://searchsecurity.techtarget.com/definition/asymmetric-cryptography>. Acesso em: 29 nov. 2018. Citado na página 3.

SCHILDT, H. *C# 4.0: The complete reference*. [S.l.]: Tata McGraw-Hill Education, 2010. Citado na página 8.

SEDGWICK, K. How to use multisig to keep your coins ultra-safe. *Bitcoin.com*, mar. 2019. Disponível em: <https://news.bitcoin.com/how-to-use-multisig-to-keep-your-coins-ultra-safe/>. Acesso em: 28 mai. 2019. Citado 2 vezes nas páginas 3 e 4.

SWAN, M. *Blockchain: Blueprint for a new economy*. [S.l.]: "O'Reilly Media, Inc.", 2015. Citado na página 6.

TECHOPEDIA. Application programming interface (api). 2018. Disponível em: <https://www.techopedia.com/definition/24407/application-programming-interface-api>. Acesso em: 17 mar. 2019. Citado na página 9.

VERSLUIS, G. Using mvvm in your xamarin.forms app. *dotnetcurry.com*, ago. 2017. Disponível em: <https://www.dotnetcurry.com/xamarin/1382/mvvm-in-xamarin-forms>. Acesso em: 09 jun. 2019. Citado na página 10.

WAGNER et al. Introdução à linguagem c# e ao .net framework. jul. 2015. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>. Acesso em: 29 nov. 2018. Citado na página 12.

WELCH, J. Casa is sovereignty-as-a-service. *Casablog*, fev. 2019. Disponível em: <https://blog.keys.casa/casa-is-sovereignty-as-a-service/>. Acesso em: 28 mai. 2019. Citado na página 4.