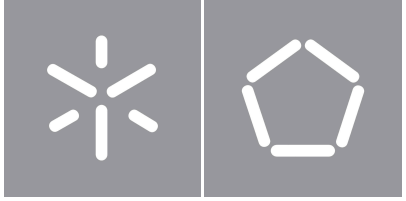
Universidade do Minho
Escola de Engenharia

João Pedro Araújo Parente

Gestão de permissões e acesso a dados para Hyperledger Fabric

novembro 2022



Universidade do Minho
Escola de Engenharia

João Pedro Araújo Parente

Gestão de permissões e acesso a dados para Hyperledger Fabric

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Trabalho efetuado sob a orientação de
Doutor Fábio André Coelho
Doutora Ana Nunes Alonso

novembro 2022

Direitos de Autor e Condições de Utilização do Trabalho por Terceiros

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho:

CC BY-NC

https://creativecommons.org/licenses/by-nc/4.0/ *[Esta licença permite que outros remisturem, adaptem e criem a partir do seu trabalho para fins não comerciais, e embora os novos trabalhos tenham de lhe atribuir o devido crédito e não possam ser usados para fins comerciais, eles não têm de licenciar esses trabalhos derivados ao abrigo dos mesmos termos.]*

Agradecimentos

O autor do trabalho agradece a bolsa concedida pelo INESC TEC.

Este trabalho é financiado por fundos nacionais através da FCT – Fundação para a Ciência e a Tecnologia no âmbito do Projeto LA/P/0063/2020.

Agradece-se também à Nau21 - Software for the Future, LDA e ao projeto P2020 SIS[1] com a referência NORTE-01-0247-FEDER-45355.

Declaração de Integridade

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Universidade do Minho, Braga, novembro 2022

Assinado por: **João Pedro Araújo Parente**
Num. de Identificação: 14956016
Data: 2022.11.15 17:26:43+00'00'

João Pedro Araújo Parente

Resumo

A gestão de acesso e permissões afeta várias áreas como a da saúde, mais concretamente os registos de saúde eletrónicos, que se espera que nos próximo anos cresça exponencialmente e alcancem um valor no mercado de $39.7 biliões no ano 2022. A utilização de *blockchain* aparece como uma solução para estes cenários onde existem diversos domínios em que os dados são potencialmente sensíveis, quer como dados pessoais, quer como dados que podem revelar segredos de negócio.

Algumas tecnologias, como o HyperLedger Fabric, já prometem resolver estes problemas, mas sempre com uma granularidade baixa, com bastantes limitações ao nível da definição de políticas de acesso e de transformações dos dados. No contexto de HyperLedger Fabric vamos implementar um mecanismo de gestão de permissões flexível, que além de diferenciar o acesso com base na identidade de quem faz o pedido permite considerar um conjunto de atributos configurável. Adicionalmente, além de decisões binárias sobre o acesso, o sistema que implementamos permite implementar políticas que definem transformações a ser aplicadas aos dados acedidos.

Palavras-chave Gestão de Permissões, Blockchain, Hyperledger Fabric, Privacidade, Confidencialidade.

Abstract

The management of access and permissions affects several areas such as health, more specifically electronic health records, which are expected to grow exponentially in the coming years and reach a market value of $39.7 billion in the year 2022. The use of *blockchain* comes up as a solution for these scenarios where there are several domains where data is potentially sensitive, either as personal data or as data that can reveal business secrets.

Some technologies, such as HyperLedger Fabric, already promise to solve these problems, but always with a low granularity, with many limitations in terms of definition of access policies and data transformations. In the context of HyperLedger Fabric, we are going to implement a flexible permissions management mechanism, which, in addition to differentiating access based on the identity of the person making the request, allows us to consider a set of configurable attributes. Additionally, in addition to binary access decisions, the system we implement allows us to implement policies that define transformations to be applied to the accessed data.

Keywords Permission Management, Blockchain, Hyperledger Fabric, Privacy, Confidentiality.

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

ABAC *Attribute Based Access Control*.

ACL *Access Control List*.

B2B *Business-to-Business*.

DLT *Distributed Ledger Technology*.

HLF *Hyperledger Fabric*.

PERMIS *Privilege and Role Management Infrastructure Standards*.

PoS *Proof of Stake*.

PoW *Proof of Work*.

SDAM *Smart Data Access Management*.

SDDL *Security Descriptor Definition Language*.

XACML *eXtensible Access Control Markup Language*.

Capítulo 1

Introdução

O conceito de **Distributed Ledger Technology (DLT)** é utilizado para definir um sistema distribuído *peer-to-peer* de base de dados, acessível por múltiplos atores, que mantém uma visão global coerente do estado, evitando a necessidade de depender de uma entidade externa, em que todos os atores precisariam de confiar. A implementação deste tipo de sistema pode ser feita sobre uma *blockchain*, que apresenta a propriedade fundamental de imutabilidade, ou seja, de que não será possível, ou mais precisamente, de que será demasiado custoso modificar o registo de transações já aplicadas sem que tal seja detetado. Daí retira-se também a propriedade de não ser possível negar que uma transação foi efetuada e quais as identidades intervenientes. [Shalaby, 2020]. Estes aspetos tornam este tipo de sistema de registo de transações mais atrativo do que os sistemas tradicionais em cenários onde os vários atores do sistema não se conhecem e/ou não têm confiança entre eles. Além do registo confiável de transações, as implementações de *blockchain* suportam, tipicamente, a execução de contratos inteligentes (*smart contracts*), programas que vivem e são executados no contexto das transações e que permitem implementar código aplicacional. No ecossistema **Hyperledger Fabric (HLF)**, denominam-se por *chaincode*.

As implementações atuais ficam aquém dos sistemas gestores de bases de dados tradicionais no que diz respeito ao nível de gestão de permissões de acesso aos dados. Os mecanismos existentes nestas tecnologias proporcionam tipicamente uma granularidade baixa, com bastantes limitações ao nível da definição de políticas de acesso. Endereçar este problema de baixa granularidade e baixa flexibilidade na definição de políticas que existe nas tecnologias atuais resultará em vários benefícios. Um deles é o facto de os atores conseguirem decidir que informação partilham. Outro é dar a possibilidade ao ator decidir os atores do sistema que podem aceder a essa informação. O último benefício é a capacidade de o ator definir as transformações que a sua informação terá de passar antes que certos atores a possam visualizar. Benefícios como estes tornam esta solução bastante apelativa para casos de uso corporativos. Isto deve-se ao facto de as empresas, dependendo com quem partilham a informação, quererem ajustar a quantidade e a forma como a partilham. Por exemplo, poderá não ser do interesse de uma empresa passar

a informação relativamente ao custo de produção de um produto a um possível cliente, mas fará todo o sentido um gestor da empresa ter acesso a essa informação. Outro exemplo disto, é um funcionário que preste apoio técnico poderá de precisar de alguma informação relativa ao cliente, mas não de conhecer todos os pormenores do contrato associado.

1.1 O problema

A especificação e aplicação de políticas a uma rede num único domínio administrativo é comum nas redes de hoje, sendo considerado como já resolvido. No entanto, o mesmo não acontece em domínios multi-administrativos entre diferentes empresas, [Khan, 2020]. Participantes numa rede **Business-to-Business (B2B)** podem ser extremamente sensíveis sobre que informação partilham e com quem. O problema está em conseguir gerir as permissões que esses atores têm para aceder a esses dados, que é algo que as tecnologias atuais permitem, mas sempre com uma granularidade baixa, com bastantes limitações ao nível da definição de políticas de acesso e de transformações dos dados.

Um cenário que representa o problema pode ser definido por participantes numa rede colaborativa terem confiança limitada uns nos outros e desejarem manter o controlo total sobre as políticas de acesso. Isto ocorre porque, num cenário onde existem várias empresas, as empresas não estão dispostas a deixar o controlo de acesso para terceiros. Além disso, porque cada empresa deve ser capaz de revogar qualquer política de acesso a qualquer momento, respetivamente [Khan, 2020].

Os mecanismos de controlo de acesso existentes no **HLF**, além de apenas suportarem uma granularidade baixa, foram desenhados para o controlo de acesso de organizações a conjuntos relacionados de dados. Assim, não se adequam à definição e implementação de políticas relacionadas com o consentimento de utilizadores ou indivíduos específicos, tal como, a possibilidade de um dado utilizador (ou indivíduo) dar o seu consentimento para a utilização dos seus dados para um propósito específico, mas não para outros (por exemplo, marketing), no contexto da mesma organização. Embora existam alternativas, como a implementação de controlo de acesso mais fino ao nível dos métodos, ao nível do *chaincode*, estas pecam pela dificuldade de manutenção, já que seria necessária a implementação em todos e pela dificuldade de alteração, já que cada alteração à política *chaincode* implicaria, muito provavelmente, a alteração de múltiplos *chaincodes* bem como do acordo de múltiplas organizações para que as novas versões fiquem ativas. Assim, este método de atualização também não é adequado para suportar a dinâmica de alterações que poderá advir das decisões de consentimento dos utilizadores/indivíduos.

Em particular, este trabalho de dissertação foi motivado pelo desenvolvimento de uma plataforma co-

laborativa que permite que processos no domínio dos Seguros sejam desmaterializados e automatizados, em particular na negociação contratual e processamento de pedidos. Na rede participam várias entidades de foros diversos. A questão de lidar com dados pessoais e o consentimento de clientes tem impacto não apenas nos processos mais diretos de gestão de apólices mas também em outros usos, como para a análise automatizada do negócio, usando por exemplo, aprendizagem automática, ou para propósitos de divulgação comercial.

1.2 Objetivos e Contribuições

O objetivo desta dissertação foi o desenvolvimento de um mecanismo de gestão permissões e controlo de acesso a dados, compatível com a utilização de *smart contracts* implementados em *chaincode* em **HLF**, que seja suficientemente flexível para acomodar as diferentes necessidades que podem surgir por parte de diferentes organizações que participem em redes colaborativas e que permita integrar a informação dinâmica relativa ao consentimento de utilizadores/indivíduos. Adicionalmente, pretende-se que a integração do mecanismo tenha um impacto mínimo na implementação do **HLF** e que a definição e gestão de políticas seja o mais simples possível.

Como contribuição principal, apresenta-se um mecanismo de gestão de permissões, preparado para ser inserido *de forma transparente* em redes colaborativas suportadas pelo **HLF**, que melhora o sistema existente nos seguintes aspetos:

- permite definir e implementar políticas de acesso mais complexas e mais flexíveis, permitindo adicionalmente a utilização de atributos específicos à camada aplicacional;
- a definição e alteração de políticas é mais flexível, não sendo necessário alterar definições da rede, canais ou *chaincode* para o efeito;
- passa a suportar uma linguagem (**eXtensible Access Control Markup Language (XACML)**) e um motor de regras especificamente desenhados para a implementação de políticas de acesso a dados;
- além de decisões binárias sobre a permissão de acesso, permite aplicar transformações obrigatórias sobre os dados de acordo com as políticas definidas.

Adicionalmente, o mecanismo foi desenvolvido e avaliado em termos de desempenho, quer por componente, quer de forma holística, sendo comparado com uma versão de **HLF** sem este mecanismo,

tendo concluído que o impacto no desempenho é aceitável tendo em consideração os benefícios da sua utilização.

Como contribuição adicional, foi desenvolvido um estudo sobre algumas tecnologias *blockchain*(Bitcoin, Ethereum, **HLF**) e a capacidade de gestão de permissões, exemplificando as políticas de acesso a dados que se podem definir com o mecanismo tradicional em **HLF** em **XACML**.

O trabalho desenvolvido no contexto desta dissertação deu origem ao artigo "Flexible Fine-grained Data Access Management for Hyperledger Fabric" [J. Parente, 2022] publicado na conferência BCCA 2022 - *International Conference on Blockchain Computing and Applications*.

Capítulo 2

Trabalho Relacionado

2.1 Blockchain

Em 2008 foi apresentado por Satoshi Nakamoto a primeira *blockchain*, Bitcoin, uma rede *peer-to-peer* que permite transferir montantes de uma criptomoeda, gerada e gerida pela própria blockchain Bitcoin, de uma conta para outra, sem necessitar de uma terceira parte confiável como, por exemplo, um banco [Nakamoto, 2008].

Um sistema *Blockchain* instancia um conjunto de blocos de dados, interligados entre si. Um bloco é composto por 3 secções: *Data*, *Hash* e *Previous Hash*. A secção *Data* contém toda a informação do bloco, ou seja um conjunto de transações. A secção *Hash* contém um resumo criptográfico gerado aplicando uma função de *hash* ao conteúdo do bloco (Figura 1). Pelas propriedades da função de resumo criptográfico, qualquer alteração a uma transação do bloco leva a que se necessite de recalcular a chave. A secção *Previous Hash* contém o resumo criptográfico do bloco anterior, o que garante que uma alteração em qualquer bloco anterior consegue detetar-se [Shalaby, 2020].

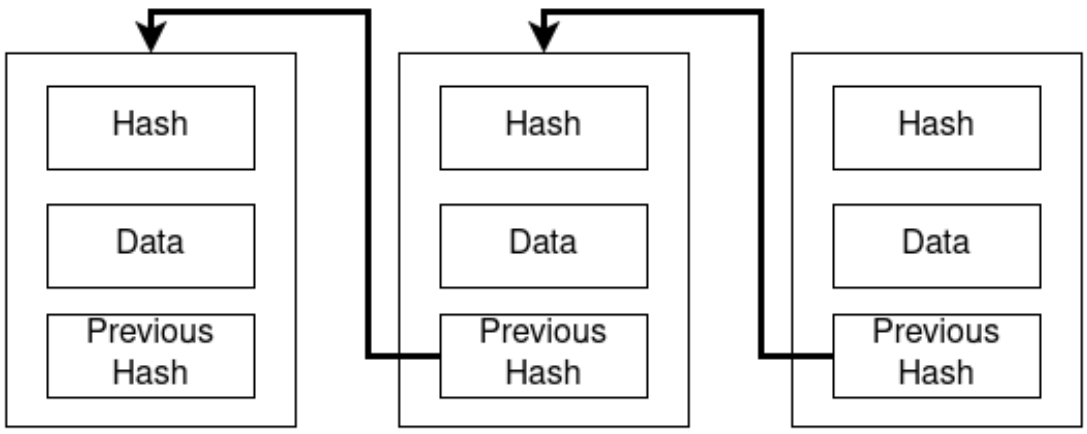


Figura 1: Sequência de blocos.

Cada elemento de uma rede *blockchain* contém uma cópia desta sequência de blocos, que vai evo-

luindo com a criação de transações na rede que são agrupadas em blocos e por fim adicionadas à sequência de blocos. Neste último passo de adicionar novos blocos ao *blockchain* os processos da rede implementam um protocolo de acordo distribuído. Este protocolo define as condições necessárias para um bloco ser inserido no *blockchain* e a ordem pela qual os novos blocos serão inseridos, para que todos os processos da rede adicionem exatamente os mesmos blocos pela mesma ordem.

2.1.1 Smart Contracts

O termo *smart contract* (ou contrato inteligente) foi definido nos meados da década de 1990 pelo cientista de computação e criptógrafo Szabo como um conjunto de promessas especificadas em formato digital. No seu famoso exemplo, Szabo comparou os *smart contracts* a máquinas de venda automática: as máquinas recebem moedas e, por um mecanismo simples, dispensam troco e produto consoante o preço exibido (Figura 2). Este conceito só começou a ser implementado com o aparecimento da tecnologia *blockchain* devido a esta tecnologia ter propriedades como imutabilidade e mecanismos de acordo distribuído que tornaram os *smarts contracts* viáveis [Wang et al., 2019].

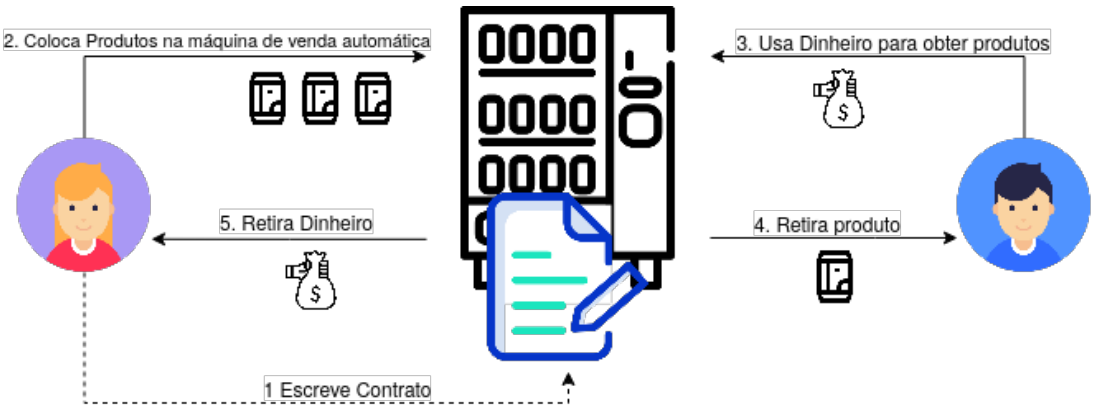


Figura 2: Exemplo de um *smart contract* do criptógrafo Szabo.

Os *smart contracts* facilitam, verificam e aplicam um contrato entre dois ou mais utilizadores de uma *blockchain*, sendo o código do *smart contract* disponibilizado e verificado na *blockchain*, tornando possível ser analisado por qualquer interveniente. Sendo que os contratos são criados e geridos por transações, aportando as mesmas propriedades de segurança das restantes [Wang et al., 2019].

2.1.2 Tipos de Blockchain

Consideramos dois tipos de *blockchains*: com permissões e livres de permissões.

Livres de Permissões

Nesta categoria, qualquer pessoa pode ter acesso e participar na rede, não existindo autoridades para garantir a identificação dos utilizadores, o que as torna realmente descentralizadas. Estas usam, em geral, algoritmos de acordo distribuído como, **Proof of Work (PoW)** ou **Proof of Stake (PoS)**, que envolvem custos consideráveis como parte de um sistema de incentivo para evitar que a rede não seja afetada por possíveis utilizadores mal-intencionados. Tal tem também como consequência que a latência de confirmação de transações seja alta. Como são redes abertas, tendem a ter um grande número de utilizadores, que dificulta a utilização de protocolos de acordo distribuído tolerantes a faltas bizantinas mais tradicionais, como o protocolo PBFT [Castro et al., 1999], cujo desempenho de deteriora com o aumento do número de participantes.

Com Permissões

Em *Blockchains* com permissões ou privadas, as identidades dos intervenientes são conhecidas, sendo que apenas entidades autorizadas podem aceder à rede. Esta característica potencia a utilização de protocolos de acordo distribuído tolerantes a falhas bizantinas mais tradicionais, com latências de confirmação de transações mais baixas, e menor dispêndio energético.

2.1.3 Bitcoin

A rede Bitcoin [Nakamoto, 2008] usa o algoritmo de acordo distribuído **PoW**, em que um grupo de processos (*miners*) verificam e competem para resolver um problema computacionalmente intensivo. Assim que um processo encontra uma solução, este transmite o novo bloco gerado, que inclui um conjunto de transações. Em troca da geração do bloco, o processo é recompensado com um determinado montante na criptomoeda nativa da rede, bitcoin [Lo and Wang, 2014].

A rede Bitcoin suporta *smart contracts* básicos, ou seja, as suas transações serão validadas apenas se certas condições forem satisfeitas. No entanto, projetar um *smart contract* com lógica complexa não é possível devido às limitações da linguagem de *scripting* suportada, que apresenta apenas algumas operações aritméticas, lógicas e criptográficas básicas [Wang et al., 2019].

Sendo uma rede *blockchain* livre de permissões, ou seja, em que a participação apenas requer o download e instalação de uma aplicação, não existe propriamente uma gestão de permissões, pois todos têm acesso aos mesmos dados.

2.1.4 Ethereum

A rede Ethereum foi a primeira plataforma de *blockchain* livre de permissões a suportar *smart contracts* complexos, implementados numa linguagem Turing-completa que executa numa máquina virtual, a *Ethereum Virtual Machine (EVM)*. Cada participante na rede Ethereum executa executa as mesmas instruções numa EVM [Wang et al., 2019].

A rede Ethereum tem também uma criptomoeda nativa, *ether* que pode ser negociada em bolsas de criptomoedas ou transferida entre contas. Os utilizadores transferem ether para outros submetendo transações cujo campo de valor indica a quantidade de ether transferida. A execução de transações exige que os utilizadores paguem taxas em ether. A produção de blocos é premiada com ether [Chen et al., 2020].

Existem duas categorias de contas: contas externas (EOAs) e *smart contracts*. A principal diferença é que apenas os *smart contracts* contêm código executável desenvolvido pelos utilizadores [Chen et al., 2020].

A rede Ethereum permite a criação de aplicações descentralizadas (*dapps* [Cai et al., 2018]) através da implementação de *smart contracts* usando várias linguagens de programação de alto nível, como Solidity e Serpent. O código é compilado para *EVM bytecode* e implantado na *blockchain* para execução [Wang et al., 2019].

Para criar um *smart contract* no Ethereum, uma transação, cujo campo de dados contém o *bytecode* em vez da fonte é enviada para o endereço nulo, indicando a criação do contrato. É de realçar que um *smart contract* pode ser criado por uma conta externa ou por outro *smart contract*. Ao ser criado é gerado um endereço para o *smart contract*. Qualquer utilizador do Ethereum pode invocar o *smart contract* enviando uma transação cujo campo destinatário é o endereço do contrato [Chen et al., 2020].

Sendo uma rede livre de permissões, ou seja aberta para todos, não existe propriamente uma gestão de permissões, pois todos têm acesso aos mesmos dados. No entanto, algumas *frameworks* desenvolvidas pela comunidade têm permitido usar os *smart contracts* do Ethereum com controlo de acesso baseado em **Attribute Based Access Control (ABAC)** [Yutaka et al., 2019].

2.1.5 HyperLedger Fabric

HLF é um projeto de código aberto da Linux Foundation, com o objetivo de ser uma base para o desenvolvimento de aplicações de nível empresarial. Ao contrário das tecnologias Bitcoin e Ethereum, o **HLF** não tem uma criptomoeda nativa, embora seja possível a sua implementação como aplicação.

Smart contracts em **HLF** são programas denominados por *chaincode*. Estes programas podem ser escrito em várias linguagens de programação como Go, JavaScript (node.js) e Java, que implementam uma *interface* prescrita. Estes programas são executados num *container* Docker. É através dos *chaincodes* que se pode inicializar e gerir estado, propondo transações, pela invocação de métodos aí definidos por aplicações cliente ou outros *chaincodes* [Hyperledger]. É de notar que estado criado por um dado *chaincode* só pode ser acedido por este. No entanto, dentro da mesma rede, com a permissão apropriada, um *chaincode* pode invocar métodos de outro *chaincode*, acedendo assim ao seu estado.

Quanto à execução de transações em **HLF**, esta decorre em três etapas: execução, ordenação e validação [Shalaby, 2020].

A fase de execução inicia-se quando uma aplicação cliente envia uma proposta de transação aos processos da rede responsáveis por executar transações (*peers*). Estes executam o *chaincode* associado a essa transação, simulando a execução sobre a base de dados de estado, e cada um envia uma resposta ao cliente.

Na fase de ordenação, o cliente envia as respostas que obteve na fase de execução ao processo ou grupo de processos responsáveis por ordenar as transações e agrupar as transações autorizadas em blocos.

Finalmente, na fase de validação, o(s) ordenador(es) dissemina(m) os blocos para a rede. Todos os processos validam os blocos, procedem à adição dos blocos ao seu *ledger* e atualizam a sua réplica da base de dados de estado.

O tópico de gestão de acesso no **HLF** é analisado em profundidade na Secção 2.3.

2.2 Gestão de acesso

Existem vários tipos de gestão de acesso, entre os quais se destacam três: modelo baseado em funções, modelo baseado em atributos e modelo baseado em identidade. Embora cada modelo tenha as suas próprias vantagens e limitações, com o passar do tempo, os modelos de controlo de acesso evoluíram de ser baseados em identidade para ser baseados em funções e em atributos [H. Guo and Shen., 2019].

Modelos baseados em funções Neste modelo, as permissões são associadas a papéis e cada utilizador é associado a um papel. Um utilizador pode aceder a um recurso se este tiver associado o papel associado ao utilizador [Sandhu, 1998]. Este processo é representado na Figura 3. Uma vantagem deste modelo é o facto de permitir uma integração rápida e simples de novos utilizadores. Por outro lado, sofre do problema de explosão de funções. Um exemplo deste problema é o do

caso hospitalar em que a função de médico precisa de ser definida para cada paciente, para que apenas o médico responsável tenha acesso aos dados do seu paciente. Assim, o número de funções aumentará drasticamente embora tenham em comum quase todas as mesmas permissões [Jin and Krishnan., 2012].

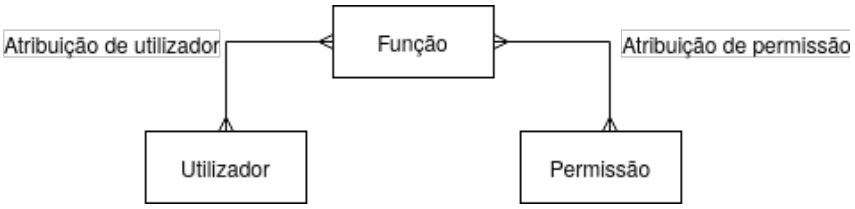


Figura 3: Esquema do modelo baseado em funções [Xia et al., 2014]

.

Modelos baseados em atributos Neste modelo, tanto utilizadores como recursos têm associados a si um conjunto de atributos. As decisões de acesso podem mudar simplesmente alterando os atributos associados, sem exigir mudanças nos relacionamentos sujeito / objeto que definem os conjuntos de regras subjacentes [Hu, 2015].

Modelos baseados em identidade Neste modelo, cada utilizador tem um identificador único e o acesso aos recursos é decidido analisando as características de cada utilizador. Este tipo de modelo é tipicamente usado em situações com poucos utilizadores, pois ter de associar cada utilizador aos seus recursos é algo difícil de escalar para grandes números de utilizadores. Um exemplo é apresentado na Figura 4, onde observamos que, para 100 utilizadores e 100 tabelas, utilizar um modelo baseado em identidade equivale a 10000 atribuições, enquanto que utilizar um modelo de funções apenas requer 200 atribuições. Esta diferença é considerável e vai crescendo com o aumento de utilizadores e tabelas.

É de realçar que é possível combinar diferentes modelos no contexto de um ecossistema de gestão de acesso, conforme descrito em detalhe na Secção 2.2.1.

2.2.1 Linguagens de gestão de acesso

Para analisar algumas linguagens (e ferramentas) de gestão de acesso a recursos é apresentado na Figura 5 um exemplo. Consideremos três pessoas, a **A**na, o **B**runo, e o **P**edro e um recurso, o íman. O Bruno tem um **Í**man e, quando não o está a usar, entrega-o ao Pedro, que o guarda e fica responsável por ele. O Pedro só pode entregar o íman ao Bruno.

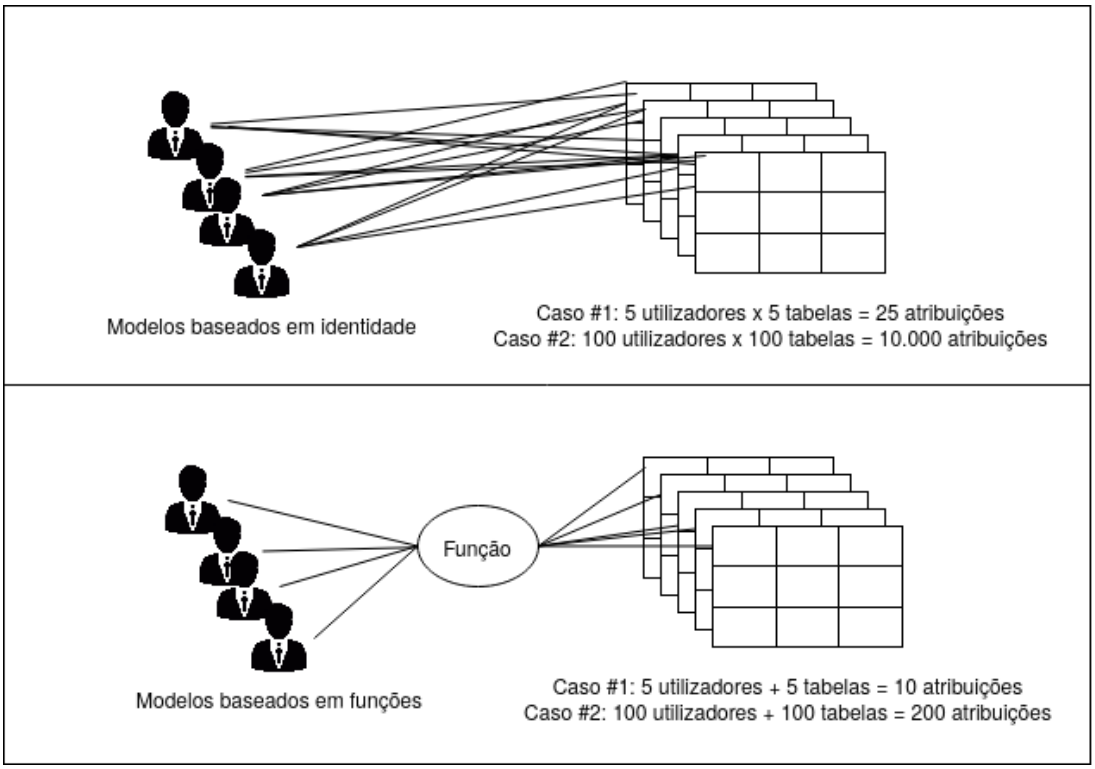


Figura 4: Escalabilidade do modelo baseado em identidade vs do modelo baseado em atributos [Elliott and Knight, 2010].

Este exemplo pode ver ser visto como um problema de gestão de acesso, onde existem dois atores A e B, e um recurso I. Para os atores acederem a I têm de cumprir as regras impostas por uma política P.

PERMIS

Privilege and Role Management Infrastructure Standards (PERMIS)[1] é uma linguagem baseada em XML, baseada no modelo de funções, mas também no modelo de atributos.

Com **PERMIS** as políticas são regras, que podem ser de dois tipos: de confiança ou de privilégios. Regras de confiança especificam a confiança do sistema nas autoridades que atribuem atributos. Regras de privilégios definem a hierarquia de funções, os privilégios associados a cada função e condições especiais para algumas políticas. O **PERMIS** fornece uma ferramenta de composição de políticas, o Editor de Políticas, que os utilizadores podem usar para compor e editar as políticas, evitando assim escrever as políticas em XML [Chadwick et al., 2006].

Algo que diferencia o **PERMIS** de outras linguagens de políticas, como **XACML** (ver Secção 2.2.1), é ter uma componente que faz a validação de credenciais, o que, dependendo do caso em questão, pode ser interessante. Um exemplo disso é o caso onde queremos que os intervenientes do sistema interajam

[1] Permis website http://www.openpermis.info/

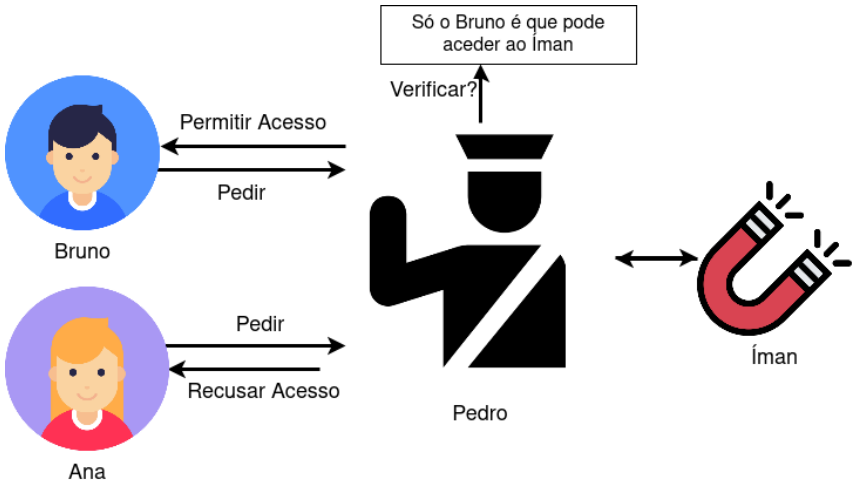


Figura 5: Exemplo de gestão de acesso.

diretamente com o módulo de políticas no sistema para definir as suas políticas. Com **PERMIS**, utiliza-se um ficheiro de configuração em XML para definir as funções existentes, as ações que é possível realizar e a relação entre estas. A resolução do exemplo de gestão de acesso com **PERMIS** é apresentada na Figura 6. Na linha número 5 declara-se uma função **Bruno**, na linha número 10 declarada-se a ação **aceder** e nas linhas número 25 e 28 é autorizada à função **Bruno** a ação **aceder**.

```
<X.509_PMI_RBAC_Policy OID="...">
    <!-- define as funções>
    <RoleHierarchyPolicy>
        <RoleSpec OID="..." Type="permisRole">
            <SupRole Value="Bruno"/>
        </RoleSpec>
    </RoleHierarchyPolicy>
     <!-- define as ações -->
    <ActionPolicy>
        <Action Name="aceder"/>
    </ActionPolicy>
    <-- define as políticas de acesso aos targets -->
    <TargetAccessPolicy>
        <!-- utilizadores sem função não têm direitos -->
        <TargetAccess>
            <RoleList/>
            <TargetList>
            </TargetList>
        </TargetAccess>

        <!-- utilizadores com a função Bruno tem o direito à
        ação aceder -->
        <TargetAccess>
            <RoleList>
                 <Role Type="permisRole" Value="Bruno"/>
            </RoleList>
            <TargetList>
                <Target Actions="aceder">
                </Target>
            </TargetList>
       </TargetAccess>
</X.509_PMI_RBAC_Policy>
```

Figura 6: Ficheiro de configuração de PERMIS que resolve o exemplo de acesso 5

SDDL

Security Descriptor Definition Language (SDDL)[2] é uma linguagem definida pela Microsoft e usada para expressar um descritor de segurança, geralmente como uma *string* de texto. Em maior detalhe, **SDDL** é usado no atributo *nTSecurityDescriptor* para definir uma **Access Control List (ACL)**, e em chaves de registo e arquivos NTFS [Microsoft]. Cada *string* **SDDL** nTSecurityDescriptor é composta por 4 partes primárias que correspondem ao cabeçalho: relação entre objetos e utilizadores DACL (D:), lista de controlo de acesso do sistema SACL (S:) (opcional), grupo primário (G:) e proprietário (O:). Os tipos principais de parâmetros do **SDDL** são códigos SID[3](*Strings que correspondem a valores possíveis dos campos O:,S: e admnistradores em ACE*) ou ACE[4](entradas de D:).

Proprietário O: dono do objeto/registo em questão, identificados por um SID.

Grupo Primário G: grupo primário do objeto, identificados por um SID.

DACL D: "Discretionary Access Control List entries" onde o conteúdo realmente das políticas é definido. Este campo é uma lista onde cada elemento é uma ACE que define uma política. Cada elemento tem os seguintes parâmetros:

```
ace_type;ace_flags;rights;object_guid;inherit_object_guid;account_sid;
```

ace_type especifica o valor do direto, se é autorizado ou não, etc.

ace_flags opções extra;

rights especifica o direito, se é de leitura, escrita, etc.

inherit_object_guid identifica o objeto de quem herda.

object_guid identifica o objeto a gerir a permissão.

account_sid identifica o utilizador a que esta regra se aplica.

A resolução do exemplo de acesso usando **SDDL** é apresentada na Figura 7.

[2] Documentação do **SDDL** oficial da Microsoft em https://docs.microsoft.com/en-us/windows/win32/secauthz/security-descriptor-definition-language

[3] Tabela com os códigos SID possíveis https://docs.microsoft.com/pt-pt/windows/win32/secauthz/sid-strings

[4] Tabela com os códigos ACE possíveis https://docs.microsoft.com/pt-pt/windows/win32/secauthz/ace-strings

```
1   O:BA G:SY D:(A;;id_iman;;;id_bruno)(D;;id_iman;;;id_ana)
```

Figura 7: Problema da Figura 5 resolvido com SDDL.

Interpretando a frase de **SDDL** na Figura 7, temos que o proprietário desta entrada é o BA, que é o código para os administradores do sistema e o grupo é o SY, que significa sistema local. Depois, a primeira entrada das DACL, A diz que autoriza, id_iman identifica o objeto a gerir o acesso, e o id_bruno identifica quem tem este direito. A segunda entrada é análoga com a diferença de que o código D nega o acesso ao id_iman ao utilizador com id_ana.

XACML

Um "standard" na área de gestão de acesso é o **XACML**[5]. Este padrão define uma linguagem declarativa de política de controlo de acesso baseada em atributos, uma arquitetura e um modelo de processamento que descreve como avaliar as solicitações de acesso conforme as regras definidas nas políticas [Open, 2005]. Alternativamente, é possível implementar controlo de acesso baseado em papéis como uma especialização do modelo baseado em atributos.

O **XACML** é composto por 3 elementos principais: <Rule>,<Policy>,<PolicySet>.

1. **Rule:** Contém uma expressão booleana.
2. **Policy:** Contém um conjunto de elementos *Rule*.
3. **PolicySet:** Contêm um conjunto de vários elementos Policy e/ou PolicySet.

[5] Para uma explicação mais detalhada sobre o **XACML**, consultar o artigo [Ramli et al., 2014]

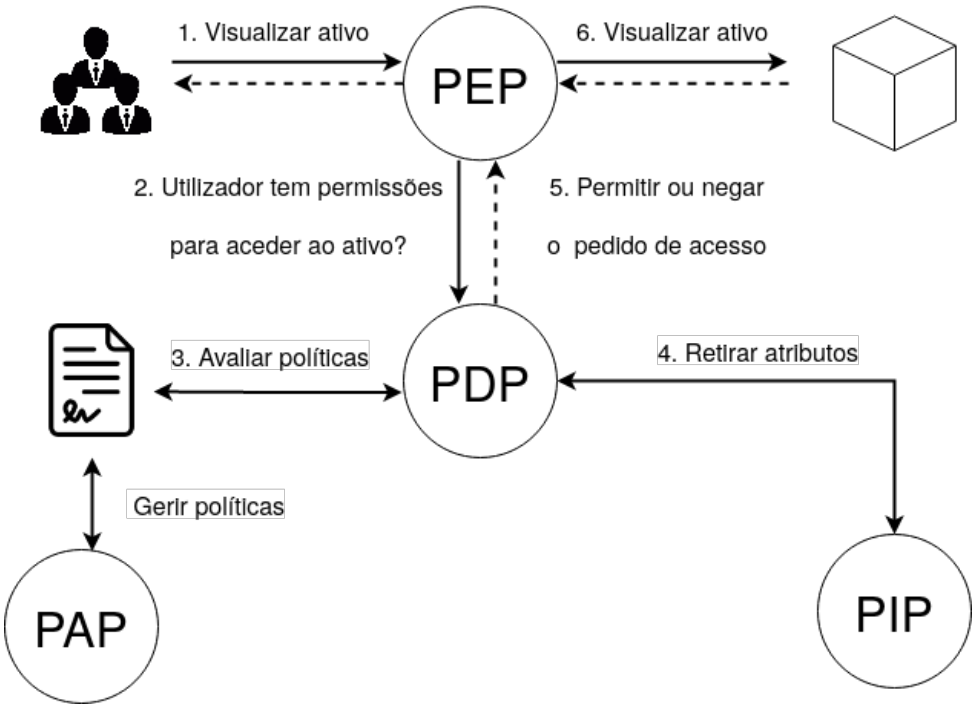


Figura 8: Arquitetura do XACML.

Como se pode observar na Figura 8, o **XACML** é composto por quatro elementos, o PEP, o PDP, o PAP e o PIP. O elemento PAP é responsável por escrever e gerir as políticas do sistema. O PEP recebe os pedidos de acesso dos clientes e reencaminha esses pedidos ao PDP na sua forma nativa, incluindo atributos, sujeitos, recursos, ações e ambiente. O PIP retira os atributos dos pedidos, para poderem ser analisados. O PDP é responsável por pedir os atributos dos pedidos ao PIP e por pedir a verificação das políticas relacionadas com o pedido ao PAP, após receber os resultados das outras componentes retorna o resultado do pedido de acesso ao PEP. Assim que o PEP tem uma resposta do PDP, fornece ao cliente consoante a resposta o acesso ou não acesso ao recurso requisitado. A utilização de **XACML** para resolver o problema de acesso é apresentado na Figura 9. Este exemplo começa com uma declaração de uma política, onde na linha número 8 é utilizado o elemento *<Target>* para restringir esta política a pedidos de acesso onde o valor do recurso é **Íman**. Dentro desta política, na linha número 18 é criada uma regra com o efeito *Permit*, que permite o acesso, caso a condição que especifica que o sujeito a realizar o pedido de acesso seja o **Bruno** seja satisfeita, como é especificado na linha número 28.

```
<!-- define a politica -->
<Policy>
    <!-- referencia os recursos a que as regras se aplicam-->
    <Target>
            <Resources>
                <ResourceMatch Matchid="equal">
                        <AttrValue>
                            Iman
                        <AttrValue>
                </ResourceMatch>
            </Resources>

            <Actions>
                <AnyAction/>
            </Actions>
    </Target>
    <!-- regra que permite o Bruno realizar qualquer ação ao recurso -->
    <Rule Effect="Permit">
            <Target>
                    <Subjects>
                            <Subject>
                                <SubjectMatch MatchId=
                                "urn:oasis:names:tc:xacml:1.0:function:
                                rfc822Name-match">
                                    <AttributeValue DataType=
                                        "http://www.w3.org/2001/
                                        XMLSchema#string">
                                            Bruno
                                    </AttributeValue>
                                </SubjectMatch>
                            </Subject>
                    <Subjects>
            </Target>
    </Rule>
</Policy>
```


Figura 9: Problema da Figura 5 resolvido com XACML.

2.3 Aprofundamento da gestão de acesso em HLF

Existem três áreas onde é possível gerir permissões quanto a **HLF**: na configuração do canal, acesso a dados e aprovação de transações.

Configuração do canal define quem pode ou não pode efetuar alterações ao ficheiro de configuração do **HLF**.

Acesso a dados define quem pode aceder a um dado recurso.

Aprovação de transações define quem precisa de aprovar uma transação.

Esta gestão de permissões é realizada em várias componentes do **HLF** conforme aqui se exemplifica.

(a) Uma rede **HLF** é constituída por várias organizações e cada participante tem de fazer parte de uma delas. As organizações geram certificados para os seus membros, que servem de identificação na rede e onde está referido a *função* que cada membro desempenha.

Uma organização pode estar em vários canais em simultâneo, tendo um *ledger* (e uma base de dados de estado) em separado para cada canal. Esta característica do **HLF** pode ser usada para uma gestão de acesso pouco granular.

Além disso, uma organização pode também querer partilhar conjuntos de informação diferentes com vários grupos de organizações, como está exemplificado no esquema da Figura 10. Uma alternativa é usar a funcionalidade de coleções privadas[6] do **HLF** que causa um efeito semelhante, que se detalha na Secção 2.3.1.

[6] Para mais informações sobre coleções privadas consultar a documentação do **HLF** https://hyperledger-fabric.readthedocs.io/en/release-2.2/private-data/private-data.html

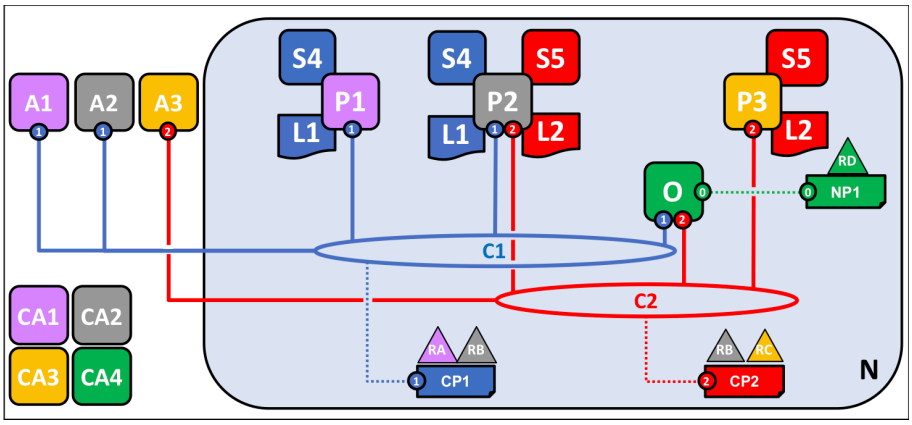


Figura 10: ex: O processo P1 da organização 1 pertence ao canal C1, que tem associado a ele o *chaincode* S4 e o *ledger* L1, o processo P3 da organização 3 pertence ao canal C2 que tem associado a ele o S5 e o L2, e o processo P2 da organização 2 pertence aos canais C1 e C2.

As coleções privadas e os vários canais numa rede são mecanismos associados à gestão de permissões de **Acesso a dados**.

(b) No ficheiro de configuração configtx.yaml na secção *organizations* são definidas todas as organizações que fazem parte do canal. Nesta definição referem-se as permissões gerais que cada organização tem relativamente aos direitos de Leitura, Escrita, Aprovação e Administração, podendo assim relacionar esta gestão de permissões à gestão de **Configuração de Canal**, **Acesso a dados** e **Aprovação de transações**.

(c) Na secção *Application* ainda do ficheiro configtx.yaml, na subsecção *Policies* (ou seja *Application.Policies*) são estabelecidas políticas por omissão e podem ser ainda definidas novas políticas com um nome. Estas políticas posteriormente podem ser utilizadas na secção *Application.ACL*, onde existe uma **ACL** que permite associar a recursos ou funcionalidades políticas de acesso.

A seguir, na Figura 11 é apresentado um exemplo, onde se especifica quem pode propor um *chaincode* para o canal. Neste caso para um processo poder propor um *chaincode(peer/Propose)* terá de cumprir a política */Channel/Application/Writers*. Esta relação é criada na linha número 4. Esta política está definida em *Application.Policies.Writers*, ou seja, na linha número 7. Esta política diz que qualquer processo que tenha o direito de escrita pode propor um *chaincode*. Este direito de escrita dos processos é estabelecido na definição da organização no canal, como foi explicado anteriormente no ponto b desta enumeração.

```
Application: &ApplicationDefaults
    ACLs: &ACLsDefault
                    # Política ACL para invocar chaincode nos peers
                    peer/Propose: /Channel/Application/Writers

Policies: &ApplicationDefaultPolicies
    Writers:
            Type: ImplicitMeta
            Rule: "ANY Writers"
```

Figura 11: Excerto de um ficheiro configtx.yaml

Esta **ACL** permitem gerir permissões ao nível de **Configuração do Canal** e **Acesso a dados**.

(d) Outro mecanismo de gestão de permissões de **Acesso a dados** no **HLF** é a sua implementação no próprio *chaincode(smart contract)*, verificando quem está executar o *chaincode* e restringindo ou dando acesso dependendo dos atributos da entidade em questão, apresentados no certificado associado à entidade, utilizando, por exemplo, a biblioteca CID[7].

2.3.1 Coleções Privadas

A definição de coleções privadas [8] permite a especificação de permissões de acesso a dados de forma mais granular que a definição de canais.

Nesta secção apresenta-se a gestão de ativos entre organizações como caso de uso de coleções privadas no **HLF**. Para tal, foi criado um canal com duas organizações A e B, e um processo *peer* por organização. A cada organização associa-se uma coleção privada a que apenas a própria tem acesso (*Private Data Collection A* e *Private Data Collection B*), e uma coleção a que ambas as organizações têm acesso (*Private Data Collection*), conforme representado na Figura 12.

O acesso à coleção privada é definida no ficheiro de configuração da Figura 13 por uma série de políticas. Foram definidas então as seguintes coleções:

[7] CID: Client Identity Chaincode Library https://github.com/hyperledger/fabric-chaincode-go/tree/main/pkg/cid

[8] Para um tutorial de como usar coleções privadas no **HLF** https://https://hyperledger-fabric.readthedocs.io/en/latest/private_data_tutorial.html

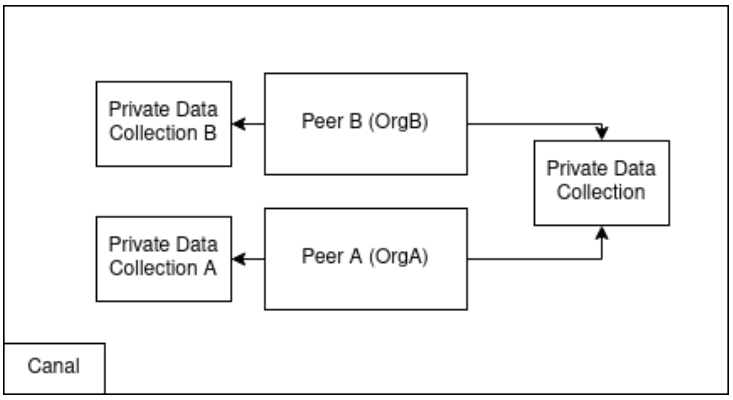


Figura 12: Arquitetura: canal, organizações e coleções privadas.

assetCollection define que ambas as organizações podem manter a coleção privada nos seus processos. As propriedades "memberOnlyRead"e "memberOnlyWrite"especificam que apenas processos da orgA e orgB podem ler e escrever na coleção (linhas número 4-11).

Org1MSPPrivateCollection define que a coleção só pode estar em processos da orgA e define que cada atualização na coleção tem de ser aprovada pela orgA (linhas número 11-21).

Org2MSPPrivateCollection define que a coleção só pode estar em processos da orgB e define que cada atualização na coleção tem de ser aprovada pela orgB (linhas número 21-31).

Definiu-se o ativo como tendo as seguintes propriedades *type, id, cor, size, owner e appraisedvalue*. Para a implementação dividiu-se o ativo em duas estruturas. A coleção comum às duas organizações guarda os campos *type, id, cor, size, owner* e cada coleção de cada organização vai guardar *id e appraisedvalue*, como se pode observar na Figura 14.

A seguir, como se pode constatar no diagrama da Figura 15, (1.) criou-se um ativo da orgA e (2.) foi inserido na coleção privada comum e (3.) na coleção da orgA onde terá o preço associado. Desta forma todos os participantes do canal conseguem (6. e 7.) observar as propriedades do ativo, mas apenas os processos da orgA conseguem aceder ao seu preço. Se a organização B quiser comprar o ativo terá de (8.) submeter uma proposta de preço à organização A por aquele ativo e se o valor for igual ou superior ao *appraisedvalue* que apenas a orgA tem acesso, a orgA (9.) pode aprovar aquela transferência do ativo, criando assim na coleção privada da orgB um (11.) registo do ativo, (10.) apagando-a da coleção da orgA e (12.) atualizando a propriedade *owner* na coleção pública do ativo. Por fim (13.) a orgA notifica a orgB da transação efetuada com sucesso.

Como demonstrado neste exemplo, o **HLF** já oferece mecanismos para diferenciar o acesso a recursos a participantes de diferentes organizações. Neste caso utilizaram-se coleções privadas, mas poderia ser

```
// collections_config.json
[
   {
   "name": "assetCollection",
   "policy": "OR('Org1MSP.member', 'Org2MSP.member')",
   "requiredPeerCount": 1,
   "maxPeerCount": 1,
   "blockToLive":1000000,
   "memberOnlyRead": true,
   "memberOnlyWrite": true
   },{
   "name": "Org1MSPPrivateCollection",
   "policy": "OR('Org1MSP.member')",
   "requiredPeerCount": 0,
   "maxPeerCount": 1,
   "blockToLive":3,
   "memberOnlyRead": true,
   "memberOnlyWrite": false,
   "endorsementPolicy": {
       "signaturePolicy": "OR('Org1MSP.member')"
   }},{
   "name": "Org2MSPPrivateCollection",
   "policy": "OR('Org2MSP.member')",
   "requiredPeerCount": 0,
   "maxPeerCount": 1,
   "blockToLive":3,
   "memberOnlyRead": true,
   "memberOnlyWrite": false,
   "endorsementPolicy": {
       "signaturePolicy": "OR('Org2MSP.member')"
   }}
]
```

Figura 13: Definição de estruturas a ser guardadas em cada coleção privada.

```
type Asset struct {
       Type  string `json:"objectType"`
       ID    string `json:"assetID"`
       Color string `json:"color"`
       Size  int    `json:"size"`
       Owner string `json:"owner"`
}

type AssetPrivateDetails struct {
       ID             string `json:"assetID"`
       AppraisedValue int    `json:"appraisedValue"`
}

type AssetPrivateDetails struct {
       ID             string `json:"assetID"`
       AppraisedValue int    `json:"appraisedValue"`
}
```

Figura 14: Definição de estruturas a ser guardadas em cada coleção privada.

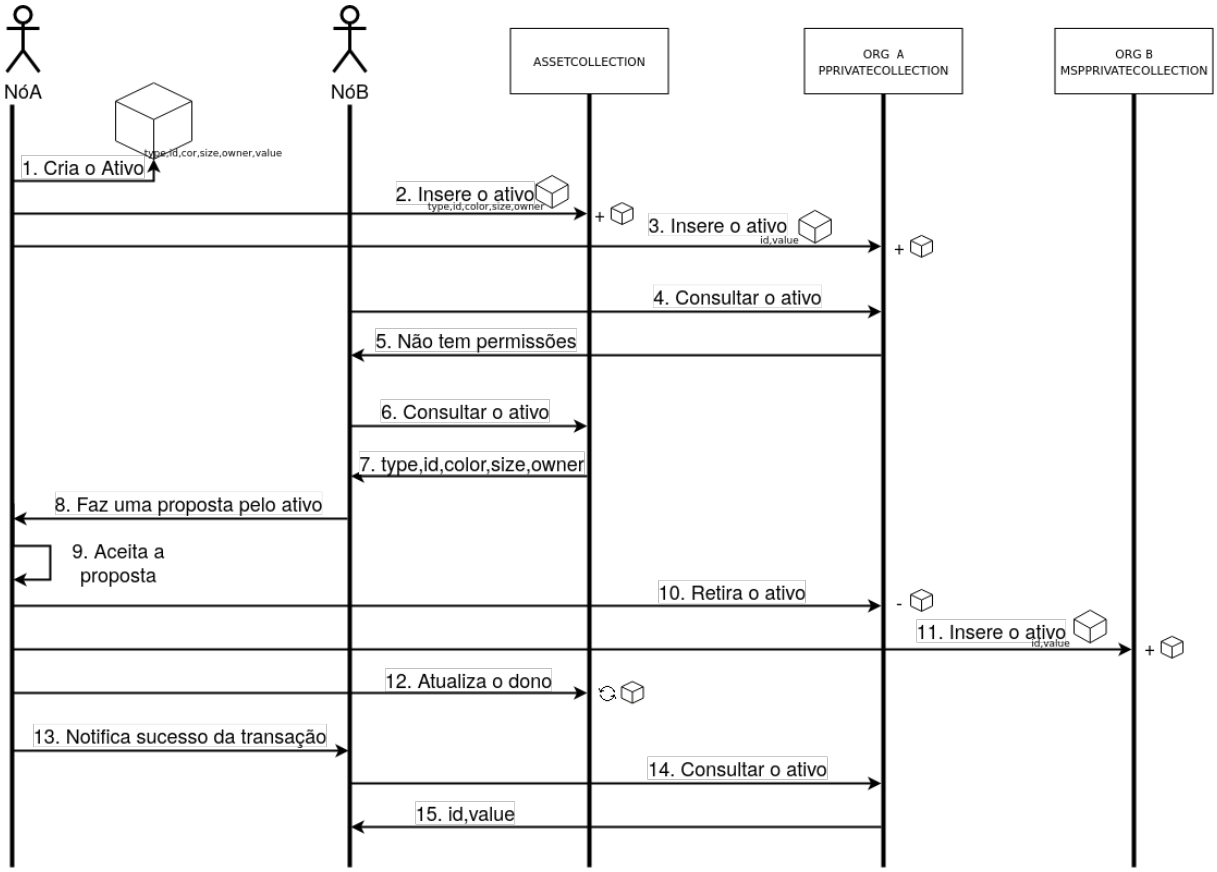


Figura 15: Criação de um ativo e uma transação do mesmo.

realizado algo semelhante usando vários canais. No entanto, a utilização de múltiplos canais para este efeito não se adequa a alterações dinâmicas nas permissões.

Em alguns cenários pode ser relevante diferenciar acesso dentro de uma organização, por exemplo, no caso de uma organização representar uma empresa, esta empresa poderá ter vários departamentos e/ou diferentes estatutos que impliquem diferentes graus de acesso aos dados da empresa. No pior caso, poderíamos chegar a ter uma coleção por cada departamento para cada organização.

Adicionalmente, querendo efetuar transformações mais complexas do que apenas omitir ou não omitir dados, a única possibilidade seria implementar as transformações e, opcionalmente a lógica de diferenciação de acesso associada no próprio*chaincode*. Sendo o *chaincode* escrito numa linguagem de alto nível, onde não existe uma estrutura que se têm de seguir fica assim ao cargo de quem o desenvolve colocar esta lógica complexa: podem ser múltiplas transformações, dependendo de quem é o dono do recurso e quem o acede, à responsabilidade dos administradores da rede, o que não é nada trivial, e é muito provável suscitar erros. Além disso, as políticas de acesso aos dados evoluem com o tempo, sendo que mudanças nas transformações ou na política de acesso obrigariam a reinstalar o *chaincode* em todos os processos (*peers*) da rede e potencialmente o acordo de processos (*peers*) de múltiplas orga-

nizações. Este mecanismo não é compatível com dinamismo. Com estes possíveis cenários de utilização encontram-se as limitações do **HLF**, a definir políticas de acesso e a definir transformações dos dados.

2.3.2 XACML no HyperLedger Fabric

Como apresentado em 2.2.1, o **XACML** permite exprimir políticas de acesso complexas e estruturadas. Como exercício de melhor definição da capacidade de expressar políticas de permissões em **HLF**, traduzem-se aqui em **XACML**.

O **HLF** tem essencialmente duas categorias de políticas: políticas de assinatura e políticas implícitas.

Políticas de assinatura

Políticas de assinatura são políticas de baixo nível e especificam as funções particulares de processos que podem aceder a um dado recurso. São normalmente usadas na definição de organizações e usam operadores básicos como *AND* e *OR*. Um exemplo de uma política de assinatura é *OR('OrgA.admin')*. Na Figura 16 pode-se observar como *OR('OrgA.admin,OrgB.admin')* em **HLF** seria representado em **XACML**, em pseudo-código.

Neste exemplo, como é usado um *OR*, na representação em **XACML** usa-se a regra de combinação *permit-overrides* (linha número 1), que valida a política se pelo menos uma regra for cumprida. Se fosse um *AND*, ter-se-ia que usar a regra de combinação *deny-overrides* que verifica que todas as regras são cumpridas, se não acontecer a política não é cumprida.

Políticas implícitas

Políticas implícitas são políticas mais complexas definidas através de outras políticas agregadas por operadores como *ANY, ALL, MAJORITY*. A Figura 17 ilustra uma política definida no **HLF** utilizando o operador *MAJORITY*, que especifica que para a política ser válida a maioria das políticas dos *Admins* tem de ser respeitadas, linha número 5.

Representar em **XACML** os *ANY, ALL* é algo trivial já que as regras de combinação oferecem isso, respetivamente *permit-overrides* e *deny-overrides*. Bastaria criar uma regra para cada organização e depois combiná-las com a devida regra. Representar *MAJORITY* já não é tão fácil, pois não existem regras de combinação que ofereçam esse operador. Para representar operadores como *MAJORITY* no **XACML** é preciso construir uma política mais complexa que tira partido de funções como *map, anyof, count, ...* .

```
<policy rule-combining: permit-overrides>
    <rule>
        <target>
            <subjects>
                    <subject match OrgA.admin/>
            </subjects>
    </rule>
    <rule>
        <target>
            <subjects>
                    <subject match OrgB.admin/>
            </subjects>
   </rule>
</policy>
```

Figura 16: Política de assinatura representada em XACML.

```
Application:
        Policies:
                 Admins:
                  Type: ImplictMeta
                  Policy: "MAJORITY Admins"
```

Figura 17: Política implícita representada em XACML.

Capítulo 3

Smart Data Access Management

Este capitulo introduz a aplicação **SDAM** que confere a possibilidade de aumentar a flexibilidade da gestão de permissões no sistema **HLF** de uma forma transparente. Descreve-se a abordagem seguida, detalhando o funcionamento da aplicação desenvolvida, seus componentes e como é conseguida a integração da mesma com o **HLF**.

3.1 Modelo adotado

O **SDAM** configura um sistema de middleware, intercetando todos os pedidos trocados entre o **HLF** e a respetiva instância de base de dados. A figura 18 destaca o esquema lógico desta abordagem, na qual se realça a composição interna do **SDAM**. Este apresenta um componente PDP e um componente PEP, ambos seguindo o mesmo papel que na arquitetura apresentada em **XACML**. Apresenta-se uma descrição das responsabilidades inerentes a cada componente:

PEP recebe pedidos de acesso com um conjunto de atributos relevantes para a decisão de acesso. O PDP valida o pedido de acesso ao recurso, devolvendo uma resposta ao cliente segundo a resposta do PDP. Além de filtrar a informação da resposta, podem ser aplicadas outras transformações aos dados.

PDP avalia as políticas e processa uma resposta ao PEP. Esta decisão tem em conta vários atributos, por exemplo, quem pede o acesso, as suas características ou seu propósito. A decisão tem que

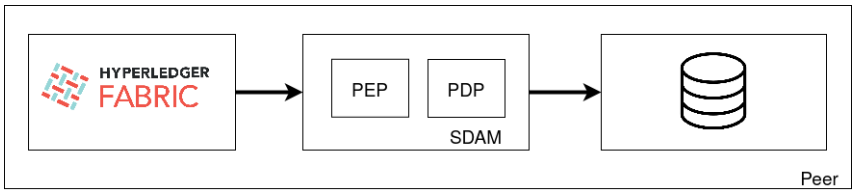


Figura 18: Esquema com a abordagem escolhida.

respeitar os parâmetros definidos pelo sujeito que inseriu o recurso.

A estruturação e definição de permissões desencadeada pelo **SDAM** permite atingir o objetivo de sistematização do processo. Permite também maior flexibilidade por parte das instituições tomadoras, já que evita decisões binárias (acesso ou não acesso) sobre o acesso a dados. Obtém-se assim a capacidade de disponibilizar ou omitir atributos dependendo de quem tenta aceder ao recurso. Tal facilidade não implica qualquer alteração no estado disponível na base de dados, nem interfere com o processo de validação de transações, sendo assim transparente ao **HLF**.

Nesta abordagem considera-se que a base de dados é protegida por uma *password*. No entanto, a base de dados executa no domínio administrativo de uma organização, e portanto, a *password* não configura proteção suficiente para garantir a segurança dos dados, nem que não existe acesso indevido.

A forma mais direta de mitigar esta situação é considerar que os dados pertencentes a outras organizações se encontram cifrados nas bases de dados de terceiros, sendo a disponibilização dos dados em texto limpo e a sua cifragem (determinística) assegurada pelo **SDAM** sempre que necessário, como mais um tipo de transformação de dados. A componente de gestão de chaves deixa-se como trabalho futuro.

É de notar que independentemente de considerar esta mitigação para dados de outras organizações, o controlo de acesso intra-organização é extremamente relevante. Neste aspecto, com **SDAM**, torna-se possível restringir o acesso intra-organização de acordo com o processo de negócio a desenvolver. Evita-se assim, por exemplo, a situação em que um funcionário acede de forma espúria a informação sensível de terceiros.

Em relação à confiança nos valores dos atributos dos pedidos, apresentados ao **SDAM** para aplicação das políticas, esta é delegada na infraestrutura de gestão de certificados da organização de origem do pedido e pela autenticação externa da aplicação que interage com o sistema e que injeta estes atributos no pedido.

Cada organização define as políticas de acesso a dados que se aplicam aos seus documentos, independentemente do *peer* ou organização através da qual chega o pedido de acesso. Assim, e de forma a garantir o determinismo no acesso a dados essencial para o mecanismo transacional, as várias instâncias do módulo de políticas (pelo menos uma por organização) deverão ter uma visão coerente das políticas ativas. Esta pode ser conseguida através de um mecanismo leve de propagação de alterações com um período de aplicação conhecido, cuja implementação também se deixa como trabalho futuro.

3.2 Arquitetura e funcionamento do SDAM

O **SDAM** serve como proxy entre o **HLF** e a respetiva base de dados CouchDB, permitindo transformações aos dados, bem como a especificação de políticas de acesso mais flexíveis. O **SDAM** adota a API REST do *CouchDB*, não impondo qualquer alteração no **HLF** para a sua utilização.

O **SDAM** implementa dois módulos:

1. O módulo proxy, serve de *interface* para os *peers* do **HLF**, executa *queries* sobre a base de dados e serve como PEP, aplicando decisões de acesso emitidas pelo módulo de políticas.

2. O módulo de políticas, armazena as políticas de controlo de acesso e emite decisões sobre pedidos de autorização, servindo como PAP e PDP.

O módulo proxy é o módulo central do **SDAM**. Este expõe a API REST do *CouchDB*, permitindo uma integração direta com o **HLF**, sem requerer alterações.

Como exemplo, obter um documento através do seu *id* pode ser conseguido através do método *GET /db/docid* da API REST do *CouchDB* que corresponde a *GET http://url/basedados/documentoID*.

O **SDAM** suporta todas as chamadas à API usadas pelo **HLF**. Estas chamadas carregam informação adicional no corpo do pedido(e.g., a1,b1 e b2), nomeadamente atributos expressados em *JSON*, podendo estar relacionados com a identidade, motivo, etc. Estes atributos são injetados por lógica de aplicação, tornando este processo transparente aos *peers* do **HLF**.

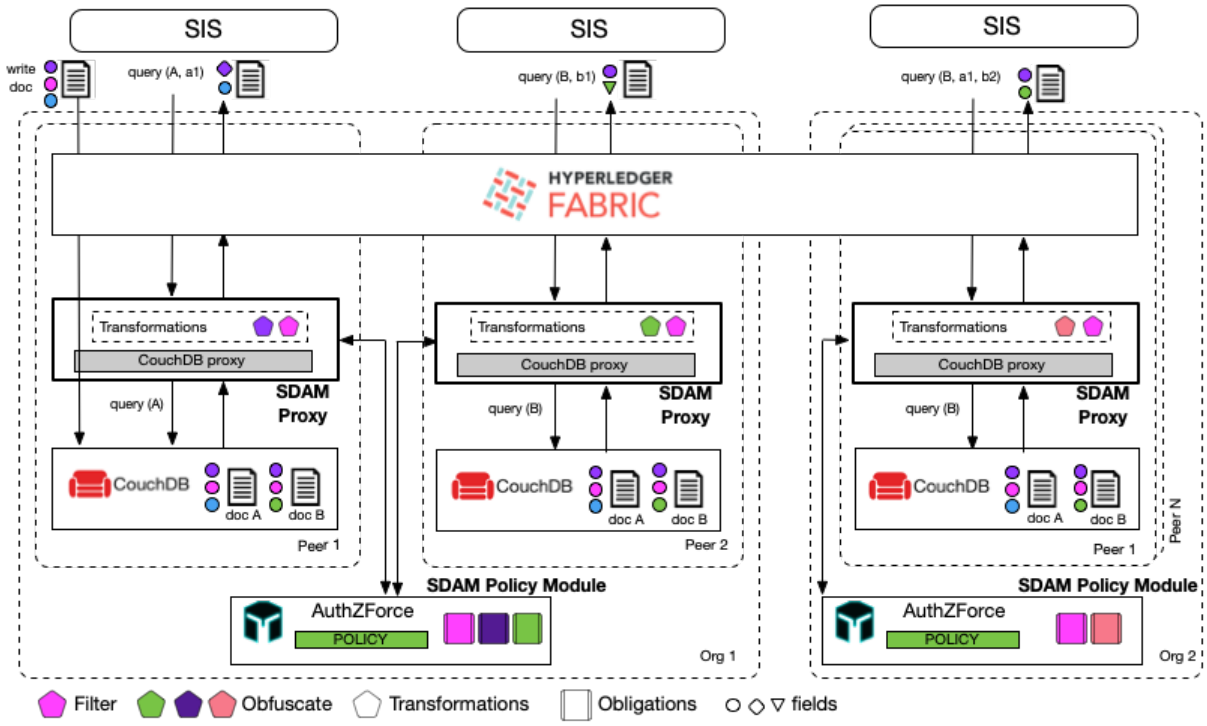


Figura 19: Arquitetura do HLF. Os pentágonos representam as implementações de transformações disponíveis e os quadrados representam as obrigações definidas no Módulo de Políticas.

Quando um pedido é recebido pelo proxy (e.g., query(A,a1)), os atributos são extraídos antes de reencaminhar o pedido (query(A)) para o *CouchDB*. É criado um pedido **XACML**, com base nestes atributos e com atributos extraídos do documento obtido do *CouchDB* (i.e. atributos que identificam o documento, propriedades do documento, etc). O pedido **XACML** é enviado para o módulo de políticas que responde com um *Permit* ou um *Deny*, além de expressões opcionais (*ObligationExpression*) ou transformações obrigatórias a executar sobre o documento antes de retornar ao *peer*.

Uma *obligation* requer que a propriedade *JobInfo* e qualquer sub-campo seja removido do documento, caso a intenção do pedido for *PROCESS_POLICY*.

Atualmente são suportadas duas transformações. A transformação *HIDE* (pentágono cor-de-rosa) remove um campo específico do documento (o círculo correspondente desaparece - representado na query(A,a1)), e *OBFUSCATE* (pentágonos de cor verde e roxo), que em vez de remover campos, aplica outras transformações. A transformação *OBFUSCATE* em vez de esconder o valor, substitui-o por um que ainda pode ser usado para algumas operações, criando assim um *hash code* (círculo roxo torna-se um diamante - representado na query(A,a1)), ou substituindo um valor numérico por outro de menor precisão, e.g., substituir idade por grupo de idade (círculo verde transforma-se num triângulo - representado na

query(B,b1)).

Adicionar novas transformações requer a implementação de um método que opera num *ObjectNode* e um construtor que recebe a correspondente *ObligationExpression*. Pedidos de escrita e resposta são reencaminhados pelo proxy e sujeitos apenas aos controlos de acesso nativos do **HLF**. Este módulo está implementado usando *Spring Reactive Framework*.

O módulo de políticas gere e armazena as políticas de controlo de acesso escritas em **XACML** e toma decisões sobre pedidos de acesso. Neste módulo, utiliza-se o PDP *AuthZForce*, sem qualquer modificação. Para a maioria dos cenários, a configuração ideal instância um módulo de políticas por organização, permitindo que sejam geridas a um nível organizacional.

3.3 SDAM e componentes

3.3.1 Módulo proxy (PEP)

A componente PEP é responsável por receber pedidos dos clientes, reencaminhar esses pedidos para a componente PDP e devolver a resposta ao cliente. Esta componente no **SDAM** foi implementada como um proxy que interceta os pedidos por parte do *chaincode* à base de dados e mapeia esses pedidos para o método associado, por forma que o PDP possa avaliar o pedido.

PIP

A componente PIP é a componente do **XACML** responsável por armazenar informações sobre os utilizadores. Nesta configuração todos os atributos relacionados com o pedido e o utilizador estão presentes no corpo do pedido http.

Rotas

Foram implementadas três rotas do *CouchDB*, que tornam possível o uso do **SDAM**. Nestas rotas o corpo dos pedidos esperado é exatamente o mesmo que o *CouchDB* espera, acrescido de uma chave onde tem os atributos do pedido. O **SDAM** ao receber estes pedidos retira os atributos adicionados, ficando assim só com o corpo esperado pelo *CouchDB* e submetendo o pedido.

Foram implementadas as rotas seguintes.

(a) *GET db/docid* : Que fornecendo o nome da base de dados e o *id* do ativo, devolve o ativo.

    (a) API do *CouchDB* (figura 20)

```
GET /movies/Contagion HTTP/1.1
Accept: application/json
Host: localhost:5984
```

Figura 20: Corpo de um pedido *GET db/docid* à base de dados *CouchDB*.

(b) API do **SDAM** (figura 21)

```
GET /movies/Contagion HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: localhost:8080
{
   "sdam_options": [
       {
           "category": "subject",
           "attributeID": "subject:group",
           "datatype": "http://www.w3.org/2001/XMLSchema#string",
           "value": ["admin"]
       }
   ]
}
```

Figura 21: Corpo de um pedido *GET db/docid* ao SDAM.

O pedido ao ser enviado ao **SDAM** requer que o JSON inclua no seu corpo o campo *sdam_options*, definido por um array de atributos caracterizadores do pedido. Neste caso, o único atributo existente configura uma categoria *subject* de id *subject:group* do tipo string e com o valor *admin*.

(b) *POST /db/_find* : Permite executar uma *query* sobre a base de dados.

(a) API do *CouchDB* (figura 22)

```
POST /movies/_find HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: localhost:5984
{
    "selector": {
        "year": {"$gt": 2010}
    },
    "fields": ["_id", "_rev", "year", "title"],
    "sort": [{"year": "asc"}],
    "limit": 2,
    "skip": 0,
    "execution_stats": true
}
```

Figura 22: Corpo de um pedido *POST /db/_find* à base da dados *CouchDB*.

(b) API do **SDAM** (figura 23)

```
GET /movies/Contagion HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: localhost:8080
{
    "sdam_options": [
            {
                "category": "subject",
                "attributeID": "subject:group",
                "datatype": "http://www.w3.org/2001/XMLSchema#string",
                "value": ["admin"]
            }
        ]
    ,
    "selector": {
            "year": {"$gt": 2010}
        },
    "fields": ["_id", "_rev", "year", "title"],
    "sort": [{"year": "asc"}],
    "limit": 2,
    "skip": 0,
    "execution_stats": true

}
```

Figura 23: Corpo de um pedido *POST /db/_find* ao SDAM.

O corpo inclui um objeto *JSON*, com uma chave *sdam_options* e a restante *query* do *CouchDB*.

(c) *POST /db/_bulk_get* : Permite aceder a vários ativos ao mesmo tempo.

(a) API do *CouchDB* (figura 24)

```
POST /{db}/\_bulk\_get HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: localhost:5984
{
    "docs": [
        {
            "id": "Contagion",
        }
        {
            "id": "TheConjuring",
        }
    ]
}
```

Figura 24: Corpo de um pedido *POST /db/_bulk_get* à base de dados *CouchDB*.

(b) API do **SDAM** (figura 25)

```
POST /{db}/\_bulk\_get HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: localhost:8080
{
    "sdam_options": [
            {
                "category": "subject",
                "attributeID": "subject:group",
                "datatype": "http://www.w3.org/2001/XMLSchema#string",
                "value": ["admin"]
            }
    ],
    "docs": [
        {
            "id": "Contagion",
        }
        {
            "id": "TheConjuring",
        }
    ]
}
```

Figura 25: Corpo de um pedido *POST /db/_bulk_get* ao SDAM

O corpo do pedido inclui um objeto *JSON*, com uma chave *sdam_options*, sendo os ativos pedidos colocados na chave *docs*.

3.3.2 Módulo de políticas (PDP)

A figura 18 apresenta a componente PDP, responsável pelo processo de avaliação de políticas, excluindo a responsabilidade de retirar os atributos dos pedidos e o armazenamento de todas as políticas. A componente PDP gere o fluxo da execução dessas tarefas e toma a decisão final de aplicação de transformações.

PRP e PAP

As componentes PRP e PAP do **XACML** são responsáveis por armazenar e gerir as políticas de acesso. Como referido na secção 3.5, foram analisados alguns sistemas como *WSO2 Balana* e *AuthZForce*, optando-se pelo sistema *AuthZForce* que apresenta uma API REST simples, desempenhando o papel de PAP e PRP.

3.3.3 Transformações

As transformações são operações implementadas no **SDAM** como classes, que obtém a resposta do *AuthZForce* contendo a *obligaton* e o recurso que o *CouchDB* fornece. Com esses dados transformam o recurso, consoante a política definida por quem o inseriu.

Transformações Implementadas

Disponibilizam-se as transformações *HIDE*, que permite omitir um determinado campo recebendo uma *String* como argumento que especifica o caminho até ao atributo em questão, e a transformação *OBFUSCATE*, que permite ofuscar um determinado campo e que dispõe de um argumento com um objetivo análogo ao argumento da transformação *HIDE*. A figura 28 apresenta um exemplo da representação em **XACML** da transformação *HIDE*.

Adicionar Transformações

Por forma a adicionar uma nova transformação é necessário apenas criar uma classe, correspondendo ao *ObligationID* que implementa a *interface Transformation*. A figura 26 apresenta um exemplo da resposta do *AuthZForce*. Destaca-se o caso de transformações com múltiplos argumentos, onde a chave *AttributeAssignment*(linha número 3) é um *array*.

```
{
    "@ObligationId": "HIDE",
    "AttributeAssignment": {
            "@AttributeId": "arg",
            "@DataType": "http://www.w3.org/2001/XMLSchema#string",
            "$": "/name"
    }
}
```

Figura 26: Excerto da resposta do AuthZForce a um pedido de acesso, onde se identifica a transformação a realizar.

3.3.4 Integração do SDAM no Sistema

Uma das vantagens do **SDAM** é a forma transparente como é integrado no funcionamento do **HLF**. A sua integração é conseguida pela alteração do endereço IP e porto que o **HLF** tem como sendo a base de dados *CouchDB*, passando a usar o endereço IP e porto onde o **SDAM** está em execução.

3.4 Definição de Políticas

A linguagem **XACML** apresenta regras bem definidas relativamente à estrutura das suas mensagens e uma documentação rica, permitindo descrever novas políticas.

As linguagens de gestão de acesso permitem definir políticas para regular o acesso a recursos, criando assim uma relação entre utilizadores e recursos. Neste caso, existe também o objetivo de conseguir especificar nas políticas as transformações que o recurso terá que sofrer. Para isso, analisou-se a estrutura das políticas em **XACML** com o intuito de encontrar algum elemento que possa representar essa transformação.

3.4.1 Obligation

O elemento *Obligation* foi criado com o objetivo de especificar verificações necessárias a fazer antes da reposta ser validada. Um exemplo é uma política que define um acesso a um registo médico de um paciente por um médico, onde se quer que o acesso seja dado, apenas se for enviada uma notificação

por email ao paciente. Para tal, o recurso pode ser identificado através de um id. Esta será a operação de notificação, associada ao elemento *Obligation*, que a componente PEP terá de implementar e executar com sucesso antes da resposta ao pedido de acesso ser validada. Aproveitando a componente *Obligation*, consegue-se identificar a transformação a realizar, caso essa política autorize o acesso. Em alternativa à componente *Obligation*, foi empregue a componente *ObligationExpression*, que disponibiliza uma versão melhorada da primeira e que oferece mais possibilidades.

Documentação

Como definido na documentação (figura 27), a *ObligationExpression* contém um atributo, *ObligationID* (linha número 7), podendo ter zero ou mais ocorrências de elementos do tipo *AttributeAssignment* (linhas número 4-5).

```
1 <xs:element name="ObligationExpression" type="xacml:ObligationExpressionType"/>
2 <xs:complexType name="ObligationExpressionType">
3   <xs:sequence>
4     <xs:element ref="xacml:AttributeAssignmentExpression" minOccurs="0"
5           maxOccurs="unbounded"/>
6   </xs:sequence>
7   <xs:attribute name="ObligationId" type="xs:anyURI" use="required"/>
8   <xs:attribute name="FulfillOn" type="xacml:EffectType" use="required"/>
9 </xs:complexType>
```

Figura 27: Definição do elemento *ObligationExpression*.

Exemplo da especificação de uma transformação

A representação de uma transformação é conseguida através do campo *ObligationID*, que identifica a transformação a ser aplicada. O campo *AttributeAssignment* define os argumentos necessários à transformação. Interpretando o exemplo ilustrado na figura 28, a transformação *HIDE* (linha número 2) recebe um argumento */name* (linha número 6), que será ocultado.

```
<ObligationExpressions>
    <ObligationExpression ObligationId="HIDE" FulfillOn="Permit">
        <AttributeAssignmentExpression AttributeId="arg">
            <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#string">
                /name
            </AttributeValue>
        </AttributeAssignmentExpression>
    </ObligationExpression>
</ObligationExpressions>
```

Figura 28: Especificação de uma transformação numa *ObligationExpression*

3.5 Armazenamento de Políticas

As políticas são persistidas recorrendo a um módulo de políticas que permita o armazenamento e avaliação de pedidos de acesso com políticas armazenadas escritas em **XACML**. Dos sistemas encontrados, destacou-se foi o *AuthZForce*. Também foi analisado o sistema *WSO2 Balana*, apresentado um maior número de funcionalidades cujo valor acrescentado ao **SDAM** é limitado.

3.5.1 AuthZForce

O *AuthZForce* é uma framework que segue o padrão **XACML** *OASIS XACML standard v3.0* para a gestão de permissões.

A API REST do *AuthZForce* oferece várias rotas que podem ser identificadas com um de três tipos: Domínios, Políticas, Decisão de Acesso. No *AuthZForce* existe o conceito de domínio. Cada domínio contém um conjunto de propriedades associadas, como por exemplo a identificação da política *root*, caminho onde é iniciada a avaliação dos pedidos. Além disso, dispõe também de um conjunto de políticas às quais cada pedido de acesso tem de ser associado.

Estão definidas a seguintes rotas básicas de encaminhamento.

(a) Domínios: operações CRUD sobre domínios.

```
POST /domains
```

```
 # Adicionar domínio
GET /domains/:domainId/properties
 # Obter domínio
PUT /domains/:domainId/properties
 # Atualizar domínio
DELETE /domains/:domainId
 # Eliminar domínio
```

(b) Políticas: operações CRUD sobre as políticas.

```
POST /domains/:domainId/pap/policies
 # Adicionar/Atualizar política
GET  /domains/:domainId/pap/policies
 # Obter todas as políticas
GET /domains/:domainId/pap/policies/:policy
 # Obter versões de uma política
GET /domains/:domainId/pap/policies/:policy/:version
 # Obter versão de uma política
DELETE /domains/:domainId/pap/policies/:policy
 # Eliminar uma política
DELETE /domains/:domainId/pap/policies/:policy/:version
 # Eliminar uma versão de uma política
```

(c) Decisão de Acesso: avaliar acesso e propriedades do PDP.

```
POST /domains/:domainID/pdp
 # Executar um pedido de Acesso
GET /domains/:domainId/pap/pdp.properties
 # Obter as propriedades.
PUT /domains/:domainId/pap/pdp.properties
 # Atualizar as propriedades.
```

3.6 Deployment do SDAM

A implementação desta abordagem requer a instanciação do **SDAM**, inicializando as componentes PDP e PEP. Para inicializar a componente PEP, basta executar a aplicação JAVA, alterando nas suas configurações a referência da base de dados que mais tarde vai ser ligada. No caso da componente PDP é necessário inicializar o módulo de políticas *AuthZForce* e realizar um pedido *http* para adicionar um domínio. A partir desse momento é possível inserir políticas no módulo de políticas. Com o **SDAM** a ser executado é possível iniciar o **HLF** e a base de dados, sendo necessário configurar o **HLF**, por forma que a visão da base de dados seja tomada pelo **SDAM**. É relevante realçar que como o módulo de políticas não depende das outras componentes do sistema, é possível atualizar as políticas de acesso a serem utilizadas na rede sem necessidade de reiniciar o sistema ou causar constrangimentos na rede.

3.7 Intenções, Blocos, Documentos e Consentimentos

Com o desenvolvimento do projeto surgiram alguns conceitos relevantes, nomeadamente as intenções, os blocos, os documentos e os consentimentos. Estes conceitos encontram-se relacionados de uma forma estrita, balizando o contexto (i.e. a intenção) no qual um acesso a um determinado bloco de um documento é executado.

3.7.1 Definição de Intenção

A intenção configura o contexto ou motivo no qual um determinado bloco de um documento é acedido. A intenção materializa-se num identificador que representa esse pedido. As intenções existentes são guardadas num *smart contract* próprio na rede. Aquando de um pedido de acesso ao chaincode, este necessita incluir a intenção que o contextualiza, permitindo que o **SDAM** opere as transformações necessárias

Biblioteca Shim

Os métodos dentro de um chaincode utilizam um *Stub*, fornecido por uma biblioteca *Shim* do HLF, que fornece os métodos necessários para executar leituras e escritas na rede *blockchain*. Aproveitando o facto de esta biblioteca ser necessária e transparente na utilização do HLF, decidiu-se criar uma biblioteca para uso com o **SDAM** que fornece este *Stub* e, além disso, insere nos pedidos de leituras e escritas as intenções como atributos dos pedidos.

3.7.2 Definição de Documento

Um Documento configura os recursos a serem guardados no *blockchain*. Cada documento é definido por uma estrutura flexível, facilitadora da definição de consentimentos. Cada documento é caracterizado pela seguinte estrutura.

Tipo Um identificador do tipo de documento.

Blocos Lista de blocos com a informação neles contida.

Utilizadores Lista com utilizadores com acesso ao documento.

3.7.3 Definição de Blocos

O conceito de Bloco configura uma secção de um documento, incluindo um conjunto de propriedades que lhe estão associadas. Os blocos existentes são guardados em *smart contracts* próprios na rede.

3.7.4 Definição de Consentimento

O consentimento permite a definição por parte do proprietário da informação, dos utilizadores que podem obter acesso. Permite também a definição das transformações, eventualmente necessárias a aplicar em função das respetivas intenções. Este conceito é definido em 5 propriedades.

Quem O utilizador de qual a informação é referente.

Para quê Identificação da informação através de uma lista de pares, onde o primeiro elemento é o identificador de um bloco e o segundo elemento as transformações a realizar caso esse bloco esteja presente no documento.

Com quem Identificação dos utilizadores com quem é autorizado ser partilhado.

Em que situação Identificação das intenções, em que o consentimento é válido.

Quando/Durante Por quanto tempo é válido.

Exemplo

Um exemplo de um consentimento em linguagem corrente é definido como:

```
"Eu, utilizador x, dou consentimento a que acedam à minha morada,
exceto o número da porta e à minha  informação de nascimento,
exceto a localidade para as intenções [F1,F2,F3],
a clientes pertencentes à organização A, a partir de hoje."
```

Figura 29: Exemplo de um consentimento.

Da descrição em linguagem natura ilustrada na figura 29 (assumindo que a morada faz parte do Bloco1 e a informação de nascimento do Bloco2), deriva-se a seguinte desconstrução nas 5 propriedades:

Quem utilizador x

Para quê [(Bloco1,[Hide("NúmeroDaPorta")), (Bloco2,[Hide("Localidade")]]

Com quem [organização A]

Em que situação [F1, F2, F3]

Quando/Durante 15 de novembro de 2022- ∞

3.7.5 Armazenamento dos consentimentos

Uma forma alternativa de descrever os consentimentos, é descrevê-los como uma estrutura padrão para definir políticas. Com esta estrutura definida é importante pensar onde estas serão guardadas e colocadas em prática. Na secção 3 onde se abordaram temas como definição de políticas e armazenamento, foi encontrada a framework *AuthZForce* que fornecia a capacidade de armazenar políticas e de realizar pedidos de acesso sobre elas, para tentar se reutilizar esta framework é necessário compreender se é possível representar o consentimento como uma política em **XACML**.

Representação XACML

Considerado o exemplo da figura 29 o consentimento representou-se em **XACML** da seguinte forma tal como ilustrado na figura 35. Declarou-se uma lista de políticas, com um algoritmo de combinação *deny-overrides* (linha número 3) para que todas as transformações diferentes sejam verificadas e o pedido

de acesso não seja retornado logo após encontrar uma regra que permita o acesso. Com o elemento *<Target>* (linha número 4-37) é especificado que esta política se aplica para documentos do *utilizador x* (linha número 10), quem acede tem de pertencer à *orgA.com* (linha número 19) e a data do pedido de acesso tem de ser maior que *15 de novembro de 2022* (linha número 28). Nas linhas número 41-74, existe outro elemento *<Target>*, a especificar as intenções *F1, F2, F3* (linhas número 47, 57, 67) para o qual esta lista de políticas é válida. Por fim é declarado na linha número 75, uma política de id *p1*, dentro desta política é verificado se o documento contém um bloco do tipo *Bloco1*, caso tenha então são retornadas as transformações representadas nas linhas número 99-106. Para a política *p2* o processo é análogo.

```
1  <PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
2  PolicySetId="root" Version="1.0.0" PolicyCombiningAlgId="urn:oasis:names:tc:
3  xacml:3.0:policy-combining-algorithm:deny-overrides">
4     <Target>
5         <AnyOf>
6              <AllOf>
7                  <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
8                  string-equal">
9                         <AttributeValue DataType="http://www.w3.org/2001/
10                        XMLSchema#string">utilizador x</AttributeValue>
11                        <AttributeDesignator MustBePresent="false"
12                        Category="user" AttributeId="user:id"
13                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
14                 </Match>
15                 <Match
16                  MatchId="urn:oasis:names:tc:xacml:1.0:function:
17                  rfc822Name-match">
18                        <AttributeValue DataType="http://www.w3.org/2001/
19                        XMLSchema#string">orgA.com</AttributeValue>
20                        <AttributeDesignator MustBePresent="false"
21                        Category="subject" AttributeId="subject:id"
22                        DataType="urn:oasis:names:tc:xacml:
23                        1.0:data-type:rfc822Name"/>
24                 </Match>
25                 <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
```

```
                date-less-than">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#date">15 de novembro de 2022</AttributeValue>
                        <AttributeDesignator MustBePresent="false"
                        Category="urn:oasis:names:tc:xacml:1.0:subject-category
                        :access-subject" AttributeId="urn:oasis:names:tc:
                        xacml:1.0:environment:current-date"
                        DataType="http://www.w3.org/2001/XMLSchema#date" />
                </Match>
            </AllOf>
        </AnyOf>
    </Target>
    <PolicySet PolicySetId="policyuser1"  Version="1.0.0"
    PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-
    algorithm:deny-overrides">
        <Target>
                <AnyOf>
                    <AllOf>
                        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
                        string-equal">
                            <AttributeValue DataType="http://www.w3.org/2001/
                            XMLSchema#string">F1</AttributeValue>
                            <AttributeDesignator MustBePresent="false"
                            Category="intention"  AttributeId="intention:id"
                            DataType="http://www.w3.org/2001/XMLSchema#string"/>
                        </Match>
                    </AllOf>
                    <AllOf>
                        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
                        string-equal">
                            <AttributeValue DataType="http://www.w3.org/2001/
                            XMLSchema#string">F2</AttributeValue>
                            <AttributeDesignator MustBePresent="false"
                            Category="intention"  AttributeId="intention:id"
```

```
                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
                    </Match>
                </AllOf>
                <AllOf>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
                    string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">F3</AttributeValue>
                        <AttributeDesignator MustBePresent="false"
                        Category="intention"  AttributeId="intention:id"
                        DataType="http://www.w3.org/2001/XMLSchema#string"/>
                    </Match>
                </AllOf>
            </AnyOf>
        </Target>
        <Policy PolicyId="p1" Version="1.0"
        RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:
        rule-combining-algorithm:deny-overrides">
            <Target/>
            <Rule RuleId="B1" Effect="Permit">
                <Target>
                        <AnyOf>
                            <AllOf>
                                <Match MatchId="urn:oasis:names:tc:xacml:1.0:
                                function:string-equal">
                                    <AttributeValue DataType="http://www.w3.org/
                                    2001/XMLSchema#
                                    ">Bloco1</AttributeValue>
                                    <AttributeDesignator MustBePresent="false"
                                    Category="document"
                                     AttributeId="document:block" DataType=
                                     "http://www.w3.org/2001/
                                     XMLSchema#string"/>
                                </Match>
```

```
                        </AllOf>
                    </AnyOf>
            </Target>
        </Rule>

        <ObligationExpressions>
            <ObligationExpression ObligationId="HIDE" FulfillOn="Permit">
               <AttributeAssignmentExpression AttributeId="args">
                    <AttributeValue DataType="http://www.w3.org/2001/
                    XMLSchema#string">/NúmeroDaPorta</AttributeValue>
                </AttributeAssignmentExpression>
            </ObligationExpression>
        </ObligationExpressions>

    </Policy>
    <Policy PolicyId="p2" Version="1.0"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:
    rule-combining-algorithm:deny-overrides">
        <Target/>
        <Rule RuleId="B2" Effect="Permit">
            <Target>
                    <AnyOf>
                        <AllOf>
                            <Match MatchId="urn:oasis:names:tc:xacml:1.0:
                            function:string-equal">
                                <AttributeValue DataType="http://www.w3.org
                                /2001/XMLSchema
                                #string">Bloco2</AttributeValue>
                                <AttributeDesignator MustBePresent="false"
                                Category="document"
                                AttributeId="document:block"
                                DataType="http://www.w3.org/2001/XMLSchema
                                #string"/>
                            </Match>
```

```
                            </AllOf>
                        </AnyOf>
                </Target>
            </Rule>

            <ObligationExpressions>
                <ObligationExpression ObligationId="HIDE" FulfillOn="Permit">
                   <AttributeAssignmentExpression AttributeId="args">
                        <AttributeValue DataType="http://www.w3.org/2001/
                        XMLSchema#string">/Localidade</AttributeValue>
                    </AttributeAssignmentExpression>
                </ObligationExpression>
            </ObligationExpressions>
        </Policy>
    </PolicySet>
</PolicySet>
```

Figura 30: Política de teste a inserir no *AuthZForce*.

Capítulo 4

Avaliação

Este capitulo apresenta a campanha de avaliação conduzida para identificar e avaliar a robustez, desempenho e escalabilidade do **SDAM**.

4.1 Arquiteturas de teste

A avaliação de desempenho foi conduzida tendo por base duas arquiteturas de referência. Em ambas, a avaliação compreende a identificação dos limites de desempenho e escalabilidade à medida que o número de clientes concorrentes aumenta, induzindo uma carga operacional crescente sob todo o sistema. Para tal desenhou-se (1) uma arquitetura básica, construida por uma rede **HLF** básica; e (2) uma arquitetura incluindo o **SDAM**. Ambas as arquiteturas são descritas de seguida.

4.1.1 Arquitetura básica

Para testar a viabilidade desta solução foi utilizada a rede de teste com *CouchDB* de *fabric-samples*[1], que consiste numa rede com duas organizações, onde cada organização possui um *peer* e uma base de dados *CouchDB*, onde os recursos são armazenados. Para interagir com a rede foi instalado na mesma um *chaincode*, onde é definida a estrutura dos recursos e algumas das funções padrão do *chaincode* para aceder os recursos.

4.1.2 Arquitetura com SDAM

Esta arquitetura parte da configuração básica, adicionando o **SDAM** através da adição do sistema *AuthZ-Force*. Nesse cenário, o **HLF** considera que a base de dados é o **SDAM**, transmitindo exatamente as mesmas operações.

[1] Códito e turoriais de utilização dos *fabric-samples*: https://github.com/hyperledger/fabric-samples

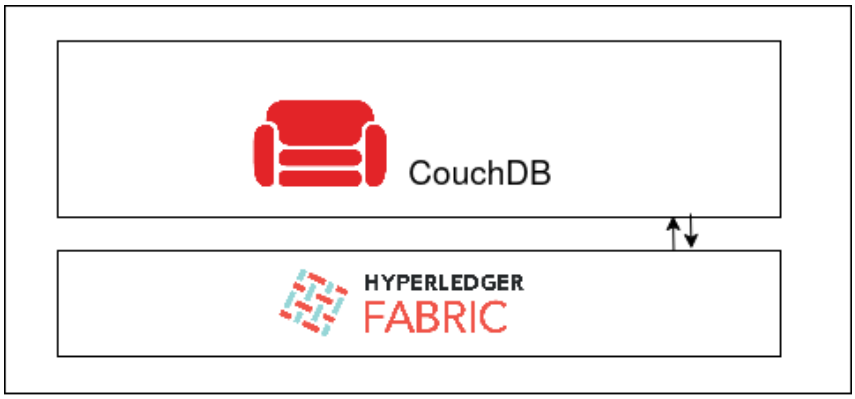


Figura 31: Arquitetura básica.

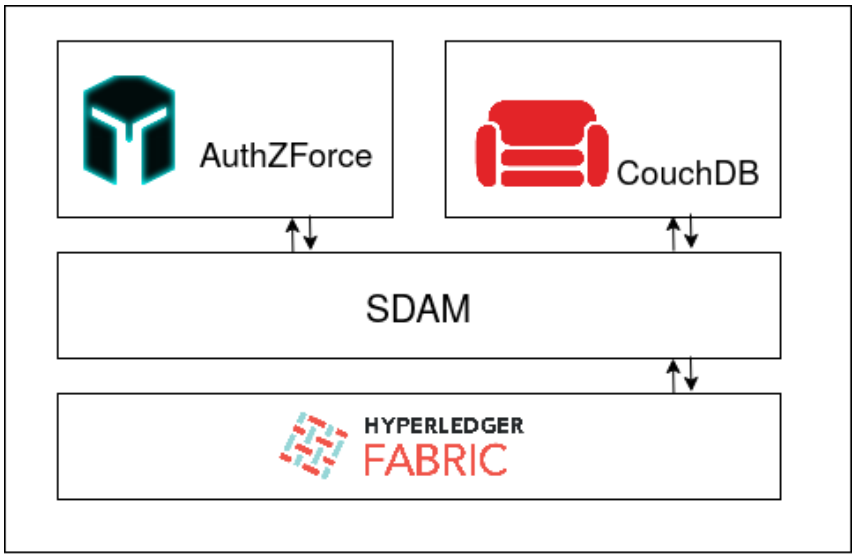


Figura 32: Arquitetura com SDAM.

4.2 Ferramentas de teste

Para realizar e obter dados relevantes dos testes foi utilizada a ferramenta Apache Jmeter [2], que nos permitiu computar a latência total. Além disso, no **SDAM** foram criados *checkpoints* para as várias fases, nomeadamente: *CouchDB* (realizar o pedido e receber a resposta), *AuthZForce* (realizar o pedido e receber a resposta) e *Transformações-ação* (Transformar os dados), por forma a permitir obter o tempo que cada uma destas fases ocupa (figura 33).

Os testes foram executados em três máquinas iguais, equipadas com um processador Intel(R) Core(TM) i5-10505 CPU @ 3.20GHz, 16GB RAM e 1 disco HDD de 500GB, estando interligadas por uma rede *Gigabit* à distância máxima de 1 salto. Durante o processo de avaliação estas maquinas são referidas como

[2] Site oficial: https://https://jmeter.apache.org

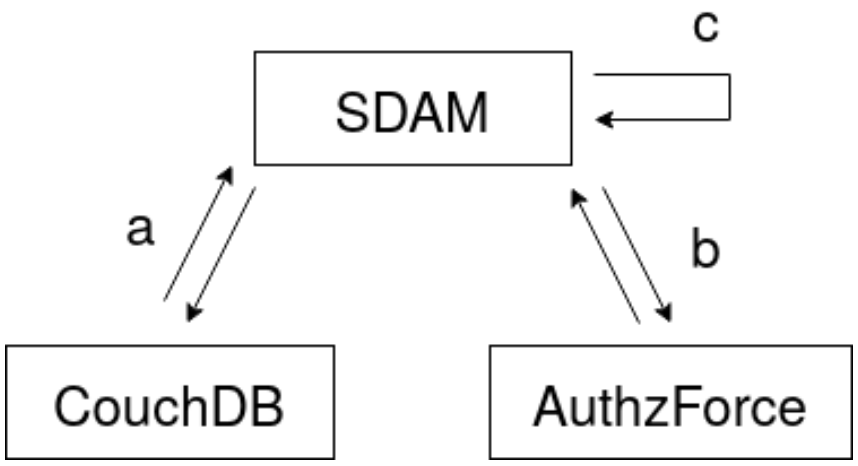


Figura 33: Sequência de processamento do SDAM (a=*CouchDB* -> b=*AuthZForce* -> c=*Transformações*).

máquina A, máquina B e máquina C.

4.3 Documentos para testes

Foram necessários alguns ficheiros para a realização dos testes, nomeadamente um ficheiro para utilizar como o recurso, outro documento que contém uma política a inserir no *AuthZForce* para ser utilizado com o **SDAM**. e um *chaincode* que permita interagir com o **HLF**.

4.3.1 Recurso

Como referido na secção 1.2, este projeto foi motivado pelo desenvolvimento de uma plataforma colaborativa que viabiliza processos no domínio segurador, serem desmaterializados e automatizados com foco em negociação de contratos e processamento de pedidos, recorrendo ao **HLF**. Dessa forma, o recurso utilizado para testes foi um recurso disponibilizado deste domínio (figura 34).

```
{
    "type": "com.nau21.sis.party.model.pojo.SimplePersonData",
    "addresses": [
        {
            "type": "com.nau21.sis.party.model.pojo.SimplePartyAddressData",
            "address": {
                "type": "com.nau21.sis.party.model.pojo.SimpleAddressData",
                "addressLines": [
                    "ny street"
```

```
                ],
                "city": "Guimaraes",
                "country": "PT",
                "county": "Minho",
                "postalCode": "4111-976",
                "region": "Braga"
            },
                "addressType": "GENERAL",
            "preferred": true,
            "preferredForType": true
        }
    ],
    "birthDate": "2002-04-01",
    "externalKeysMap": {
        "type": "com.nau21.sis.util.model.pojo.SimpleExternalKeysMap",
        "externalKeysMap": [
            {
                "external": {
                    "key": "IS1C1E1",
                    "system": "SISD1-DAML"
                },
                "internal": {
                    "key": "IS1C1E1",
                    "system": "SISD1-DAML"
                }
            }
        ]
    },
    "fiscalInformation": [
        {
            "type": "com.nau21.sis.party.model.pojo.SimpleFiscalInformationData",
            "fiscalCountry": "PT",
            "fiscalNumber": "125594062"
        }
```

```
    ],
    "gender": 2,
    "name": {
        "type": "com.nau21.sis.party.model.PersonName",
        "familyName": "Doe",
        "firstName": "Jane",
        "name": "Jane Doe"
    },
    "partyRef": "353847a6-fa96-4481-b026-f371ad162994",
    "valueLists": {
        "Identification_Numbers": {
            "values": [
                {
                    "key": {
                        "type": "simple_string",
                        "code": "CIN"
                    },
                    "value": "13353414"
                },
                {
                    "key": {
                        "type": "simple_string",
                        "code": "SSN "
                    },
                    "value": "11910192748"
                }
            ]
        },
        "JOB_INFOS": {
            "values": [
                {
                    "key": {
                        "type": "index_date",
                        "code": "Position",
```

```
                "index": "2020-09-30T23:00:00Z"
            },
            "value": "Software Arquitect"
        },
        {
            "key": {
                "type": "index_date",
                "code": "Position",
                "index": "2022-09-30T23:00:00Z"
            },
            "value": "Lead Software Arquitect"
        },
        {
            "key": {
                "type": "index_date",
                "code": "Salary",
                "index": "2020-09-30T23:00:00Z"
            },
            "value": 10000
        },
        {
            "key": {
                "type": "index_date",
                "code": "Salary",
                "index": "2021-09-30T23:00:00Z"
            },
            "value": 10001
        },
        {
            "key": {
                "type": "index_date",
                "code": "Salary",
                "index": "2022-01-01T00:00:00Z"
            },
```

```
                    "value": 10001
                }
            ]
        }
    }
}
```

Figura 34: Recurso a utilizar para testes.

4.3.2 Política

Foi criada também uma política em **XACML** (figura 35), para inserir no *AuthZForce*. Esta especifica que se o atributo de id *resource:id* (linha número 21) for igual a *asset1* (linha número 18), então para permitir o acesso, o atributo de id *subject:group* (linha número 33) tem que ser igual a *admin* (linha número 29). Além disso, se o acesso for concedido, será necessário efetuar a Transformação *HIDE* (linha número 43) com o argumento */name* (linha número 46).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
PolicySetId="root" Version="1.0.0" PolicyCombiningAlgId="urn:oasis:names:tc:
xacml:3.0:policy-combining-algorithm:deny-unless-permit">

    <Target />
    <Policy PolicyId="p1" Version="1.0"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:
    rule-combining-algorithm:deny-unless-permit">
        <Target>
            <AnyOf>
                <AllOf>
                    <Match
                    MatchId="urn:oasis:names:tc:xacml:1.0:function:
                    string-equal">
                        <AttributeValue
                        DataType="http://www.w3.org/2001/XMLSchema
```

```
                    #string">asset1</AttributeValue>
                    <AttributeDesignator MustBePresent="false"
                    Category="urn:oasis:names:tc:xacml:1.0:resource"
                    AttributeId="urn:oasis:names:tc:xacml:1.0:resource:id"
                    DataType="http://www.w3.org/2001/XMLSchema#string" />
                </Match>
                <Match
                MatchId="urn:oasis:names:tc:xacml:1.0:function:
                string-equal">
                    <AttributeValue
                    DataType="http://www.w3.org/2001/XMLSchema
                    #string">admin</AttributeValue>
                    <AttributeDesignator MustBePresent="false"
                    Category="urn:oasis:names:tc:xacml:1.0:subject"
                    AttributeId="urn:oasis:names:tc:xacml:1.0:
                    subject:group" DataType="http://www.w3.org/2001/
                    XMLSchema#string" />
                </Match>
            </AllOf>
        </AnyOf>
    </Target>
    <Rule RuleId="Rulea" Effect="Permit" />


    <ObligationExpressions>
        <ObligationExpression ObligationId="HIDE" FulfillOn="Permit">
            <AttributeAssignmentExpression AttributeId="arg">
                <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#string">/name
                </AttributeValue>
            </AttributeAssignmentExpression>
        </ObligationExpression>
    </ObligationExpressions>

```

```
52      </Policy>
53
54  </PolicySet>
```

Figura 35: Política de teste a inserir no *AuthZForce*.

4.3.3 Chaincode

Para poder operar sobre os dados do **HLF** é necessário inserir na rede do **HLF** um *chaincode* que permita aceder a esses dados. Foi criado em JAVA e GO um *chaincode* simples com as estruturas próprias para funcionar com documentos que adoptem a mesma estrutura representada na figura 34; fornecendo os métodos **ReadAssets, InitLedger, CreateAsset, QueryResults**, por forma a operar sobre os dados.

4.4 Cenários de teste

Com estas arquiteturas, antes da realização dos testes, instalou-se as componentes nas devidas máquinas e fez-se a inserção de dados na base de dados e as políticas no gestor de políticas. A execução dos testes foi conduzida de acordo com 3 cenários distintos: (1) cenário básico, (2) cenário com **SDAM** e **HLF** e (3) cenário com **SDAM** sem **HLF**.

4.4.1 Cenário básico

Neste cenário, implementou-se a arquitetura básica. Na máquina A instalou-se o **HLF** e colocou-se a base de dados num *container*. Nas máquinas B e C instalou-se o *Apache Jmeter* (figura 37). Sobre a base dados é também inserido o recurso *asset1* (figura 34), através da respetiva operação no *chaincode*.

Exemplo Pedido

Neste cenário, o pedido que vai ser executado pelo *Apache Jmeter* utiliza o comando *peer* do **HLF** e executa a transação *QueryAssets* (figura 36).

```
peer chaincode query -C mychannel -n ledger -c '{"Args":["QueryAssets",
"{\"selector\":{\"gender\":2}}"]}'
```

Figura 36: Comando *peer* de teste para o cenário básico.

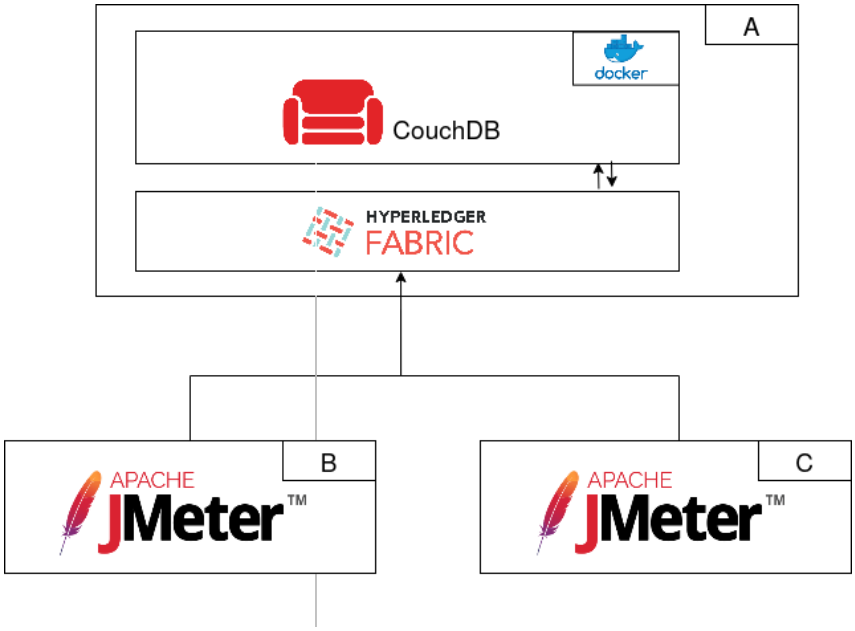


Figura 37: Cenário básico.

4.4.2 Cenário de SDAM

Neste cenário, implementou-se a arquitetura com **SDAM**. Na máquina A instalou-se o **HLF** e colocou-se o módulo de políticas e a base de dados em *containers*. Nas máquinas B e C instalou-se o *Apache Jmeter* (figura 39). Na base de dados inseriu-se o recurso *asset1* (figura 34), através da respetiva operação no *chaincode*, e inseriu-se a política (figura 35) de acesso no módulo de políticas através de um pedido *http*.

Exemplo Pedido

Neste cenário o pedido a ser executado pelo *Apache Jmeter* utiliza o comando *peer* do **HLF** e executa a transação *QueryAssets* (figura 38).

```
peer chaincode query -C mychannel -n ledger -c '{"Args":["QueryAssets",
"{\"attributes\":[{\"category\":\"subject\",\"attributeID\":
\"subject:group\",\"datatype\":\"http://www.w3.org/2001/XMLSchema#string\"
,\"value\":\"admin\"}],\"selector\":{\"assetID\":{\"$eq\":\"asset1\"}
},\"execution_stats\":true}"]}'
```

Figura 38: Comando *peer* de teste para o cenário de SDAM.

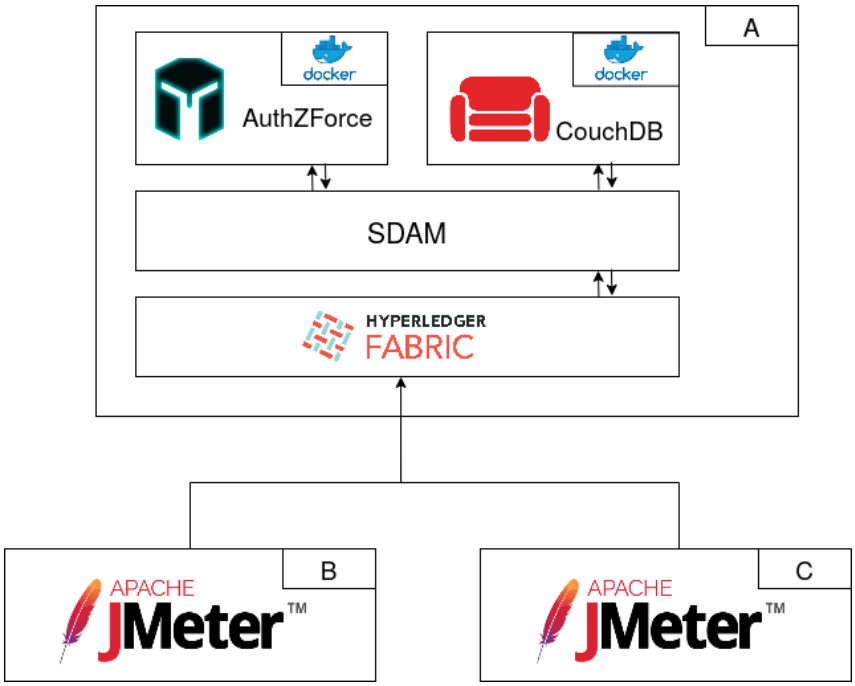


Figura 39: Cenário com SDAM.

4.4.3 Cenário de SDAM, sem HLF

Neste cenário, cosiderou-se o cenário anterior, porém não ligando o **SDAM** ao **HLF**. Ou seja, o *Apache Jmeter* vai fazer pedidos diretamente ao **SDAM**.

Exemplo Pedido

Neste cenário o pedido a ser executado pelo *Apache Jmeter* é um pedido http *GET* à rota */db/docid*.

Na linha número 5 da figura 40 é apresentado o corpo do pedido em *JSON* que contém apenas uma chave *sdam_options* e onde todas as informações relativamente ao atributo de id *subject:group* estão contidas.

```
GET /mychannel_ledger/asset1 HTTP/1.1
Accept: application/json
Content-Type: application/json
Host: localhost:8080
{   "sdam_options": [
        {
            "category": "subject",
            "attributeID": "subject:group",
            "datatype": "http://www.w3.org/2001/XMLSchema#string",
            "value": ["admin"]
        }]
}
```

Figura 40: Corpo do teste do cenário de SDAM, sem HLF

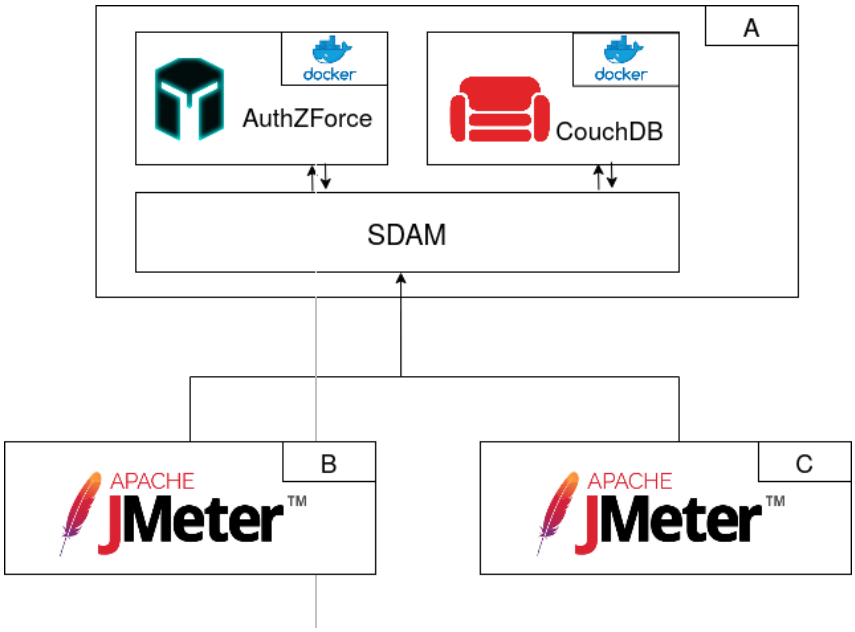


Figura 41: Cenário com SDAM, sem HLF.

4.5 Resultados

Foram executados vários testes de carga, tendo cada um durado 30 segundos. O número de clientes a executar pedidos variaram entre os valores [1, 10, 30, 60, 100, 150, 200, 500, 100]. Os valores apresen-

tados são o resultados de 3 execuções, computando a média aritmética das 3 experiências. Utilizaram-se os cenários apresentados anteriormente, aumentando o número de clientes a serem lançados pelo *Apache Jmeter*, tendo sido divididos igualmente entre máquina B e C.

4.5.1 Primeira execução

Cenário Básico

	1° execução	2° execução	3° execução	Média
1	31,11	31,56	31,35	31,34
10	34,27	34,29	34,25	34,27
30	77,07	77,18	77,10	77,12
60	159,73	159,89	159,99	159,87
100	267,47	268,32	268,65	268,15
150	400,25	404,29	404,43	402,99
200	516,38	514,76	535,91	522,35
500	630,82	573,04	601,62	601,82
1000	650,31	654,10	638,81	647,74

Tabela 1: Tabela com o tempo de resposta em ms dos primeiros resultados do cenário básico, variando entre 1 e 1000 clientes.

Da observação dos resultados da tabela 1, o tempo de resposta é afetado pelo número de clientes concorrentes no sistema. Porém, este aumento é baixo, considerando em especifico o diferencial entre o teste com 500 e 1000 clientes, onde se registou um diferencial de 45,92 ms no tempo de resposta total.

	1° execução	2° execução	3° execução	Desvio Padrão
1	4,31	4,44	5,20	4,65
10	3,19	3,20	3,14	3,18
30	17,33	17,33	17,36	17,34
60	55,75	55,25	54,28	55,09
100	130,73	131,60	131,92	131,41
150	230,42	222,96	231,10	228,16
200	322,11	326,52	316,20	321,61
500	557,78	458,95	535,13	517,29
1000	493,92	501,54	480,29	491,92

Tabela 2: Tabela com o desvio padrão dos primeiros resultados do cenário básico, variando entre 1 e 1000 clientes.

Considerando as observações, os desvios padrão computados indicam que os tempos de resposta apresentam uma variação bastante elevada. Por exemplo, a 3ª execução de 1000 clientes teve uma média de 638,81ms para o tempo de resposta (tabela 1) e teve 480,29ms de desvio padrão (tabela 2), o que é mais de metade da média.

Cenário de SDAM

	1° execução	2° execução	3° execução	Média
1	42,53	37,89	35,79	38,74
10	84,14	81,72	82,41	82,75
30	243,89	235,97	232,03	237,30
60	513,55	457,51	479,65	483,57
100	822,83	796,58	832,11	817,17
150	1282,52	1212,34	1264,91	1253,25
200	1643,77	1605,91	1675,11	1641,60
500	3720,97	3837,22	3997,34	3851,84
1000	7482,33	7786,07	7155,80	7474,73

Tabela 3: Tabela com o tempo de resposta em ms dos primeiros resultados do cenário com SDAM, variando entre 1 e 1000 clientes.

Observando os valores representados na tabela 3, percebe-se que o aumento registado de tempo de resposta é proporcional ao número de clientes.

	1º execução	2º execução	3º execução	Desvio Padrão
1	13,09	9,22	7,58	9,96
10	31,21	27,64	29,06	29,30
30	100,45	94,62	92,60	95,89
60	201,77	278,96	347,84	276,19
100	413,01	632,48	540,13	528,54
150	552,92	813,46	841,83	736,07
200	831,99	1456,90	1351,53	1213,47
500	3729,05	3769,18	1599,86	3032,70
1000	4492,09	4575,44	3976,20	4347,91

Tabela 4: Tabela com o desvio padrão dos primeiros resultados do cenário com SDAM, variando entre 1 e 1000 clientes.

A tabela 4 apresenta os resultados dos testes conduzidos para o cenário com **SDAM**. Embora o crescimento do número de clientes induza o crescimento no tempo de execução, o desvio padrão registado motiva uma grande variabilidade nas leituras do sistema. Conduziu-se dessa forma uma avaliação mais detalhada dos resultados, através da observação dos tempos parciais, associados a cada fase do processo.

A tabela 5 apresenta a média das observações em cada uma das fases do processo para o cenário com **SDAM**. Verifica-se que independentemente do número de clientes aplicado ao sistema, o impacto na fase de transformação de dados é negligenciável. Verifica-se também que a variabilidade registada em conjuntamente da fase CouchDB e consulta ao AuthZFroce.

Conjuntamente, a tabela 6 evidencia o desvio padrão registado para o mesmo cenário. Da experiência, concluiu-se que a variabilidade registada no sistema advém do tempo de resposta associado ao CouchDB.

Média

	CouchDB	AuthZForce	Transformação
1	2,33	1,45	0,01
10	19,97	15,99	0,00
30	69,15	87,55	0,00
60	201,00	146,98	0,01
100	380,94	207,19	0,00
150	559,10	321,93	0,00
200	749,37	443,24	0,00
500	2081,50	792,12	0,00
1000	2865,35	2142,90	0,00

Tabela 5: Tabela com o tempo de resposta em ms dos resultados parciais dos primeiros resultados do cenário com SDAM, variando entre 1 e 1000 clientes.

Desvio Padrão

	CouchDB	AuthZForce	Transformação
1	1,58	1,61	0,36
10	61,79	60,90	0,22
30	151,41	199,83	0,25
60	586,06	438,51	0,49
100	1202,60	736,66	0,27
150	1625,69	942,24	0,25
200	2650,22	1736,80	0,23
500	7511,65	2908,72	0,25
1000	7981,82	6346,81	0,37

Tabela 6: Desvio padrão dos resultados parciais dos primeiros resultados do cenário com SDAM, variando entre 1 e 1000 clientes.

Comparação dos resultados

O gráfico 42 apresenta as três fases do **SDAM**: *CouchDB*, *AuthZForce* e *Transformações* apresentadas na figura 33. Exibe ainda o *Resto* que é a diferença entre o tempo de resposta e o total da soma das três

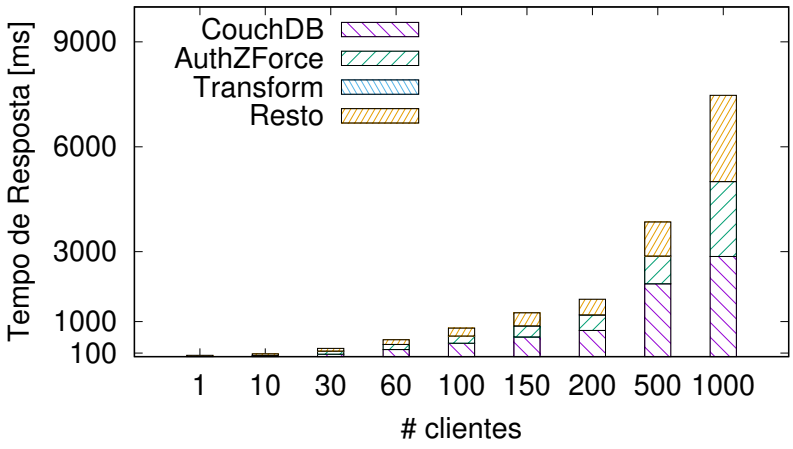


Figura 42: Gráfico com primeiros resultados dos resultados parciais do cenário de SDAM.

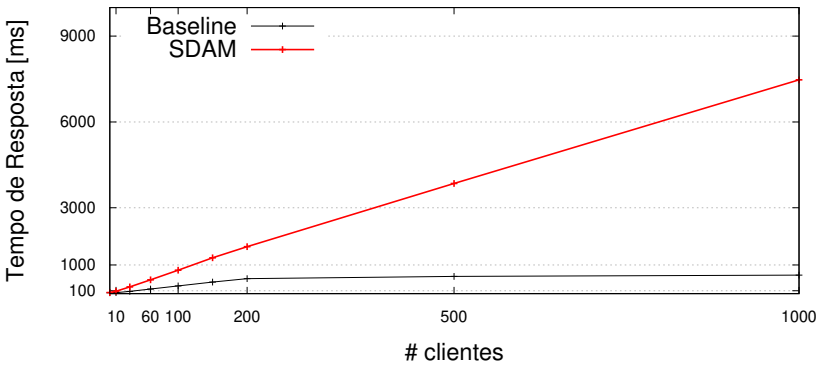


Figura 43: Gráfico com primeiros resultados com SDAM e sem SDAM.

fases, representando assim a latência de rede, gestão e processamento do **HLF** e o tempo que demora o **SDAM** a receber os pedidos e mapear para os métodos correspondentes, coletivamente.

Interpretando a figura 43 e as tabelas 1 e 3, conclui-se que a partir de 60 clientes existe um deterioramento no tempo de resposta evidente no cenário com **SDAM**. Até lá, a utilização de 1 cliente implica uma diferença de 7,39ms, para 10 clientes, uma diferença de 48,48ms, para 60 clientes, uma diferença de 160,17ms. Com o aumento de clientes a partir de 60 clientes, o tempo de resposta dobra, tomando valores impraticáveis, chegando a demorar 7474,73ms para 1000 clientes (tabela 3).

Olhando novamente para o gráfico 42, podemos observar que algo que causa este aumento no tempo de resposta é o aumento rápido do tempo de resposta do *CouchDB* com o número de clientes, algo que foge ao controlo disponível, pois a base de dados funciona exatamente da mesma maneira com e sem **SDAM**. Evidencia-se que por natureza, o **HLF** não é um sistema otimizado para oferecer resposta a interrogações de forma eficiente, sendo este reponsável por uma grande parte do tempo de resposta. Com o aumento no módulo de políticas (*AuthZForce*), uma possível solução a validar, (viável devido às

políticas serem algo que não é alterado constantemente) é criar uma *cache* que guarda as decisões e as *Transformações* a efetuar para um conjunto de atributos. Esta informação é periodicamente carregada, atualizando a referida *cache*. O *Resto* também apresenta um crescimento rápido. Identifica-se também associado a este crescimento do **SDAM** o facto deste não conseguir realizar uma alocação de *threads* eficiente para os pedidos que recebe, podendo esta facto influenciar o tempo de reposta do *CouchDB*.

4.5.2 Identificação de problemas

Na análise dos resultados, conclui-se que os primeiros resultados de tempo de resposta do *CouchDB* que chegava a 2865ms para 1000 clientes (tabela 5) não são resultados normais, mas sim o resultado de algum erro de execução. Realizaram-se testes direcionados ao *CouchDB*, ou seja, 1 teste de carga, com o *Apache Jmeter* instanciando 1000 clientes que fazem pedidos http para obter um documento alojado no *CouchDB*. Observando os resultados da tabela 7, temos que na média de 3 execuções para 1000 clientes a fazerem pedidos ao *CouchDB* durante 30s, observa-se em média 377,83ms de tempo de resposta, o que está um nível de grandeza abaixo, comparando com os resultados parciais do *CouchDB* obtidos nos primeiros testes (tabela 5).

Considerou-se ainda a existência de alguma sobrecarga do *cpu*, por isso mesmo foram repetidos os primeiros testes e, em simultâneo, observou-se o *cpu*. Não se encontrou qualquer evidência de sobrecarga, tendo sido encontrados valores de uso do *cpu* que variavam entre 18-26% durante as execuções dos testes.

	1° execução	2° execução	3° execução	Média
Tempo de Resposta	373,06	381,01	379,42	377,83
Desvio Padrão	94,97	77,74	70,40	81,04

Tabela 7: Tempo de Resposta e Desvio padrão em ms dos segundos resultados do teste jmeter->*CouchDB* para 1000 clientes.

Um dos pontos que se observou nos primeiros testes foi o facto dos resultados derivados do teste parcial ao *CouchDB* crescerem rapidamente. Conjuntamente, o teste anterior da tabela 7, confirma que o **SDAM**, quando faz pedidos ao *CouchDB*, obtém tempos de resposta piores dos que os esperados. Por isso neste segundo teste, criou-se uma **thread pool** elástica, dedicada a realizar pedidos ao *CouchDB*. Além disso, reparou-se que se executava o SDAM em **modo debug**, promovendo o decaimento de performance associado ao **SDAM**

4.5.3 Resultados finais

Cenário de SDAM

Nesta campanha final de resultados, adotou-se a *thread pool* e desligou-se o modo *debug*. Visto que as políticas armazenadas configuram informação que não carece de atualizações constantes, considerou-se a implementação da referida **cache** no **SDAM**, por forma a guardar as decisões e *Transformações* a aplicar num conjunto de atributos que definem um pedido. A *cache* será atualizada periodicamente para a sua correta utilização. Após estas alterações, repetiram-se os testes, cujos resultados se encontram nas tabelas 8, 9, 10 e 11, respetivamente.

	1° execução	2° execução	3° execução	Média
1	50,60	54,96	48,68	51,41
10	54,84	73,89	56,90	61,88
30	142,50	146,71	227,97	172,39
60	260,27	269,33	275,80	268,47
100	395,15	406,46	419,83	407,15
150	523,85	542,29	564,49	543,54
200	608,18	633,12	680,51	640,60
500	1510,62	1564,41	1722,46	1599,16
1000	3465,46	3536,91	3758,11	3586,83

Tabela 8: Tempo de resposta em ms dos segundos resultados do cenário com SDAM, variando entre 1 e 1000 clientes.

	1° execução	2° execução	3° execução	Desvio Padrão
1	18,81	22,58	16,79	19,40
10	16,93	27,94	18,30	21,05
30	80,00	81,81	114,22	92,01
60	124,60	138,55	132,20	131,78
100	206,43	195,31	200,33	200,69
150	247,86	251,75	270,72	256,78
200	304,05	301,59	324,66	310,10
500	902,25	886,41	859,35	882,67
1000	2001,40	1892,48	2045,73	1979,87

Tabela 9: Tabela com o desvio padrão em ms dos segundos resultados do cenário com SDAM, variando entre 1 e 1000 clientes.

Comparação de Resultados

Com estas alterações, conseguiu resolver o crescimento rápido que se havia verificado nos resultados parciais do *CouchDB*. Tal torna-se claro, pela observação das tabelas 5 e 10 ou para os gráficos das figuras 42 e 45. Analisando o tempo de resposta (tabela 8), no cenário **SDAM** e comparando-o com o cenário básico (sem **SDAM**, tabela 1), aferiu-se que numa configuração até 200 clientes, o acréscimo incorrido ao usar o **SDAM**, em média, é de 92,76ms (tabela 12, figura 44). A evolução progressiva do desempenho obtido para o **SDAM** e **HLF** sugere que o sistema é capaz de escalar eficientemente à marca de 200 clientes, cenário que se considera adequado para o caso-de-uso em particular.

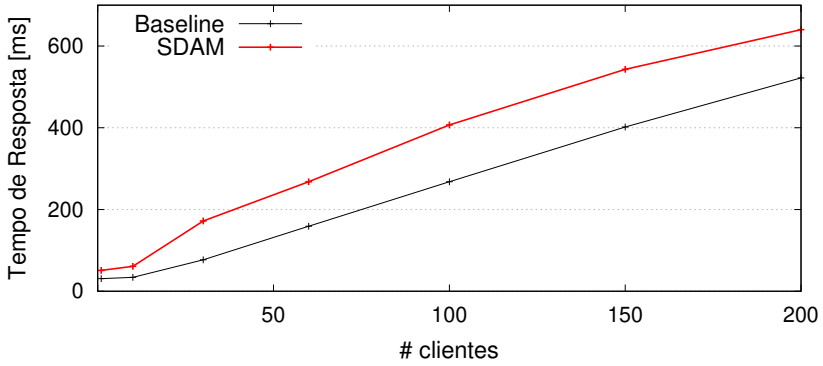


Figura 44: Gráfico com segundos resultados com SDAM e sem SDAM

As funções de distribuição cumulativa para o tempo de resposta com **SDAM**, com aumento de clientes

Média

	CouchDB	AuthZForce	Transformações
1	8,15	0,01	0,00
10	26,72	0,02	0,00
30	82,17	0,01	0,00
60	129,99	0,01	0,00
100	190,18	0,01	0,00
150	221,06	0,01	0,00
200	178,31	0,01	0,00
500	130,73	0,01	0,00
1000	7,67	0,02	0,00

Tabela 10: Tabela com o tempo de resposta em ms dos resultados parciais dos segundos resultados do cenário com SDAM, variando entre 1 e 1000 clientes.

Desvio Padrão

	CouchDB	AuthZForce	Transformações
1	4,25	0,13	0,05
10	17,01	0,15	0,06
30	62,70	0,14	0,06
60	96,49	0,11	0,05
100	142,74	0,11	0,04
150	170,16	0,10	0,05
200	193,34	0,21	0,03
500	195,64	0,12	0,05
1000	18,40	0,21	0,07

Tabela 11: Tabela com o desvio padrão em ms dos resultados parciais dos segundos resultados do cenário com SDAM, variando entre 1 e 1000 clientes.

é representado na figura 46. De 1 a 10 clientes, mais de 90% dos pedidos são concluídos em menos de 5 milissegundos. Com 30 clientes, 90% dos clientes concluíram em menos de 10 milissegundos, com

	c/ SDAM	s/ SDAM	diferença
1	51,41	31,34	20,07
10	61,88	34,28	27,60
30	172,39	77,12	95,27
60	268,47	159,87	108,60
100	407,15	268,15	139,00
150	543,54	402,99	140,55
200	640,60	522,36	118,24
500	1599,16	601,83	997,33
1000	3586,83	647,74	2939,09

Tabela 12: Diferença das médias do tempo de resposta em ms com SDAM e sem SDAM (cenário básico), variando entre 1 e 1000 clientes.

os 10% restantes a acumular tempos de resposta que vão até 100 milissegundos. Com 100 ou mais clientes, as distribuições têm caudas maiores, indicando maior variabilidade nos tempos de resposta. Como mencionado, o comportamento observado mostra-se uma consequência direta da variabilidade introduzida pelo sistema de base dados.

Realizaram-se ainda testes para o cenário com **SDAM**, sem **HLF**, com o número de clientes a variar entre [1, 10, 100, 200]. Os tempos de resposta com **SDAM**, sem **HLF**, apresentados na figura 13 são

	Média	Desvio Padrão
1	6,49	2,19
10	29,35	16,21
100	218,41	471,04
200	238,34	518,54

Tabela 13: Média de tempo de Resposta (ms) e desvio padrão de três execuções para o cenário com SDAM, sem HLF, variando entre 1 e 200 clientes.

relativamente baixos.

Comparando estes tempos de resposta com os calculados nos primeiros testes para o cenário básico, temos que a realização dos pedidos pelo **HLF**, impõe uma penalização no tempo de resposta que varia

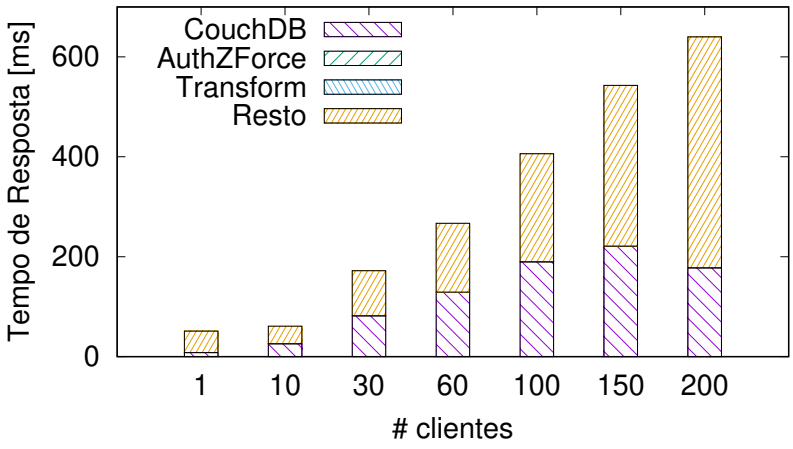


Figura 45: Gráfico com os segundos resultados dos resultados parciais do cenário de SDAM

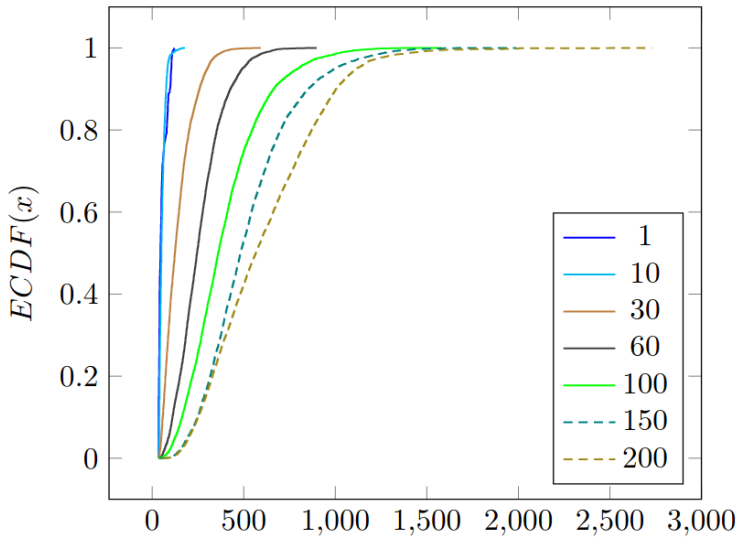


Figura 46: Gráfico com as cdfs para SDAM com aumento de clientes, tempo de resposta em ms.

	s/ **SDAM**	c/ **SDAM**, sem **HLF**	diferença
1	51,41	6,49	44,91
10	61,88	29,35	32,52
100	407,15	218,41	188,73
200	640,6	238,34	402,25

Tabela 14: Diferença das médias do tempo de resposta em ms sem SDAM (cenário básico) e com SDAM, sem HLF, variando entre 1 e 200 clientes.

entre 44,91ms e 402,25ms, como se pode observar na tabela 14.

4.5.4 Avaliação do módulo de políticas

Executaram-se também testes de carga ao módulo de políticas para se medir o tempo de resposta e desta forma aferir o impacto que o número de políticas guardadas no módulo de políticas e o número de pedidos de acesso concorrentes causam no desempenho deste módulo.

Estes testes foram realizados com um módulo de políticas a ser executado na máquina A, com o número de políticas para diferentes testes a variar entre [10, 100, 1000, 10000]. Ao mesmo tempo, foi executado durante 30 segundos na máquina B e C a aplicação *Apache Jmeter* a realizar pedidos de acesso *http* ao módulo de políticas, com o número de clientes total das duas aplicações *Apache Jmeter* a variar para cada teste entre [10, 30, 60, 100, 150, 200, 500, 1000]. Cada pedido criado, gera um número entre 0 e o número máximo de políticas, e utiliza esse número para efetuar o pedido de acesso, evitando assim exercitar sempre a mesma política. Cada resultado apresentado nas tabelas é resultado de uma média de três execuções.

Ao comparar os valores das tabelas 15, 16, 17 e 18 conclui-se que a diferença de desempenho do módulo de políticas causado pelo número de políticas entre 10 e 10000 é praticamente nulo. Na tabela 18 para 500 clientes com 10000 políticas, temos uma média de 27,83ms, o que é até mais baixo que para 10 políticas, com o valor de 28,34ms como se observa na tabela 15. Para 10000 políticas e 1000 clientes, tabela 18 o tempo de resposta alcança valores fora do normal obtendo uma média de 130578,37ms, neste caso o número de clientes já é muito alto e não é relevante para o cenário em estudo.

	10 Políticas	
	Média	Desvio padrão
10	0,61	0,55
30	3,17	2,75
60	3,16	2,86
100	5,29	5,74
150	8,08	11,75
200	10,82	16,55
500	28,34	28,51
1000	57,08	34,67

Tabela 15: Média e desvio padrão em ms para testes de carga ao módulo de políticas, para 10 políticas guardadas, variando entre 1 e 1000 clientes.

	100 Políticas	
	Média	Desvio padrão
10	0,66	0,54
30	3,61	3,13
60	3,43	2,98
100	5,78	6,37
150	8,78	12,39
200	11,74	18,21
500	30,31	34,95
1000	61,19	37,79

Tabela 16: Média e desvio padrão em ms para testes de carga ao módulo de políticas, para 100 políticas guardadas, variando entre 1 e 1000 clientes.

	1000 Políticas	
	Média	Desvio padrão
10	0,62	0,57
30	3,12	2,81
60	3,09	2,87
100	5,18	5,73
150	7,92	10,94
200	10,67	16,23
500	27,89	30,93
1000	55,83	32,51

Tabela 17: Média e desvio padrão em ms para testes de carga ao módulo de políticas, para 1000 políticas guardadas, variando entre 1 e 1000 clientes.

	10000 Políticas	
	Média	Desvio padrão
10	0,59	0,54
30	3,09	2,87
60	3,15	2,77
100	5,20	5,74
150	7,91	10,96
200	10,68	16,28
500	27,83	29,94
1000	130 578,37	326,48

Tabela 18: Média e desvio padrão em ms para testes de carga ao módulo de políticas, para 10000 políticas guardadas, variando entre 1 e 1000 clientes.

Capítulo 5

Conclusão e Trabalho Futuro

Esta dissertação apresentou o **SDAM**, uma aplicação que estende o **HLF** com a capacidade de granularidade flexível e fina para as políticas de controlo de acesso em dados *on-chain*. Trabalhando como um proxy, o **SDAM** fica entre cada *peer* **HLF** e a sua base de dados *CouchDB*. Exporta a mesma API REST do *CouchDB* tornando a sua utilização transparente, assim, evitando a necessidade de mudar a implementação do **HLF**. As políticas de acesso são estendidas com a definição opcional de transformações de dados obrigatórias para acesso a dados. Além disso, as organizações podem optar por usar controlo de acesso flexível e ainda interagir perfeitamente com aqueles que optam por não fazê-lo, diminuindo assim a barreira para adoção. A campanha de avaliação validou o fluxo de trabalho de consulta e avaliou o *overhead* e a escalabilidade da solução: a penalidade média do tempo de resposta é de 92 milissegundos, resultando assim num impacto pequeno e aceitável no desempenho, considerando a funcionalidade adicionada.

Como trabalho futuro identificam-se dois pontos de trabalho. O primeiro ponto de trabalho compreende a implementação do mecanismo leve de sincronização de políticas entre instâncias do **SDAM**. O segundo ponto compreende o mecanismo de gestão de chaves associado ao **HLF**, de suporte à cifragem de informação extra-organização.

Bibliografia

Wei Cai, Zehua Wang, Jason B Ernst, Zhen Hong, Chen Feng, and Victor CM Leung. Decentralized applications: The blockchain-empowered software system. *IEEE Access*, 6:53019–53033, 2018.

Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

David W. Chadwick, Gansen Zhao, Sassa Otenko, Romain Laborde, Linying Su, and Tuan A. Nguyen. Building a modular authorization infrastructure. 2006.

Ting Chen, Zihao Li, Yuxiao Zhu, Jiachi Chen, Xiapu Luo, John Chi-Shing Lui, Xiaodong Lin, and Xiaosong Zhang. Understanding ethereum via graph analysis. *ACM Transactions on Internet Technology (TOIT)*, 20(2):1–32, 2020.

Aaron Elliott and Scott Knight. Role explosion: Acknowledging the problem. In *Software Engineering Research and Practice*, 2010.

Ehsan Meamari H. Guo and Chien-Chung Shen. Multi-authority attribute-based access control with smart contract. *Proceedings of the 2019 International Conference On Blockchain Technology.*, 2019.

et al. Hu, Vincent C. Attribute-based access control. *Computer 48.2 : 85-88.*, 2015.

Hyperledger. Hyperledger architecture, volume ii.

F. Coelho J. Vinagre J. Vinagre P. Bastos J. Parente, A.N. Alonso. Flexible fine-grained data access management for hyperledger fabric. *International Conference on Blockchain Computing and Applications – BCCA 2022*, 2022.

Ravi Sandhu Jin, Xin and Ram Krishnan. Rabac: role-centric attribute-based access control. *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security. Springer, Berlin, Heidelberg.*, 2012.

et al. Khan, Muhammad Yasar. An extended access control model for permissioned blockchain frameworks. *Wireless Networks (10220038) 26.7*, 2020.

Stephanie Lo and J Christina Wang. Bitcoin as money? 2014.

Microsoft. URL https://docs.microsoft.com/en-us/windows/win32/secauthz/.

Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

OASIS Open. Oasis extensible access control markup language (xacml). http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, 2005. Acessado em 11-2021.

Carroline Dewi Puspa Kencana Ramli, Hanne Riis Nielson, and Flemming Nielson. The logic of xacml. *Science of Computer Programming*, 2014. URL https://www.sciencedirect.com/science/article/pii/S0167642313001238.

Ravi S. Sandhu. Role-based access control. *Advances in computers. Vol. 46. Elsevier, 237-286.*, 1998.

et al. Shalaby, Salma. Performance evaluation of hyperledger fabric. *IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT).*, 2020.

Shuai Wang, Liwei Ouyang, Yong Yuan, Xiaochun Ni, Xuan Han, and Fei-Yue Wang. Blockchain-enabled smart contracts: architecture, applications, and future trends. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(11):2266–2277, 2019.

Hao Xia, Milind Dawande, and Vijay S. Mookerjee. Role refinement in access control: Model and analysis. *INFORMS J. Comput.*, 26:866–884, 2014.

Mirei Yutaka, Yuanyu Zhang, Masahiro Sasabe, and Shoji Kasahara. Using ethereum blockchain for distributed attribute-based access control in the internet of things. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

Este trabalho é financiado por fundos nacionais através da FCT – Fundação para a Ciência e a Tecnologia no âmbito do Projeto LA/P/0063/2020.