

UNIVERSIDADE FEDERAL DE SANTA CATARINA

**PROPOSTA DE UMA APLICAÇÃO PARA AGREGAÇÃO E
PUBLICAÇÃO DE METADADOS**

Luiz Felipe Correa Chiaradia

Florianópolis - SC

2014/1

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

**PROPOSTA DE UMA APLICAÇÃO PARA AGREGAÇÃO E
PUBLICAÇÃO DE METADADOS**

Luiz Felipe Correa Chiaradia

Trabalho de conclusão de curso a ser apresentado como requisito para a obtenção do grau de Bacharel em Sistemas de Informação pela Universidade Federal de Santa Catarina - UFSC

Orientadora:

Profa. Dra. Carla Merkle Westphall

Florianópolis – SC

2014/1

Luiz Felipe Corrêa Chiaradia

PROPOSTA DE UMA APLICAÇÃO PARA AGREGAÇÃO E PUBLICAÇÃO DE METADADOS

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Profa. Dra. Carla Merkle Westphall

Orientadora

Banca Examinadora:

Prof. Dr. Carlos Becker Westphall

Jorge Werner

Rafael Weingärtner

Resumo

Computação em nuvem até poucos anos atrás era tida somente como uma tendência, uma ideia de que ninguém mais precisaria “instalar programas” em seus computadores, economizando, assim, recursos de memória e processamento. Hoje essa concepção já é uma realidade, na qual grandes empresas vendem seus serviços em nuvem.

Com a constante necessidade de redução de custos e maior flexibilização, aplicações e sistemas de hardware vêm sendo disponibilizados na forma de serviços em ambientes de computação em nuvem.

Com o objetivo de controlar o acesso a determinados conteúdos, surge a imprescindibilidade de que cada usuário seja identificado e que sejam desenvolvidos sistemas que realizem todo o processo de gerenciamento dessas identidades digitais. Nesse contexto, o modelo de gerenciamento de identidades federadas, assim como outros, é desenvolvido.

O compartilhamento de informações entre diferentes provedores de uma federação é uma tarefa bastante complexa, devido aos acordos de privacidade existentes, além das convenções utilizadas em cada uma delas. O consórcio Internet2, visando facilitar essa tarefa, desenvolveu um framework em Java que permitiria o desenvolvimento de aplicações com a finalidade de realizar o processamento dos metadados de cada federação.

Este trabalho apresenta um estudo detalhado do framework Metadata Aggregator e uma proposta de aplicação que opere de forma semelhante, objetivando o download, agregação e publicação de metadados obtidos de provedores localizados em federações.

Palavras-chave: Computação em Nuvem, Gerenciamento de Identidade, Internet2, Metadata, Aggregator, Java, metadados.

SUMÁRIO

LISTA DE FIGURAS	3
1. <i>INTRODUÇÃO</i>	5
1.1 MOTIVAÇÃO.....	5
1.2 OBJETIVOS.....	5
1.2.1 Objetivo Geral.....	5
1.2.2 Objetivos Específicos	6
2. <i>COMPUTAÇÃO EM NUVEM E SEGURANÇA DA INFORMAÇÃO</i>	7
2.1 COMPUTAÇÃO EM NUVEM	7
2.1.1 CARACTERÍSTICAS ESSENCIAIS PARA COMPUTAÇÃO EM NUVEM.....	7
2.1.2 MODELOS DE IMPLANTAÇÃO	8
2.1.3 TIPOS DE SERVIÇO EM COMPUTAÇÃO EM NUVEM	9
2.2 SEGURANÇA DA INFORMAÇÃO	10
2.2.1 ARQUITETURA DE SEGURANÇA OSI	11
2.2.2 SEGURANÇA DA INFORMAÇÃO EM COMPUTAÇÃO EM NUVEM	12
3. GERENCIAMENTO DE IDENTIDADES	14
3.1 IDENTIDADE DIGITAL.....	14
3.2 GERENCIAMENTO DE IDENTIDADES.....	15
3.2.1 SISTEMAS DE GERENCIAMENTO DE IDENTIDADE.....	15
3.2.2 SISTEMAS DE GERENCIAMENTO DE IDENTIDADE BASEADOS NO MODELO DE IDENTIDADE FEDERADA	19
3.2.3 SHIBBOLETH.....	21
3.2.4 METADADOS	23
3.2.4.1 PROVEDOR DE METADADOS – IDP	25
3.2.4.2 PROVEDOR DE METADADOS – SP	27
4. <i>DESENVOLVIMENTO PRÁTICO</i>	32

4.1	PROPOSTA.....	32
4.2	SPRING FRAMEWORK.....	33
4.3	METADATA AGGREGATOR.....	34
4.3.1	CONFIGURAÇÕES.....	38
4.4	VISÃO GERAL DA APLICAÇÃO.....	41
4.5	ESTÁGIO DE LEITURA E AGREGAÇÃO.....	43
4.6	ASSINATURA XML.....	45
4.7	DOWNLOAD E UPLOAD DOS METADADOS.....	49
4.8	RESULTADOS.....	52
5.	<i>CONCLUSÕES E TRABALHOS FUTUROS</i>	55
6.	<i>REFERÊNCIAS BIBLIOGRÁFICAS</i>	57
	ANEXO(S) E APÊNDICE(S).....	59
A.1	ARTIGO.....	59

LISTA DE FIGURAS

Figura 1: Tipos de serviço em computação em Nuvem. [Criado pelo autor]	9
Figura 2: Modelo tradicional de sistema de gerenciamento de identidades [FELICIANO, 2011]	16
Figura 3: Modelo centralizado de sistema de gerenciamento de identidades [FELICIANO, 2011]	16
Figura 4: Modelo de identidade federada de sistema de gerenciamento de identidades [FELICIANO, 2011]	17
Figura 5: Modelo centrado ao usuário de sistema de gerenciamento de identidades [FELICIANO, 2011]	17
Figura 6: Interação entre os elementos de uma federação [SWITCH AAI-Federation]	19
Figura 7: Funcionamento do Shibboleth [SWITCH AAI-Federation]	21
Figura 8: Exemplo utilizando o elemento EntityDescriptor	23
Figura 9: Elementos referentes aos papéis – Provedor de Identidade	23
Figura 10: Elementos referentes aos papéis – Provedor de Serviço	23
Figura 11: Configuração dos elementos de organização e contato	24
Figura 12: Provedor de Metadados por Encadeamento	25
Figura 13: Provedor de Metadados via Arquivo em Máquina	25
Figura 14: Provedor de Metadados via HTTP	25
Figura 15: Provedor de Metadados via HTTP com backup local	26
Figura 16: Provedor de Metadados XML	27
Figura 17: Provedor de Metadados por Encadeamento	27
Figura 18: Filtro de assinatura	28
Figura 19: Filtro de partes confiáveis	28
Figura 20: Lista negra	28
Figura 21: Filtro de atributo obrigatório	29
Figura 22: Filtro de papéis	29
Figura 23: Filtro de atributos	29
Figura 24: Configuração do provedor de metadados XML	30
Figura 25: Modelo da aplicação proposta	32

Figura 26: Visão geral da organização dos módulos do Spring Framework [SPRING, 2014]	33
Figura 27: Estágios em um pipeline. Adaptado de [SHIBBOLETH, 2014]	35
Figura 28: Web Service. Adaptado de [SHIBBOLETH, 2014]	36
Figura 29: Diagrama de Classes das funções disponíveis para serem utilizadas em um Web Service	37
Figura 30: Elemento do arquivo de metadados que não será removido	39
Figura 31: Exemplo de configuração de um Pipeline (arquivo XML)	39
Figura 32: Parte do código em Java para o arquivo de configurações	40
Figura 33: Modelo da aplicação proposta	41
Figura 34: Workspace do projeto na IDE Eclipse	41
Figura 35: Classe LeArquivoXML.java	42
Figura 36: Classe AgregadorXML.java	43
Figura 37: Laço responsável pela agregação dos elementos do arquivo	44
Figura 38: Geração do repositório JKS	45
Figura 39: Obtenção do valor do alias do certificado digital	45
Figura 40: Método para obter a chave privada	46
Figura 41: Verificação da validade do certificado	46
Figura 42: Seleção do certificado	47
Figura 43: Arquivo XML assinado	47
Figura 44: Método para efetuar a assinatura XMLDSig	48
Figura 45: Código da classe responsável por efetuar o download dos arquivos	49
Figura 46: Código da classe responsável por efetuar o upload dos arquivos	50
Figura 47: Código da classe responsável por efetuar o upload dos arquivos	51
Figura 48: Console da IDE Eclipse durante a execução do código	52
Figura 49: Arquivo disponibilizado	52
Figura 50: Arquivos no diretório local	53

1. INTRODUÇÃO

1.1 MOTIVAÇÃO

A migração para os serviços de nuvem permitem ao usuário uma maior flexibilização e uma considerável redução de custos, principalmente relacionados à infraestrutura. Entretanto certas questões devem ser abordadas durante esse processo, principalmente relacionadas à segurança da informação, visto a necessidade de definir quem poderá ter acesso ou não aos arquivos disponibilizados e que tipo de acesso, ou seja, quais permissões serão concedidas.

A importância do gerenciamento de identidade e acesso em um ambiente de computação em nuvem cresce na medida em que serviços que necessitem utilizar autenticação e controle de acesso a usuários surgem. Uma gestão na qual todos os usuários possuem identidades e papéis definidos é necessária a fim de que falhas, como quebra de sigilo por exemplo, sejam minimizadas durante o processo de armazenamento e compartilhamento de informações na nuvem.

A adoção do modelo de federações e conseqüentemente de sistemas de gerenciamento de identidade baseados nesse modelo já é uma realidade e utilizados em larga escala. Entretanto, existem obstáculos quando se deseja realizar a compartilhamento de informações entre diversas federações, devido a enorme demanda de tempo para negociação das políticas de privacidade e liberação de atributos, além das convenções utilizadas em cada uma delas em seus arquivos de metadados.

Diante desse contexto, o consórcio Internet2 desenvolveu um framework em Java para o processamento de metadados provindos pelos provedores das federações. Entretanto, após a realização de buscas pela Internet e nos fóruns de discussão do produto, foi possível constatar que a biblioteca é fracamente documentada e sua utilização é relativamente desconhecida por parte do grande público. Sendo assim, a solução proposta é realizar um estudo do framework, fundamentando-o e a partir desse estudo, uma aplicação que opere de maneira semelhante será implementada e descrita.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

O objetivo geral deste trabalho consiste em realizar um estudo do framework Metadata Aggregator e, a partir desse estudo, propor uma aplicação com funcionalidades semelhantes.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho que podem ser citados são:

- Descrever os conceitos de identidade digital e sistema de gerenciamento de identidades;
- Realização de um estudo do framework Metadata Aggregator;
- Propor uma aplicação cujas funcionalidades sejam semelhantes às do framework.

2. COMPUTAÇÃO EM NUVEM E SEGURANÇA DA INFORMAÇÃO

2.1 COMPUTAÇÃO EM NUVEM

Computação em Nuvem pode ser entendida como o ato de fornecer recursos computacionais compartilhados, que vão desde o armazenamento de simples arquivos de textos na Internet até a terceirização de toda a infraestrutura de tecnologia de informação de uma empresa. Segundo [TAURION, 2009], o termo surgiu pela primeira vez em 2006, durante uma palestra de um representante da Google, na qual eles explicavam a sua forma de gerenciamento de data centers. O nome evoca os diagramas de rede onde a Internet é representada por uma nuvem.

De acordo com [MELL et al., 2010] em sua publicação para o NIST, computação em nuvem pode ser definido como “um modelo para acesso a rede sob demanda, ubíquo e conveniente para um conjunto compartilhado de recursos computacionais configuráveis que podem ser rapidamente provisionados com mínimo esforço de gerenciamento ou interação com o provedor de serviços”.

2.1.1 CARACTERÍSTICAS ESSENCIAS PARA COMPUTAÇÃO EM NUVEM

Em [MELL, 2010] são definidas cinco características essenciais para a computação em nuvem, que em conjunto a distinguem de outros paradigmas computacionais. Essas particularidades, além de serem utilizadas para distinção e descrição, podem ser consideradas como as vantagens que o serviço oferece em relação aos outros:

- **Ampla acesso a rede:** por meio da rede, o usuário pode acessar qualquer recurso por mecanismos que fazem uso de diferentes arquiteturas em um mesmo nó computacional, ou seja, plataformas heterogêneas (por exemplo, celulares e notebooks). É interessante ressaltar que a interface de acesso à nuvem, adota o princípio da portabilidade, no qual não existe a necessidade do usuário alterar o seu ambiente de trabalho.

- **Rápida elasticidade ou expansão:** graças à utilização da virtualização, recursos podem ser alocados de forma rápida, automaticamente em alguns casos, caso haja a necessidade, dando a impressão aos usuários de que dispõem de recursos ilimitados.
- **Serviço de mensuração:** a utilização e otimização dos recursos podem ser monitorados automaticamente por sistemas de nuvem, garantindo a transparência para o usuário e provedor do serviço. A qualidade do serviço (*Quality of Service*) é garantida baseado no SLA (*Service Level Agreement*) contratado.
- **Pool de recursos:** o provedor organiza seus recursos computacionais agrupando-os para atender a diversos usuários usando um modelo multi-inquilino, com recursos físicos e virtuais atribuídos e ajustados de acordo com a demanda. Apesar de ser haver controvérsias, os usuários não precisam saber da localização física dos recursos computacionais, que incluem: processamento, memória, máquinas virtuais, armazenamento, entre outros.
- **Autosserviço sob demanda:** o usuário pode personalizar o seu ambiente computacional de acordo com a sua necessidade, ou seja, ele pode adquirir um recurso computacional, como armazenamento na rede ou modificar a configuração da rede.

2.1.2 MODELOS DE IMPLANTAÇÃO

Conforme a definição do NIST, os modelos de implantação podem ser classificados em público, privado, comunitário e híbrido. A escolha por qual será utilizado é feito baseado nas necessidades das aplicações que serão implementadas na nuvem, restrição ou abertura de acesso, do tipo de informação e de que forma ela será vista.

- **Nuvem privada:** são construídas exclusivamente para um único domínio, o qual terá total controle sobre a infraestrutura de nuvem e o que será implementado. Sendo assim, a própria organização deverá garantir a segurança e a confiabilidade do ambiente, visto que toda a infraestrutura será de responsabilidade dele.
- **Nuvem pública:** a infraestrutura da nuvem é utilizada pelo público em geral podendo ser acessada por qualquer usuário. Neste modelo, é comum uma empresa terceirizada fornecer os serviços de nuvem para seus clientes, como o *Amazon Elastic Compute Cloud (EC2)* [AMAZON, 2013].

- **Nuvem comunitária:** a infraestrutura de nuvem é utilizada por diversas organizações, podendo existir localmente ou remota. A administração é normalmente compartilhada entre as organizações ou feita por terceiros.
- **Nuvem híbrida:** a infraestrutura de nuvem é uma combinação de duas ou mais nuvens, podendo ser pública, privada ou comunitária, mesclando os recursos de cada uma, por meio da portabilidade de serviços e possibilitando o atendimento de demandas em que haja flutuações rápidas na necessidade de recursos (por exemplo, um nuvem privada pode utilizar uma quantidade de recursos de uma nuvem pública para uma determinada transação) devido a sua escalabilidade dinâmica.

2.1.3 TIPOS DE SERVIÇO EM COMPUTAÇÃO EM NUVEM

O serviço de mais alto nível disponibilizado na nuvem é representado pelo *Software as a Service* (SaaS), concebendo as aplicações completas que são oferecidas aos usuários. Exemplos de SaaS que podem ser citados são os serviços oferecidos pela Microsoft, Office 360 [MICROSOFT, 2013], e ADOBE, que terão versões para Internet dos seus programas mais famosos, como o Photoshop, por exemplo [ADOBE, 2013].

Um alto nível de integração para implementação e teste de aplicações é oferecido pelo PaaS. O serviço fornece todas as primitivas necessárias para o desenvolvimento (sistema operacional, ferramentas de desenvolvimento e colaboração) e suporte para disponibilização do serviço *online*, facilitando e agilizando o processo de desenvolvimento das aplicações destinadas aos usuários da nuvem.

Pode-se citar como exemplo de PaaS o Windows Azure [MICROSOFT, 2013], que oferece uma plataforma para desenvolvimento de aplicações em Ruby, Java e PHP e banco de dados, além da infraestrutura dos *datacenters* da Microsoft.

Baseado em técnicas de virtualização, o serviço IaaS é responsável por prover toda a infraestrutura de computação para o SaaS e PaaS. Possui algumas singularidades como uma interface de administração, na qual o cliente terá controle total da infraestrutura por meio de acesso remoto, e suporte para mudanças nos equipamentos de forma simples.

Um exemplos de IaaS que pode ser citado é o Amazon EC2 (*Elastic Cloud Computing*) [AMAZON, 2013].

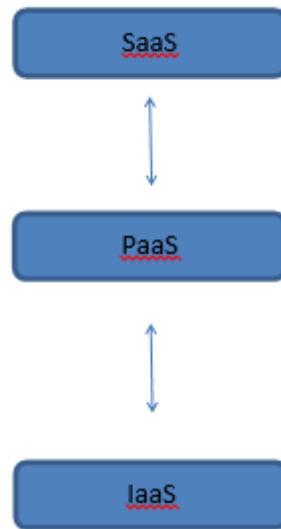


Figura 1: Tipos de serviço em computação em Nuvem. [Criado pelo autor]

2.2 SEGURANÇA DA INFORMAÇÃO

A palavra informação, de acordo com a norma NBR ISO/IEC 27002 [ABNT, 2005], é definida como “um ativo que, como qualquer outro ativo importante para os negócios, tem um valor para a organização e conseqüentemente necessita ser adequadamente protegido.”

Segurança da Informação pode ser definida como um processo de proteção de informações armazenadas em computadores para que sejam mantidos os seguintes requisitos de integridade, confidencialidade e disponibilidade das informações. Portanto, a segurança da informação é imprescindível para qualquer indivíduo, que deseja armazenar dados particulares, ou empresa, que deseja armazenar informações de caráter estratégico, por exemplo.

Conforme já mencionado anteriormente, alguns requisitos básicos devem ser respeitados para que se possa garantir a segurança da informação [ABNT NBR ISO/IEC 27002, 2005]:

- **Integridade:** garante a modificação da informação somente por partes autorizadas;
- **Confidencialidade:** garante que a informação só poderá ser acessada por partes autorizadas;

- **Disponibilidade:** informação disponível para acesso no momento desejado sem nenhum tipo de modificação. Também conhecido como continuidade do serviço;
- **Autenticidade:** está ligado ao fato da informação que esteja sendo trafegada seja de fato originada do proprietário a ela relacionado.

É importante ressaltar que aspectos humanos e tecnológicos devem ser levados em consideração quando se pretende garantir os requisitos de segurança da informação. Garantindo-os, alguns benefícios como maior controle de recursos e aumento de produtividade dos usuários por meio de um ambiente mais organizado. [FRANCISCO, 2004].

2.2.1 ARQUITETURA DE SEGURANÇA OSI

Com o surgimento das redes de computadores e posteriormente as aplicações cliente/servidor, diversos modelos foram desenvolvidos com o objetivo de definir padrões para a comunicação entre sistemas. Dentro os quais, o principal a ser citado é o Modelo OSI (*Open System Interconnection Model*), que define diretrizes genéricas para construção de redes de computadores, dividindo-as em sete camadas, sendo elas: física, dados, rede, transporte, sessão, apresentação e aplicação.

Neste mesmo contexto, baseando-se nas recomendações X.800 e RFC 2828, surgiu a Arquitetura OSI para Segurança da Informação, cuja essência se baseia nos anseios das organizações possuírem políticas de segurança e serviços que permitam avaliar todos os aspectos relacionados a segurança das suas informações. Estes serviços se resumem em:

- **Autenticação:** confirma se a identidade de uma ou mais entidades conectadas é verdadeira, ou seja, é a identificação dos envolvidos nas trocas de informação.
- **Integridade:** garantia de que a informação não será alterado durante o seu tráfego.
- **Confidencialidade:** garantia de que somente as partes autorizadas terão acesso para leitura e utilização das informações transmitidas na rede.
- **Não-repúdio:** garantia de que a fonte não poderá negar que executou determinada ação durante o processo de troca de informações.
- **Controle de acesso:** protege contra o uso não autorizado de recursos.

Vale lembrar de que não existe um mecanismo único que garanta todos os serviços citados, portanto sempre haverá a necessidade da utilização de um conjunto de mecanismos para solucionar um determinado tipo de problema.

2.2.2 SEGURANÇA DA INFORMAÇÃO EM COMPUTAÇÃO EM NUVEM

A computação em nuvem vem crescendo ao longo dos últimos anos e trouxe consigo uma enorme quantidade de benefícios, dentro os quais a possibilidade de acessar qualquer tipo de dado em qualquer lugar, desde que haja conexão com a internet, e diminuição de gastos com infraestrutura. No entanto, todos estes avanços criaram novas vulnerabilidades e novos desafios que devem ser vencidos objetivando a garantia da segurança da informação armazenada e processada no ambiente de nuvem computacional.

Visto que a mudança de modelo é um processo complexo, a transição pode afetar as políticas de segurança de uma determinada empresa ou até mesmo de um usuário de nuvem pública, por exemplo. Sendo assim, o investimento em segurança deve ser feito, pois grandes concentrações de dados, são alvos atraentes para ataques. Além disso, é interessante ressaltar que as defesas em nuvem podem ser mais robustas, escaláveis e ter uma relação custo-benefício mais atrativa.

No ano de 2013, a Cloud Security Alliance (CSA) publicou um relatório denominado The Notorious Nine: Cloud Computing Top Threats in 2013 [CSA, 2013], o qual listava as principais ameaças, afirmando que o componente central para gerenciamento dos riscos em computação em nuvem é entendendo seus principais indícios. Dentre eles, pode-se destacar:

- **Violação de dados (*data breaching*):** considerada a ameaça mais severa no ano de 2013, consiste em um incidente no qual dados protegidos e/ou confidenciais, como números de cartões de crédito, são acessados por partes consideradas não autorizadas. A CSA aconselha o uso de encriptação nos dados, para o caso de uma violação ocorrer, entretanto, caso a chave de encriptação seja extraviada, os dados serão perdidos de qualquer maneira.
- **Perda de dados (*data loss*):** dados salvos na nuvem podem ser perdidos por diversas razões, desde ataques maliciosos até acidentes naturais que de alguma forma

danifiquem a infraestrutura na qual o serviço de nuvem está instalado. Esse tipo de ameaça vai de encontro a serviços de segurança como não-repúdio e disponibilidade. Para evitar esse tipo de ameaça, a CSA recomenda a criação de backups offline, entretanto, essa medida aumenta os riscos para outros tipos problemas, como a violação de dados.

- **Desvio de conta ou serviço (*account or service traffic hijacking*):** ameaças como essa não são consideradas novas, mas a mudança para o paradigma da computação em nuvem criou novas possibilidades para usuários maliciosos. Parâmetros para acesso normalmente são reutilizados e isso pode ampliar os danos causados, visto que com essas informações, o atacante pode manipular dados de forma a influenciar pessoas próximas à vítima ou clientes, no caso de empresas, para que acessem conteúdos maliciosos.

No mesmo contexto, o *PCI Security Standards Council* [PCI, 2013] publicou um guia com requisitos específicos objetivando orientar as empresas que realizam transações com cartões de crédito a criarem ambientes virtualizados seguros. Mesmo o guia sendo direcionado para um público-alvo específico, qualquer usuário de computação em nuvem pode seguir a cartilha e descobrir quais as soluções disponíveis para mitigação dos riscos. As soluções que merecem destaque são as seguintes: criptografia de transmissão de dados, testar regularmente os processos e sistemas de segurança, restrição de acesso físico aos dados, atribuir uma identidade exclusiva para cada pessoa que tenha acesso ao ambiente e acompanhar e monitorar qualquer tipo de acesso com relação aos recursos da rede.

3. GERENCIAMENTO DE IDENTIDADES

Neste capítulo o conceito de identidade digital será fundamentado, bem como suas particularidades serão descritas. Na sequência, será abordado o funcionamento dos sistemas de gerenciamento de identidade, bem como sua caracterização a partir de determinados elementos e sua distinção em diferentes modelos, baseando-se na maneira que essas partes interagem entre si.

3.1 IDENTIDADE DIGITAL

Nos últimos anos, um novo ambiente competitivo para as diversas organizações governamentais e privadas vem se desenvolvendo e a tendência para negócios colaborativos está pressionando para uma mudança na forma na qual estas organizações são gerenciadas [Camarinha-Matos et al. 2008]. Nesse contexto, há a necessidade de que tanto usuários quanto prestadores de serviços criem suas identidades para que troquem informações de forma confiável.

De acordo com [WINDLEY, 2003] identidades digitais são coleções de dados que representam atributos (características associadas), preferências (desejos) e traços (características permanentes) de uma entidade.

Uma identidade digital irá conter dados que descreveram de forma singular algo (chamado de sujeito ou entidade na linguagem de identidade digital) ou alguém. Além disso, possuirá informações sobre as relações com outras entidades que o sujeito possui. Funciona como um meio de autenticação e autorização e são utilizadas em cenários como redes sociais, comércio eletrônico e *e-mail*.

O mundo da identidade digital possui a sua própria nomenclatura. Grande parte dos termos são usados de forma específica, apesar de familiares. Um sujeito ou uma entidade é qualquer coisa que realize requisições de acesso a uma fonte, podendo ser uma pessoa, organização, programa ou máquina. A fonte pode ser uma página na web ou dados em um banco de dados.

A gestão da identidade digital levanta novos problemas legais, como o roubo de identidade, por exemplo. No Brasil, o Marco Civil da Internet (oficialmente chamado de Lei

nº12.965, de 23 de abril de 2014) é a lei que regula o uso da Internet por meio da previsão de princípios, garantias, direitos e deveres para quem usa a rede, bem como da determinação de diretrizes para a atuação do Estado.

3.2 GERENCIAMENTO DE IDENTIDADES

Aspectos como globalização e o desenvolvimento em tempo exponencial de diversas áreas, principalmente quando se fala de tecnologia, impulsionaram um novo paradigma no qual organizações, sejam públicas ou privadas, necessitam trabalhar de forma colaborativa para adquirir vantagens competitivas. O amadurecimento da Internet, que alcançou índices mais elevados de desempenho e confiabilidade, permitiu a otimização do processo de comunicação e, conseqüentemente, uma maior eficácia nas interações por meio das redes colaborativas.

Uma rede colaborativa pode ser definida como “uma rede que consiste de várias entidades autônomas, heterogêneas e geograficamente distribuídas, que colaboram para encontrar um objetivo comum e compatível e cujas interações são suportadas pelas redes de computadores” [Camarinha-Matos et al. 2008]. A rede deve seguir uma série de requisitos de interoperabilidade, devido às diferenças entre os ambientes dos membros da rede, e segurança, para que as informações possam trafegar com confiança.

A autorização é feita por meio de uma terceira parte confiável, também chamada de autoridade de autenticação, que irá emitir credenciais para apresentação e posterior gerenciamento de identidade para controle de acesso. [JOSANG & POPE, 2005] definem o gerenciamento de identidades como um sistema integrado de políticas, processos de negócios e tecnologias que permitem às organizações o tratamento e manipulação de identidades de seus usuários.

3.2.1 SISTEMAS DE GERENCIAMENTO DE IDENTIDADE

Em qualquer cenário real, quem definirá quais informações podem ou não serem reveladas é o próprio indivíduo, enquanto no mundo digital, essa tarefa é desempenhada por um sistema de gerenciamento de identidades, utilizado para administrar, descobrir e trocar

informações visando garantir a identidade de uma entidade, permitindo relações de confiança entre os membros de uma rede colaborativa.

Neste trabalho, a caracterização de um sistema de gerenciamento de identidades será a utilizada em [Bhargav-Spantzel et. al 2007]:

- **Usuário:** aquele que deseja acessar algum serviço.
- **Identidade:** nome, filiação, data de nascimento, ou seja, qualquer atributo do usuário.
- **Provedor de Identidades (*Identity Provider* – **IdP**):** após todo o processo de autenticação, o usuário receberá uma credencial, reconhecida pelos provedores de serviço. O IdP é responsável pelo processo de autenticação.
- **Provedor de Serviços (*Service Provider* - **SP**):** após a verificação da identidade e comprovação de todos os atributos necessários para o acesso, o SP oferecerá recursos para o usuário autorizado. O SP é responsável pelo processo de autorização.

Além de sua caracterização, os sistemas de gerenciamento de identidade seguem modelos que apresentam formas diferentes de interação e disposição dos elementos citados anteriormente. Seus modelos são classificados em tradicional, centralizado, centrado no usuário e centralizado.

- **Modelo tradicional:** os provedores de serviço e identidade atuam juntos e cabe ao usuário criar uma identidade digital para cada SP que deseje interagir. Neste modelo, não existe o compartilhamento de identidades entre diferentes provedores de serviço. Vale ressaltar que este modelo pode ser custoso tanto para usuários quanto para provedores, pelo simples fato de ser extremamente tedioso para o usuário fornecer diversas vezes as mesmas informações e, assim, diminuindo a fidelidade das informações (atributos) que não consideradas essenciais para o provedor.



Figura 2: Modelo tradicional de sistema de gerenciamento de identidades [FELICIANO, 2011]

- **Modelo centralizado:** fundamentado no compartilhamento das identidades dos usuários entre provedores de serviços e no conceito de autenticação única (*Single Sign-on – SSO*). Em [Bhargav-Spantzel et. al 2007], afirma-se que o serviço *Microsoft Passport Network* foi o precursor deste modelo, dando aos seus usuários a possibilidade de interagir com diversos provedores de serviços autenticando-se apenas uma única vez.

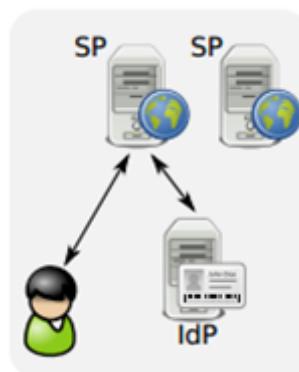


Figura 3: Modelo centralizado de sistema de gerenciamento de identidades [FELICIANO, 2011]

- **Modelo de identidade federada:** fundamentado sobre a distribuição da tarefa de autenticação dos usuários por múltiplos provedores de identidades, estando estes dispostos em diferentes domínios administrativos. Sua abordagem visa a otimização das trocas de informações relacionadas a identidade por meio de relações de confiança construídas nas federações [Camenisch e Pfitzmann 2007].

Por evitar que os usuários tenham que lidar com diversas identidades e passar diversas vezes pelo processo de autenticação, o modelo de identidades federadas consegue

oferecer inúmeras facilidades aos seus usuários. O projeto *Shibboleth*, que será descrito posteriormente, segue o modelo de gerenciamento de identidades federadas (*Federated Identity Management – FIM*).

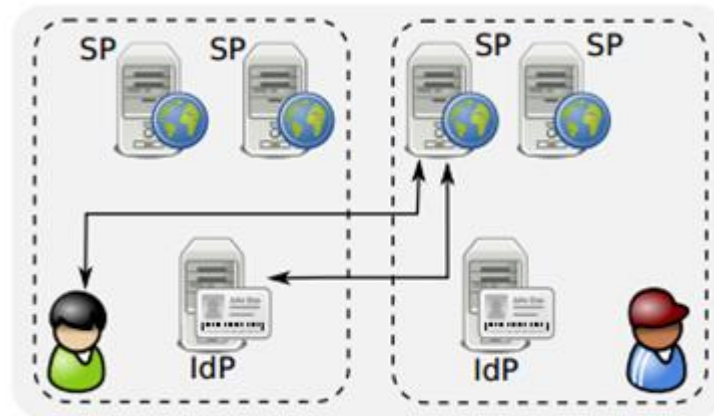


Figura 4: Modelo de identidade federada de sistema de gerenciamento de identidades [FELICIANO, 2011]

- **Modelo centrado ao usuário:** objetiva dar ao usuário o total controle de suas identidades digitais. [Josang e Pope 2005] propõe que a identidade de um usuário, destinada a diferentes provedores, seja armazenada em dispositivo físico, como um *smartcard*, por exemplo. A autenticação é feita neste dispositivo.

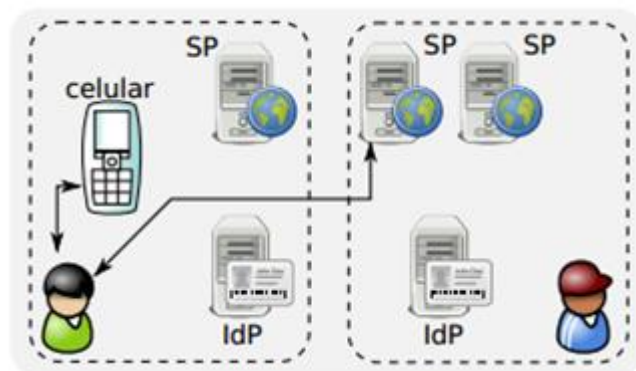


Figura 5: Modelo centrado ao usuário de sistema de gerenciamento de identidades [FELICIANO, 2011]

Um sistema de gerenciamento de identidades deve atender a uma série de requisitos objetivando a garantia de uma melhor experiência de uso para seus usuários, sem, de forma alguma, afetar a segurança de seus dados pessoais. Em [Damiani et al. 2003] é listado um conjunto de requisitos.

- **Interoperabilidade:** as identidades devem ser apresentadas em um formato comum de forma a serem validadas em múltiplos domínios administrativos.
- **Mecanismo para revogação de identidades:** aos usuários deve ser permitida a possibilidade de gerenciar as informações contidas em suas identidades, bem como revogá-las.
- **Gerenciamento de confiança:** identidades emitidas em um domínio são aceitas em outros, pois são criadas relações de confiança entre os provedores de serviços e provedores de identidades.
- **Privacidade:** deve ser possível definir as preferências de privacidade sobre as informações pessoais presentes nas identidades.
- **Anonimato:** aos usuários deve ser garantido o direito de permanecerem anônimos.

3.2.2 SISTEMAS DE GERENCIAMENTO DE IDENTIDADE BASEADOS NO MODELO DE IDENTIDADE FEDERADA

Os sistemas de gerenciamento de identidade baseados no modelo de identidade federada possuem uma abordagem que visa à otimização das trocas de informações relacionadas à identidade através de relações de confiança construídas nas federações. Por meio de acordos entre as entidades é garantido o reconhecimento de identidades emitidas por um domínio em provedores de serviços de outros domínios, permitindo assim, a criação de Federações que permitem a redução de contratos bilaterais entre usuários e provedores de serviços.

Uma federação é composta por dois elementos, denominados provedor de serviço (SP) e provedor de identidade (IdP), e três atores em ação: usuários, provedores de recursos (aplicação com o SP instalado) e instituição do usuário. O componente *Discovery Device* (DS) centraliza as informações dos provedores de identidades e suas respectivas localizações, visto que em uma federação é permitido o acesso de usuários de outras instituições e esses componentes auxiliam no redirecionamento ao seu IdP. As interações entre os elementos de uma federação podem ser descritas em oito etapas:

- **Etapa 1:** usuário acessa o provedor de serviço.

- **Etapa 2:** o serviço apresenta as escolhas fornecidas pelo repositório centralizado DS (*Discovery Device*).
- **Etapa 3:** o usuário seleciona sua instituição de origem.
- **Etapa 4:** o usuário é redirecionado ao seu respectivo provedor de identidade (IdP).
- **Etapa 5:** o IdP realiza a autenticação do usuário.
- **Etapa 6:** o SP recebe a garantia de autenticação pelo IdP.
- **Etapa 7:** quando necessário, o SP solicita atributos adicionais ao IdP.
- **Etapa 8:** o provedor de serviço decide sobre as autorizações e disponibiliza o serviço para o usuário.

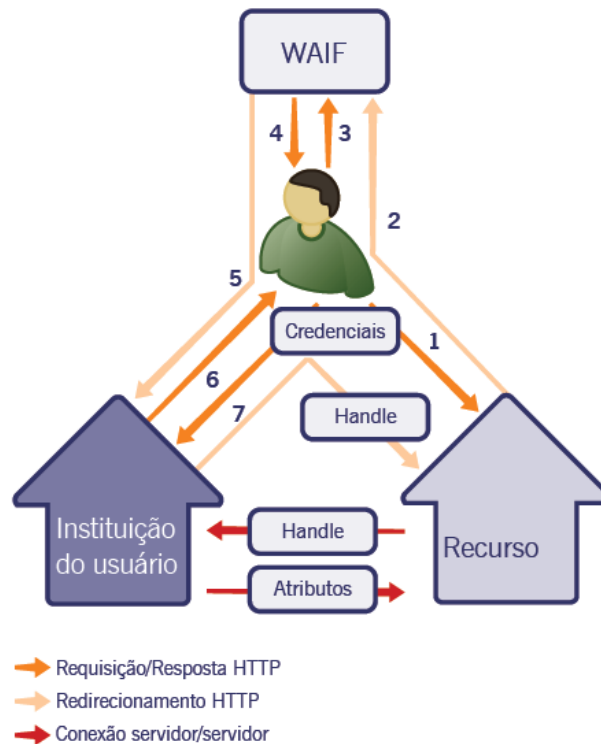


Figura 6: Interação entre os elementos de uma federação [SWITCH AAI-Federation]

Em 2005, a OASIS lançou um conjunto de especificações para definir uma infraestrutura para troca dinâmica de informações de segurança entre parceiros de negócios. A *Security Assertion Markup Language* (SAML) tem como núcleo uma gramática XML para representar informações de segurança na forma de asserções, bem como protocolos para requisições e envio dessas sessões [OASIS, 2005a]. A especificação define cinco componentes:

- **Papéis:** constituem no que cada entidade pode desempenhar na infraestrutura SAML, além disso, são apresentados metadados que os descrevem.
- **Perfis:** agregam protocolos e asserções em fluxos de dados específicos para prover funcionalidades, como o gerenciamento de identidades, por exemplo.
- **Asserções:** definidas por uma gramática XML, especificam o formato para representar informações de segurança acerca de um sujeito. Tais informações são atestadas por uma entidade denominada de *asserting part* (parte declarante) ou *SAML authority* (autoridade SAML) [OASIS 2005].
- **Protocolos:** são definidos em duas camadas, sendo a camada superior formada pelos esquemas XML das mensagens e a camada inferior composta de especificações de como usar protocolos subjacentes para transportar essas mensagens [OASIS 2005].
- **Transporte:** refere-se aos protocolos da camada inferior.

3.2.3 SHIBBOLETH

Fundamentado sobre padrões abertos como SAML e XML, a ferramenta Shibboleth foi criada para tratar de desafios relacionados ao gerenciamento de identidades e controle de acesso em instituições acadêmicas. Enfatiza a privacidade dos atributos do usuário, sendo que sua liberação para os provedores de serviço é controlada por uma política de privacidade definida pela instituição de origem e suas preferências pessoais. Segundo [CHADWICK 2009], todo o processo de autenticação é executado na instituição de origem do usuário, por meio de seu provedor de identidade, fazendo uso de mecanismos de autenticação presentes nessa instituição, podendo ser feita por nome de usuário e senha, X.509, etc.

Dentro de um ambiente federado, a padronização de atributos é fundamental, portanto o Shibboleth define uma forma padronizada para trocas de atributos, por meio das asserções, entretanto fica a cargo dos desenvolvedores a especificação dos atributos. Nesse contexto, foi proposto um conjunto padrão de atributos de identidades comuns para federações acadêmicas, sendo seis altamente recomendados, dez sugeridos e outros vinte e cinco opcionais. Este conjunto foi criado em [Internet2 2008, Wahl 1997, Smith 2000] e é denominado eduPerson.

Em [CHADWICK 2009], as asserções emitidas pelo provedor de identidades são assinadas por ele mesmo, permitindo ao provedor de serviços a verificação da autenticidade

delas. Sendo assim, a relação de confiança entre provedores de identidade e serviço será a garantia de que um usuário foi autenticado corretamente e que o seu respectivo IdP fornecerá corretamente os seus atributos.

O fluxo de funcionamento do *Shibboleth* é representado na Figura 4 e descrito a seguir:

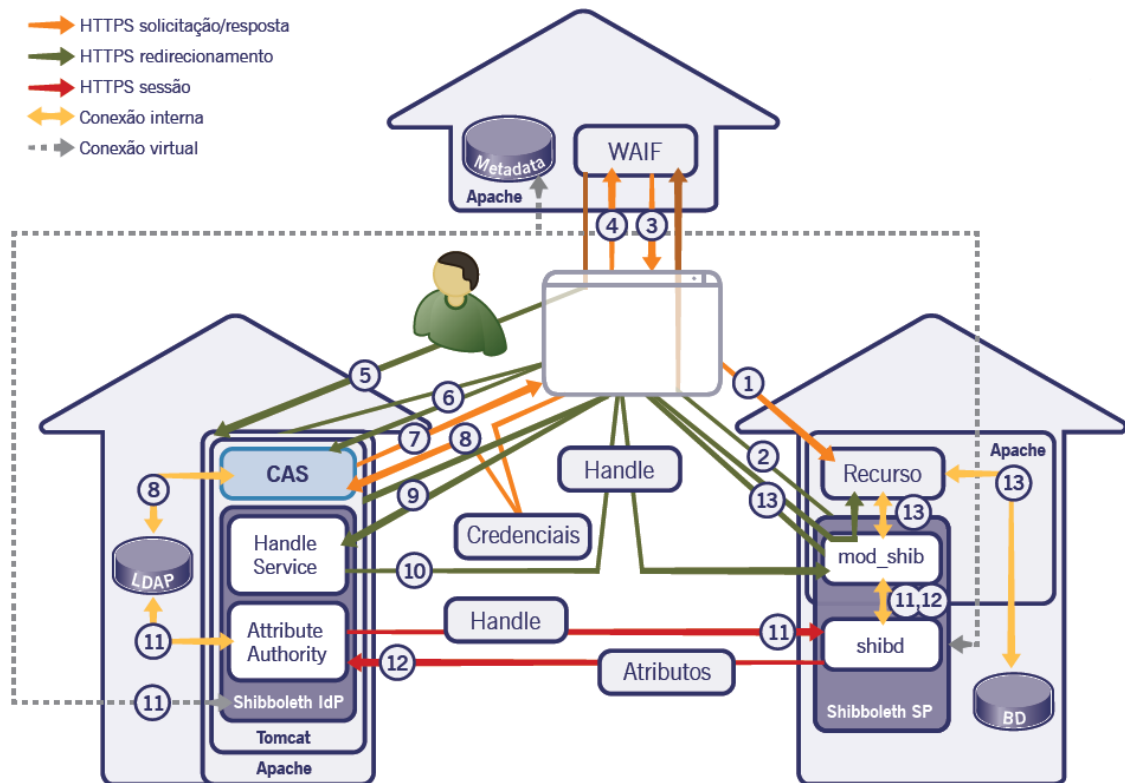


Figura 7: Funcionamento do Shibboleth [SWITCH AAI-Federation]

- **Passos 1, 2 e 3:** o usuário realiza a autenticação selecionando a sua instituição de origem.
- **Passos 4 e 5:** após o envio da requisição do usuário, o WAYF responde com um redirecionamento HTTP para o provedor de identidade do usuário.
- **Passos 6 e 7:** deverá inserir seus dados em um sistema de autenticação *Single Sign-On*. Uma vez que o usuário disponibiliza seus dados;
- **Passo 8:** é enviada uma solicitação para o sistema de autenticação (CAS), que irá verificar os dados no diretório LDAP.
- **Passos 9 e 10:** os provedores de serviço e identidade realizam as trocas dos *handles* para confirmar a autenticação do usuário.

- **Passos 11 e 12:** após a sessão HTTPS ser estabelecida entre o *shibd* e o *Attribute Authority*, é encaminhado os atributos permitidos para o provedor de serviço.
- **Passo 13:** o usuário recebe um *cookie* de sessão *Shibboleth* e é redirecionado para o recurso, que utilizará os atributos para prover um nível de autorização mais granular.

3.2.4 METADADOS

Em [RIBEIRO, 2005], metadados são definidos como descrições de dados armazenados em bancos de dados, ou seja, dados sobre dados a partir de um dicionário digital de dados. Sua principal finalidade é documentar e organizar de forma estruturada os dados das organizações com o objetivo de minimizar duplicação de esforços e facilitar a manutenção dos dados.

Metadados, no que diz respeito ao Shibboleth, referem-se a dados de configuração, comumente no formato XML, utilizados para permitir que provedores de serviço (SP) e identidade (IdP) se comuniquem. [INTERNET2, 2014]

Existem diversos esquemas de metadados definidos por diferentes softwares ou especificações, entretanto, o Shibboleth é baseado na especificação SAML 2.0 Metadata normatizada pela Oasis. De um modo geral, sua estrutura é composta por entidades ou grupos, papéis e contatos e organizações, conforme descrito a seguir:

- **Entidades ou grupos:** servidor rodando um software com o objetivo de realizar alguma tarefa, seja como SP ou IdP. Cada entidade possui um nome próprio, definido pelo elemento `entityID`, sendo de responsabilidade do usuário a definição do valor (URL) que será usado como `entityID`.

O grupos são enraizados pelos elementos `<md:EntityDescriptor>` ou `<md:EntitiesDescriptor>`, sendo o segundo mais utilizado devido a possibilidade de descrever uma grande quantidade de provedores de serviço ou identidade como uma unidade. Abaixo, segue um exemplo da configuração do item descrito:

```

<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
                    entityID="https://inf.ufsc.br/~luiz_fc/idp/shibboleth">
</md:EntityDescriptor>

```

Figura 8: Exemplo utilizando o elemento EntityDescriptor.

- **Papéis:** uma entidade é estruturada a partir de uma sequência de papéis, seguidas, ou não, por informações relacionadas ao contato/organização para fins de suporte. Esse elemento, compõe o conteúdo dos metadados que serão publicados e consumidos. Para o caso do Shibboleth, os papéis mais importantes tendem a ser `<md:IDPSSODescriptor>` e `<md:AttributeAuthorityDescriptor>` para os metadados de provedores de identidade (Figura 9) e `<md:SPSSODescriptor>` para os de provedores de serviço (Figura 10).

```

<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
                    entityID="https://inf.ufsc.br/~luiz_fc/idp/shibboleth">
  <md:IDPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol
    urn:oasis:names:tc:SAML:1.1:protocol urn:mace:shibboleth:1.0">
  </md:IDPSSODescriptor>
</md:EntityDescriptor>

```

Figura 9: Elementos referentes aos papéis – Provedor de Identidade.

```

<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
                    entityID="https://inf.ufsc.br/~luiz_fc/service/shibboleth">
  <md:SPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol
    urn:oasis:names:tc:SAML:1.1:protocol">
  </md:SPSSODescriptor>
</md:EntityDescriptor>

```

Figura 10: Elementos referentes aos papéis – Provedor de Serviço.

- **Contatos e organizações:** além dos papéis, o restante dos metadados acerca de uma entidade consistem dos elementos opcionais `<md:ContactPerson>` e `<md:Organization>`, responsáveis por informações como quem está por trás de uma organização e como contatá-lo. Abaixo, a Figura 11 exemplifica o uso desses elementos.

```

<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
                    entityID="https://inf.ufsc.br/~luz_fc/idp/shibboleth">
  <md:IDPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol
                    urn:oasis:names:tc:SAML:1.1:protocol urn:mace:shibboleth:1.0">
    </md:IDPSSODescriptor>
  <md:Organization>
    <md:OrganizationName xml:lang="en">LRG</md:OrganizationName>
    <md:OrganizationDisplayName xml:lang="en">Laboratório de Redes e Gerencia</md:OrganizationDisplayName>
    <md:OrganizationURL xml:lang="en">http://lrg.ufsc.br/</md:OrganizationURL>
  </md:Organization>
  <md:ContactPerson contactType="support">
    <md:GivenName>Luiz Felipe</md:GivenName>
    <md:EmailAddress>luizfc@inf.ufsc.br</md:EmailAddress>
  </md:ContactPerson>
  <md:ContactPerson contactType="administrative">
    <md:GivenName>Luiz</md:GivenName>
    <md:SurName>Feliipe</md:SurName>
    <md:EmailAddress>luizfc@inf.ufsc.br</md:EmailAddress>
  </md:ContactPerson>
</md:EntityDescriptor>

```

Figura 11: Configuração dos elementos de organização e contato.

De um modo geral, os provedores de identidade e serviço do Shibboleth possuem mecanismos semelhantes para a aquisição de metadados, embora haja variações em alguns detalhes. Na sequência, serão descritas as particularidades do processo de consumo e publicação de metadados por parte dos provedores de serviço e identidade.

3.2.4.1 PROVEDOR DE METADADOS – IDP

Provedores de metadados (*Metadata Providers*) são definidos pelo elemento `<MetadataProvider>` e identificados pelo atributo `xsi:type`. Além disso, com o objetivo de assegurar a sua singularidade e possuir uma referência para o provedor, é necessário definir o seu respectivo atributo `id`.

Os provedores de metadados são divididos em quatro tipos, separados de acordo com a maneira que fornecem os metadados. Abaixo, segue a descrição de cada um:

- **Provedor de Metadados por Encadeamento (*Chaining Metadata Provider*):** quando um provedor de identidade (IdP) solicita metadados a partir deste tipo de fornecedor, ele retorna a resposta do primeiro fornecedor a enviar uma resposta na cadeia. Por exemplo: um IdP está realizando uma busca e há os provedores X, Y e Z

definidos em cadeia. Y e Z contém o que o IdP está buscando, mas somente o primeiro irá enviar uma resposta que não seja vazia à solicitação. Não será realizada nenhuma tentativa de mesclar metadados de origens diferentes. Esse tipo é definido da seguinte maneira:

```

1 <!--Elementos da parte confiável (RelyingParty) acima-->
2 <MetadataProvider xsi:type="ChainingMetadataProvider"
3     xmlns="urn:mace:shibboleth:2.0:metadata"
4     id="MeusMetadados">
5 <!-- O atributo id é obrigatório -->
6 </MetadataProvider>

```

Figura 12: Provedor de Metadados por Encadeamento.

- **Provedor de Metadados via Arquivo em Máquina (*Filesystem Metadata Provider*):** esse tipo realiza a leitura dos metadados dispostos em um arquivo armazenado na máquina local. Para fins de melhora de performance, os metadados são mantidos na memória cache e, caso uma atualização seja detectada, o arquivo é alterado para a versão mais recente. Na sequência, como deve ser definido.

```

1 <!--Elementos da parte confiável (RelyingParty) acima-->
2 <MetadataProvider xsi:type="FilesystemMetadataProvider"
3     xmlns="urn:mace:shibboleth:2.0:metadata"
4     id="MeusMetadados" MetadataFile="C:/Users/luiz/metadata.xml" />
5 </MetadataProvider>

```

Figura 13: Provedor de Metadados via Arquivo em Máquina.

- **Provedor de Metadados via HTTP (*HTTP Metadata Provider*):** o provedor realiza a leitura dos metadados a partir de uma URL. Assim como *Filesystem Metadata Provider*, os metadados são mantidos na memória cache com o objetivo de melhora de performance. Mesmo estando disponível para uso, é recomendada a utilização do tipo *File Backed HTTP Metadata Provider* ao invés desse, por questões de segurança. Abaixo, sua definição:

```

1 <!--Elementos da parte confiável (RelyingParty) acima-->
2 <MetadataProvider xsi:type="HTTPMetadataProvider"
3     xmlns="urn:mace:shibboleth:2.0:metadata"
4     id="MeusMetadados"
5     MetadataURL="http://chiaradia.comyr.com/tcc/metadataUpload2.xml" />
6 </MetadataProvider>

```

Figura 14: Provedor de Metadados via HTTP.

- **Provedor de Metadados via HTTP com *backup* local (*File System HTTP Metadata Provider*):** opera de maneira semelhante ao mencionado anteriormente, entretanto, armazena os metadados em um arquivo temporário local, para que seja possível a recuperação dos dados de forma bem sucedida. Essa medida evita situações em que os metadados são buscados com sucesso, mas, devido a um grande número de requisições, o servidor se encontra desativado.

```

1 <!--Elementos da parte confiável (RelyingParty) acima-->
2 <MetadataProvider xsi:type="HTTPMetadataProvider"
3     xmlns="urn:mace:shibboleth:2.0:metadata" id="MeusMetadados"
4     MetadataURL="http://chiaradia.comyr.com/tcc/metadataUpload2.xml"
5     backingFile="C:/Users/luiz/metadata.xml" />
6 </MetadataProvider>

```

Figura 15: Provedor de Metadados via HTTP com backup local.

3.2.4.2 PROVEDOR DE METADADOS – SP

Provedores de metadados (*Metadata Providers*) são definidos pelo elemento `<MetadataProvider>` e configuram a fonte de metadados que será utilizada pelo provedor de serviço (SP). Diferentemente de outros arquivos de configurações, que descrevem como o SP deve se comportar, os metadados carregados descrevem com quais provedores de identidade ele deve interagir.

Cada aplicação determina o conjunto de partes confiáveis a partir da utilização dos seus metadados. Assim como nos provedores de identidade, existem quatro tipos possíveis para os provedores de serviço, conforme descrito abaixo:

- **Provedor de Metadados XML (*XML Metadata Provider*):** fornece metadados a partir de arquivos XML no padrão SAML 2.0. Esses arquivos podem estar armazenados localmente ou remotamente. Possui uma grande quantidade de atributos, dos quais são importantes ressaltar: `minRefreshDelay` (determina um tempo mínimo em segundos que deve ser verificado se houve uma mudança nos recursos) e `maxRefreshDelay` (possui função semelhante ao mencionado anteriormente, mas determina um tempo máximo). Abaixo, como deve ser definido:

```

1 <MetadataProvider type="XML"
2   url="http://chiaradia.comyr.com/tcc/metadataUpload2.xml"
3   backingFilePath="metadados.xml" maxxRefreshDelay="3600">
4   <!-- Verifica a assinatura no arquivo de metadados-->
5     <MetadataFilter type="Signature" certificate="inc-md-cert.pem"/>
6   <!-- Todos os metadados do IdP são consumidos no processo -->
7     <MetadataFilter type="EntityRoleWhiteList">
8       <RetainedRole>md:IDPSSODescriptor</RetainedRole>
9       <RetainedRole>md:AttributeAuthorityDescriptor</RetainedRole>
10    </MetadataFilter>
11 </MetadataProvider>

```

Figura 16: Provedor de Metadados XML.

- **Provedor de Metadados Dinâmico (*Dynamic Metadata Provider*):** quando o elemento `entityID` é definido por uma URL, este tipo tentará converter o conteúdo da página em uma instância XML. O resultado desse processo será armazenado durante o tempo definido no atributo `cacheDuration` ou `validUntil` – ou até o processo ser reiniciado. Para ser utilizado, o atributo `type` deve receber o valor “Dynamic”.
- **Provedor de Metadados por Encadeamento (*Chaining Metadata Provider*):** permite que múltiplas fontes de metadados sejam consumidas em sequência. Para ser utilizado, o atributo `type` deve receber o valor “Chaining”, conforme exemplo de configuração abaixo

```

1 <MetadataProvider type="Chaining">
2   <MetadataProvider type="XML" path="parceiros.xml"/>
3   <MetadataProvider type="XML"
4     url=" http://chiaradia.comyr.com/tcc/metadataUpload2.xml"
5     backingFilePath="metadados.xml"/>
6 </MetadataProvider>

```

Figura 17: Provedor de Metadados por Encadeamento.

- **Provedor de Metadados por Diretório (*Folder Metadata Provider*):** carrega um único diretório que irá servir como fonte de metadados. É interessante ressaltar que esse diretório será monitorado para possíveis atualizações dos arquivos existentes, ou seja, novos arquivos não serão considerados caso sejam incluídos. Para ser utilizado, o atributo `type` deve receber o valor “Folder”.

Além dos elementos que têm como função o fornecimento de metadados, existem outros cuja função é acrescentar, modificar ou excluir as informações de acordo com as suas

configurações. Tratam-se dos filtros de metadados (*Metadata Filter*), definidos pelo elemento `<MetadataFilter>`, que configura um filtro e examina os metadados obtidos.

Filtros normalmente são utilizados para a criação de requisitos adicionais de segurança ou até mesmo trabalhar em conjunto com outros recursos de software. Abaixo, seguem os filtros disponíveis para uso:

- **Filtro de assinatura (*Signature Metadata Filter*):** valida qualquer assinatura XML existente nos metadados de acordo com as informações configuradas no filtro.

```
2 <MetadataFilter type="Signature"
3   certificate="certificado.pem"/>
```

Figura 18: Filtro de assinatura.

- **Filtro de partes confiáveis (*Whitelist Metadata Filter*):** este filtro deleta os metadados de fontes cujo o entityID não esteja listado nas configurações.

```
1 <MetadataFilter type="Whitelist">
2   <Include>https://idp.confiable.com/shibboleth</Include>
3 </MetadataFilter>
```

Figura 19: Filtro de partes confiáveis.

- **Lista negra (*Blacklist Metadata Filter*):** opera de forma semelhante ao filtro mencionado anteriormente, com a diferença de que irá deletar os metadados somente das fontes listadas no filtro.

```
1 <MetadataFilter type="Blacklist">
2   <Exclude>https://idp.naoconfiavel.com/shibboleth
3 </Exclude>
4   <Exclude>urn:evil:empire:entities</Exclude>
5 </MetadataFilter>
```

Figura 20: Lista negra.

- **Filtro de atributo obrigatório – validUntil (*Require validUntil Metadata Filter*):** este filtro irá rejeitar metadados que não possuem o atributo `validUntil` ou sua validade está excedida.

```

1 <MetadataFilter type="RequireValidUntil"
2   maxValidityInterval="604800"/>

```

Figura 21: Filtro de atributo obrigatório.

- **Filtro de papéis (*EntityRoleWhiteList Metadata Filter*):** remove informações irrelevantes sobre os papéis com o objetivo de economizar memória.

```

1 <MetadataFilter type="EntityRoleWhiteList">
2   <RetainedRole>md:IDPSSODescriptor</RetainedRole>
3 </MetadataFilter>

```

Figura 22: Filtro de papéis.

- **Filtro de atributos (*EntityAttributes Metadata Filter*):** adiciona elementos em entidades com o objetivo de basear seu comportamento neles. Podem ser utilizados para popular atributos de usuário em tempo de compilação e direcionar outros filtros.

```

1 <MetadataFilter type="EntityAttributes">
2   <saml2:Attribute FriendlyName="estado" Name="urn:oid:2.5.4.8"
3     NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
4     <saml2:AttributeValue>Santa Catarina</saml2:AttributeValue>
5   </saml2:Attribute>
6   <saml2:Attribute FriendlyName="localidade" Name="urn:oid:2.5.4.7"
7     NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
8     <saml2:AttributeValue>Florianopolis</saml2:AttributeValue>
9   </saml2:Attribute>
10  <Entity>urn:mace:incommon:osu.edu</Entity>
11 </MetadataFilter>

```

Figura 23: Filtro de atributos.

O exemplo abaixo demonstra a configuração de um provedor de metadados do tipo XML filtrando de acordo com a assinatura e o elemento `validUntil` com a validade de no máximo catorze dias. Além disso, serão conservados somente os elementos `md:IDPSSODescriptor` e `md:AttributeAuthorityDescriptor`.

```
1 <MetadataProvider type="XML"
2     url="http://chiaradia.comyr.com/tcc/metadataUpload2.xml"
3 <backingFilePath="metadados.xml" maxxRefreshDelay="3600">
4     <!-- Requer validUntil de no máximo 14 dias -->
5     <MetadataFilter type="RequireValidUntil"
6         maxValidityInterval="1209600"/>
7     <MetadataFilter type="Signature"
8         certificate="cert.pem"/>
9     <MetadataFilter type="EntityRoleWhiteList">
10         <RetainedRole>md:IDPSSODescriptor</RetainedRole>
11         <RetainedRole>md:AttributeAuthorityDescriptor</RetainedRole>
12     </MetadataFilter>
13 </MetadataProvider>
```

Figura 24: Configuração do provedor de metadados XML.

4. DESENVOLVIMENTO PRÁTICO

Neste capítulo será definida a proposta de desenvolvimento prático baseado em um problema e em sua respectiva solução, proposta pelo autor.

4.1 PROPOSTA

Atualmente o Shibboleth não oferece ferramentas para a importação ou exportação de metadados. Em vez disso, ele publica e consome metadados no formato XML como um mecanismo que lista um conjunto de partes confiáveis e indica ao software como se comunicar com eles de forma segura.

O software do provedor de identidade tem suporte para a comunicação com provedores de serviço desconhecidos em determinadas situações, desde que os mecanismos POST, o qual submete os dados para serem processados por uma fonte específica, e de ligação (*binding*) estejam sendo utilizados. No entanto, não existe nenhum tipo de estrutura que permita que provedores de serviço se comuniquem com provedores de identidade desconhecidos, ocasionando em erros na busca de metadados (*metadata lookup error*).

Diante desse problema, o consórcio Internet2 desenvolveu um framework em Java denominado Metadata Aggregator, cuja função é permitir o desenvolvimento de aplicações que permitam a leitura de metadados de múltiplas fontes que seja possível verificar, filtrar e manipular esses dados [INTERNET2, 2014]. Apesar de finalizada, a biblioteca é fracamente documentada, fato que gera muitas discussões em grupos que tem como objetivo o desenvolvimento de aplicações utilizando esse framework.

Nesse contexto, a proposta deste trabalho é a realização de um estudo do funcionamento do Metadata Aggregator e, posteriormente, realizar a sua documentação enumerando suas principais funções e como elas operam. Baseando-se nessa documentação, será proposta uma aplicação desenvolvida na linguagem de programação Java, a qual irá realizar o download de arquivos de metadados de diferentes provedores, agrega-los e, via FTP (*File Transfer Protocol*), disponibilizar o arquivo com os metadados agregados para que seja consumido por um outro provedor a fim de efetuar o acordo de chaves entre os provedores. A Figura 25, demonstra o modelo de aplicação descrito:

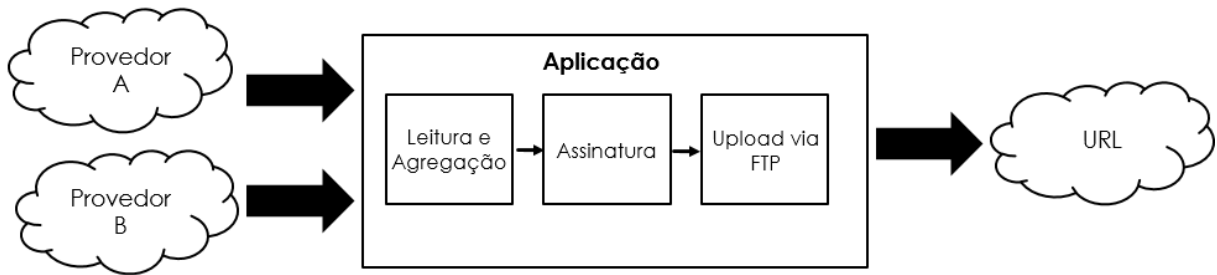


Figura 25: Modelo da aplicação proposta

4.2 SPRING FRAMEWORK

A plataforma que será utilizada para o desenvolvimento da aplicação proposta neste trabalho foi desenvolvida com bases em um dos frameworks mais utilizado nos dias de hoje: o Spring Framework.

O framework fornece uma infraestrutura de suporte ao desenvolvimento de aplicações em Java, sendo baseado nos padrões de projeto injeção de dependência (*Dependency Injection*) e inversão de controle (*Inversion of Control* ou IoC). Possui uma arquitetura baseada em POJOs (*Plain Old Java Objects*) e interfaces, facilitando a realização de testes unitários e surge como uma alternativa à complexidade existente no uso de EJBs. [SPRING, 2014]

Segundo [FOWLER, 2004], a inversão de controle consiste em uma mudança do fluxo normal de operações. Para utilizar um framework, o código da aplicação deve ser criado e mantido dentro das estruturas do framework, sendo acessível por meio de classes que estendem classes do próprio framework. O framework realiza a chamada do código da aplicação e, após a sua utilização, o controle do fluxo retorna para ele, ao invés de para o usuário.

A injeção de dependência se trata de um padrão de desenvolvimento que objetiva manter o baixo acoplamento entre os módulos de um sistema. Nesse sistema, as dependências serão definidas pela configuração de um *container* responsável por injetar em cada componente as suas dependências declaradas. [FOWLER, 2004]

Os recursos do Spring Framework estão divididos entre vinte módulos, organizados em seis diferentes grupos, conforme a Figura 26:

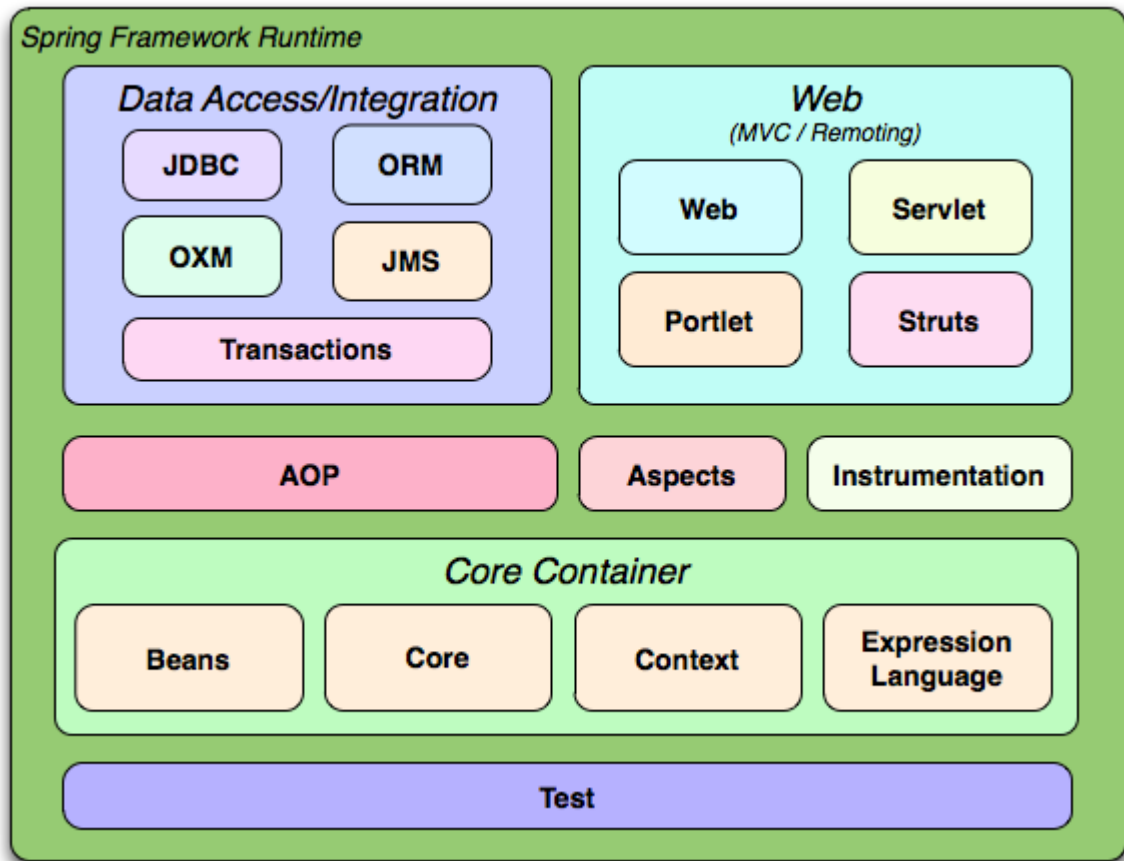


Figura 26: Visão geral da organização dos módulos do Spring Framework [SPRING, 2014]

4.3 METADATA AGGREGATOR

O Shibboleth Metadata Aggregator (MDA) é um framework desenvolvido em Java sobre a plataforma Spring Framework. A partir de uma sequência de estágios previamente configurados, realiza o processamento de uma coleção de itens. Esses estágios são implementados como *JavaBeans*, que é uma especificação que define uma API e dita regras de configuração e comunicação entre componentes e convenções de programação, ou seja, são instâncias de classes Java que possuem algumas propriedades que são configuradas anteriormente.

Cada uma dessas etapas configuradas realiza uma determinada tarefa, que irá depender diretamente de uma tarefa realizada por outra etapa para constituir uma funcionalidade propriamente dita. No contexto do MDA, o conjunto de estágios operando em conjunto para a criação de uma funcionalidade recebe o nome de “Pipeline”, mesmo termo utilizado em ambientes UNIX.

O Metadata Aggregator pode ser considerado uma ferramenta bastante genérica, o que possibilita a descoberta de novos usos para ela a medida em que é usada. De uma forma geral, ela atende qualquer caso em que um usuário que deseja ler, transformar, escrever ou procurar dados. De acordo com seus desenvolvedores [SHIBOLLETH, 2014], os seguintes casos de uso foram responsáveis pelo seu desenvolvimento:

- Uma ferramenta *Command Line* que possibilita a leitura de uma grande quantidade de metadados no padrão SAML, valide sua estrutura, filtra as entidades baseado em regras previamente definidas e assine esses arquivos digitalmente.
- Uma ferramenta *Command Line* associada a um web service que possibilita a comunicação interfederações.
- Um *web service* que opere em conjunto com os provedores de serviço e identidade, recebendo as requisições a fim de efetuar as operações mais complexas, permitindo que aqueles possam ser configurados de maneira mais simples, segura e eficaz.

A estrutura do Metadata Aggregator é composta por quatro itens, conforme descrito abaixo:

- **Itens:** em seu contexto, cada dado é denominado um item, sendo assim, pode representar qualquer coisa (uma pessoa, um grupo ou uma entidade SAML) em qualquer formato (XML, JSON, ANS1). Cada item possui suas respectivas informações que podem ser computadorizadas (metadados), que serão processadas ao longo da execução dos estágios do MDA.
- **Pipeline:** localizado no núcleo do Metadata Aggregator, há um componente através do qual toda a coleção de metadados é processado e/ou modificado e é formado por um conjunto de estágios configurados previamente.

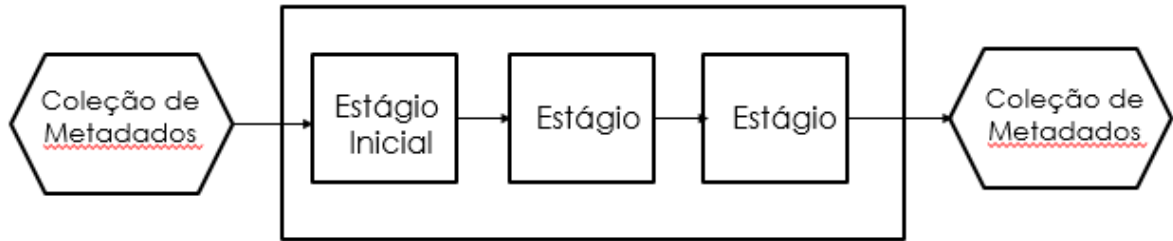


Figura 27: Estágios em um pipeline. Adaptado de [SHIBBOLETH, 2014]

A Figura 27 representa o fluxo de processamento de metadados, onde cada estágio realiza uma determinada tarefa. O primeiro estágio, denominado *Source Stage* (Estágio Inicial), se refere ao processo de obtenção e leitura dos dados. Por exemplo: um arquivo XML armazenado em um diretório do servidor será processado por diversos estágios, que irão verificar sua estrutura, elementos válidos e manipulá-los.

- **Command Line:** utilizando esta ferramenta, é possível executar os estágios do *pipeline* e escrever seus resultados em um arquivo. Pode ser invocada via script e requer dois argumentos, seguindo a ordem: diretório do arquivo de configurações e o arquivo contendo a sequência de estágios a serem realizados. Além disso, é possível a utilização da classe `ApplicationContext` para a leitura do arquivo XML.
- **Web Service:** por meio da criação de uma aplicação *RESTful*, é possível utilizar métodos HTTP, como GET e POST por exemplo, para que o consumidor obtenha acesso a um ou uma coleção de itens baseado em um identificador previamente informado. A Figura 28 demonstra o funcionamento desta parte da arquitetura do Metadata Aggregator:

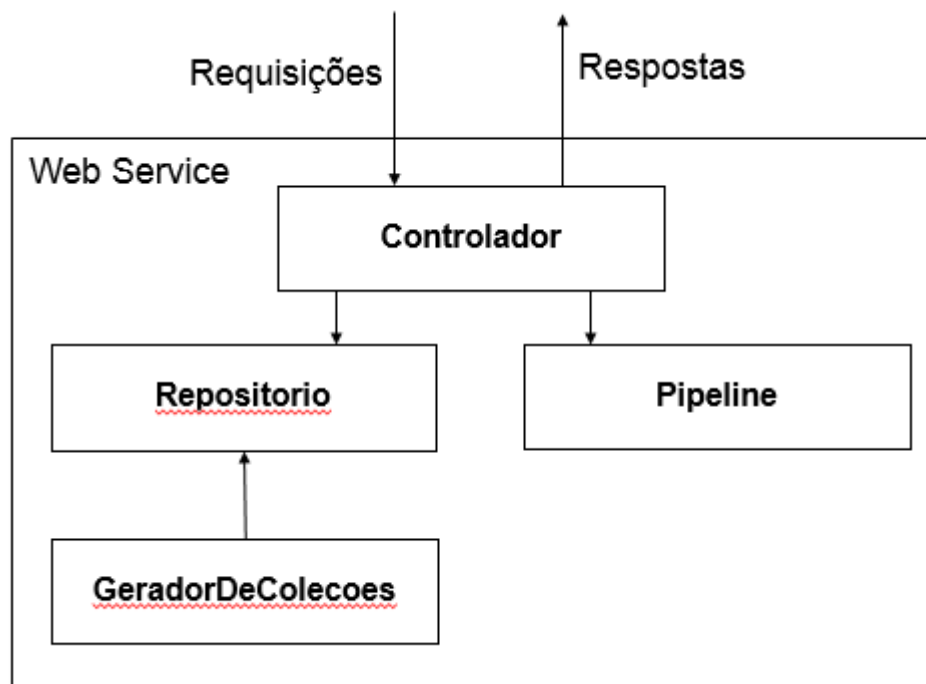


Figura 28: Web Service. Adaptado de [SHIBBOLETH, 2014].

Aplicações *RESTful* são construídas sobre a arquitetura REST, que pode ser definida como um conjunto de princípios que definem como padrões Web devem ser utilizados de forma a aproveitar ao máximo seus recursos. Conforme mencionado anteriormente, a necessidade da utilização de um identificador parte do pressuposto que a aplicação utilizará recursos com múltiplas representações, além da vinculação de todos esses recursos por meio de *links*. O web service descrito na Figura 28 é composto por:

- Um Controlador responsável por receber requisições e enviar respostas a partir de consultas em um repositório com metadados;
- Esse repositório irá armazenar e indexar as coleções de itens geradas pelo GeradorDeColecoes;
- O GeradorDeColecoes será responsável por ler e organizar os metadados de forma que seja possível busca-los;
- Um Pipeline responsável por realizar operações com metadados e encaminhar a resposta a quem realizou a requisição para o Web Service. Essas operações

podem ir desde a modificação do conteúdo do arquivo até a verificação do certificado assinante, por exemplo.

A Figura 29 mostra o diagrama de classes construído a partir da leitura do código fonte do Metadata Aggregator, disponível no repositório do Shibboleth [SVN-SHIBBOLETH, 2014]:

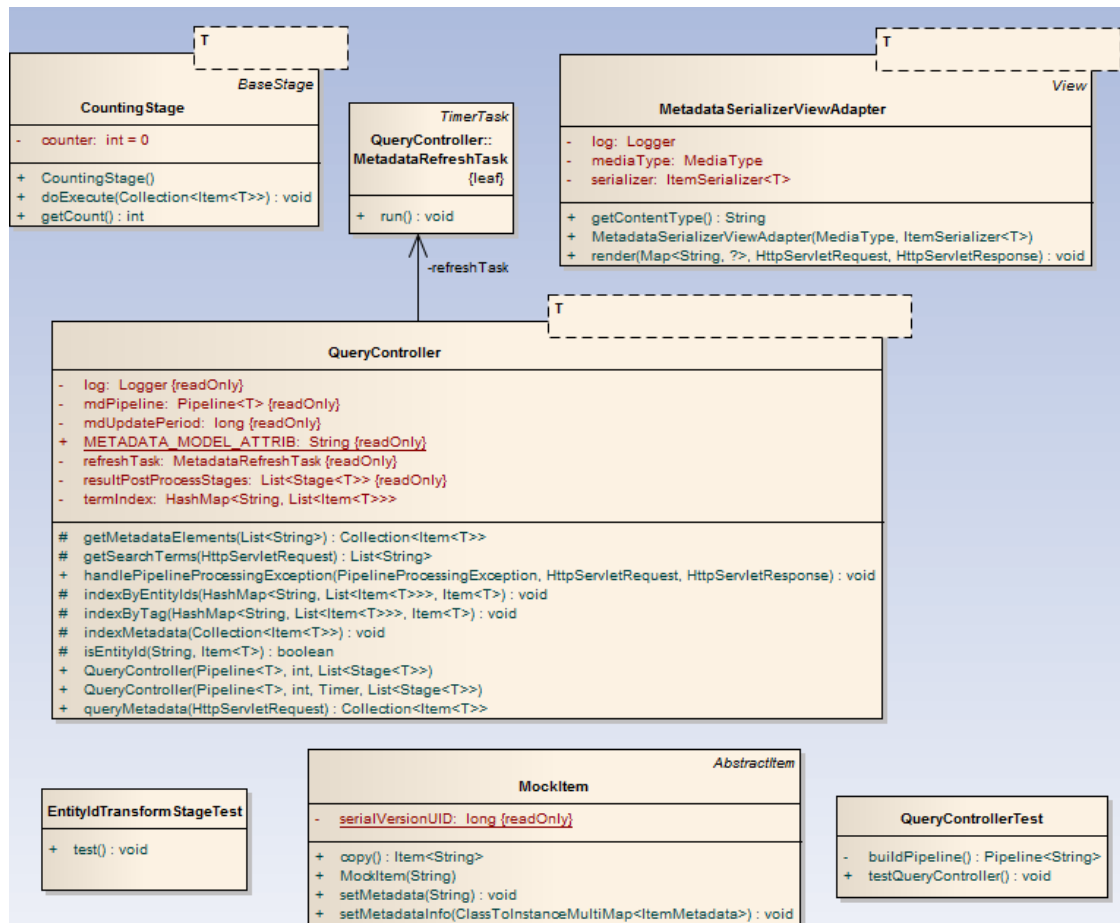


Figura 29: Diagrama de Classes das funções disponíveis para serem utilizadas em um Web Service

4.3.1 CONFIGURAÇÕES

No contexto do Metadata Aggregator, um pipeline é definido como uma coleção de estágios que irá realizar uma determinada função. Cada estágio é formado por um conjunto de classes, as quais possuem seus respectivos métodos para realizar suas tarefas. A seguir, a

relação de estágios disponíveis para a configuração de um *Pipeline* que podem ser mencionadas são:

- **Source Stage:** normalmente considerado o estágio inicial, é composto por classes que permitem que um determinado arquivo seja lido e preparado para sua posterior manipulação.
- **DOM Processing Stage:** por meio da utilização da tecnologia *Document Object Model*, DOM, para interação com objetos HTML e XML, este estágio contém classes que manipulam os arquivos de metadados em formato XML.
- **SAML Processing Stage:** opera de maneira semelhante ao estágio anterior, com a diferença de que os arquivos manipulados devem estar organizados no padrão SAML.
- **MDRPI Processing Stage:** implementa funcionalidades relacionadas à especificação SAML V2.0 *Metadata Extensions for Registration and Publication Information*, utilizada para criar uma estrutura de arquivos de Metadados a fim de que todos os atores possam interagir.
- **Item Metadata Stage:** estágio responsável pelo processo de marcação (*tag*) dos itens de uma determinada coleção.
- **Pipeline Structure Stage:** estágio cuja função está relacionada a criação de estruturas avançadas de *Pipelines*, como a combinação de resultados de diferentes *pipelines*, por exemplo.
- **Outros:** estágios responsáveis por funções alternativas, como serialização de itens e execução de scripts de compilação, por exemplo.

Para configurar um *Pipeline* é necessário definir seu identificador (id) e seus estágios (*stages*). Com relação a código, um *Pipeline* é definido como qualquer classe que implemente a interface `net.shibboleth.metadata.pipeline.Pipeline`. A Figura 31 representa um exemplo de configuração, na qual é o arquivo de metadados é lido e manipulado, para que somente elementos do tipo `<md:ContactPerson>`, com o atributo `contactType` definido com o valor “administrativo”, não sejam removidos.

```

<md:ContactPerson contactType="support">
  <md:GivenName>Luiz Felipe</md:GivenName>
  <md:EmailAddress>luizfo@inf.ufsc.br</md:EmailAddress>
</md:ContactPerson>
<md:ContactPerson contactType="administrative">
  <md:GivenName>Luiz</md:GivenName>
  <md:SurName>Feliipe</md:SurName>
  <md:EmailAddress>luizfo@inf.ufsc.br</md:EmailAddress>
</md:ContactPerson>

```

Figura 30: Elemento do arquivo de metadados que não será removido

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans default-init-method="initialize" xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.springframework.org/schema/beans
5     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
6     <!-- Esse primeiro estágio realiza a leitura de um ou vários arquivos de Metadados em um diretório -->
7     <bean id="source" class="net.shibboleth.metadata.dom.DOMFilesystemSourceStage">
8         <property name="id" value="source"/>
9         <property name="parserPool">
10            <bean class="net.shibboleth.utilities.java.support.xml.BasicParserPool"
11                init-method="initialize"/>
12        </property>
13        <property name="source">
14            <bean class="java.io.File">
15                <constructor-arg value="C:/Users/LuizFelipe/workspace/TCCMetadataAggregator/">
16            </bean>
17        </property>
18    </bean>
19    <!-- Nesse estágio é aplicado um filtro, para não remover apenas contatos do tipo "administrativo" -->
20    <bean id="removeInvalidContactPerson" class="net.shibboleth.metadata.dom.saml.ContactPersonFilterStage">
21        <property name="id" value="removeInvalidContactPerson"/>
22        <property name="tiposDesejados">
23            <list>
24                <value>administrativo</value>
25            </list>
26        </property>
27    </bean>
28    <!-- Nessa etapa, é realizada a serialização dos itens da coleção e determinada a saída -->
29    <bean id="serialize" class="net.shibboleth.metadata.pipeline.SerializationStage">
30        <property name="id" value="serializeIdPs"/>
31        <property name="outputFile">
32            <bean class="java.io.File">
33                <constructor-arg value="C:/Users/LuizFelipe/workspace/TCCMetadataAggregator/">
34            </bean>
35        </property>
36        <property name="serializer">
37            <bean id="domSerializer" class="net.shibboleth.metadata.dom.DOMELEMENTSerializer" />
38        </property>
39    </bean>
40    <!-- Nessa etapa, é definido o pipeline, que representa um conjunto de
41         tarefas para criar uma funcionalidade -->
42    <bean id="main" class="net.shibboleth.metadata.pipeline.SimplePipeline" init-method="initialize">
43        <property name="id" value="pipeline"/>
44        <property name="stages">
45            <list>
46                <ref bean="source"/> <ref bean="removeInvalidContactPerson"/>
47                <ref bean="serialize" />
48            </list> </property> </bean>< /beans>

```

Figura 31: Exemplo de configuração de um Pipeline (arquivo XML)

Na Figura 32, segue parte do código escrito na linguagem de programação Java para realizar a chamada da função, além da declaração das constantes:

```
private static final String SUPORTE = null;
private static final String ADMINISTRATIVO = null;
private static final String SERVIDOR = null;
private static final String DESENVOLVIMENTO = null;

/** Tipos permitidos de contatos. */
private ImmutableSet<String> tiposPermitidos = ImmutableSet.copyOf(new String[]
{SUPORTE, ADMINISTRATIVO, SERVIDOR, DESENVOLVIMENTO});
private ImmutableSet<String> tiposDesejados = ImmutableSet.copyOf(tiposPermitidos);
```

Figura 32: Parte do código em Java para o arquivo de configurações

4.4 VISÃO GERAL DA APLICAÇÃO

O consórcio Internet2 desenvolveu um framework em Java denominado Metadata Aggregator, cuja função é permitir o desenvolvimento de aplicações que permitam a leitura de metadados de múltiplas fontes e, posteriormente, verifica-los, filtrá-los e manipulá-los [INTERNET2, 2014]. A documentação que acompanha o pacote disponível para download não possui exemplos de como utilizar o framework e sua Wiki explica de forma bastante clara o relacionamento entre as classes, contudo não existem modelos de usos para as mesmas. Além disso, soma-se o fato do código possuir uma lógica bastante singular, sendo constituído de estruturas que são pouco utilizadas no cotidiano, fato que foi possível constatar pela análise do mesmo e nas questões levantadas nos fóruns de discussão.

Para a solução do problema destacado, será proposto o desenvolvimento de uma aplicação *stand alone* na linguagem Java, que irá realizar o download de arquivos de metadados de diferentes federações para, na sequência, percorrer os estágios de “Leitura e Agregação”, “Assinatura” e “Upload via FTP”. Ao final desse processo, um arquivo de metadados com todas as propriedades dos utilizados no início da operação, estará disponível em um endereço da Web.

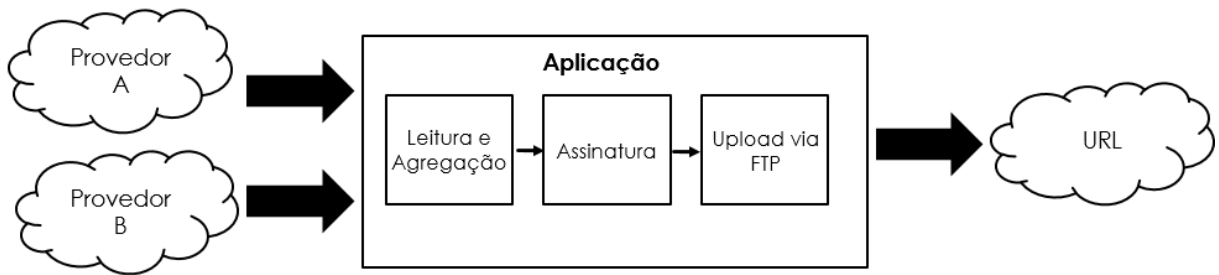


Figura 33: Modelo da aplicação proposta

Aplicações *stand alone* são completamente autossuficientes, ou seja, não necessitam de nenhum software auxiliar, como um interpretador por exemplo, sob o qual terão de ser executados. Objetivando o uso intuitivo dos recursos oferecidos pela aplicação, além de facilitar o entendimento da mesma, optou-se por separar cada estágio em uma determinada classe, a qual recebe o nome de sua tarefa. Ao final o *workspace* ficou organizado conforme demonstra a Figura 34:

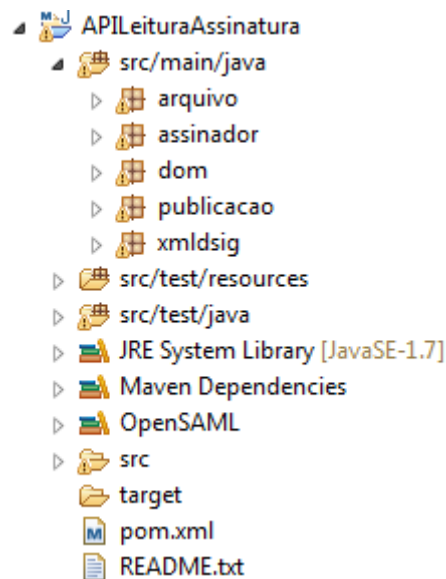


Figura 34: *Workspace* do projeto na IDE Eclipse

É importante ressaltar que para o funcionamento correto da aplicação, o elemento `<MetadataProvider>` dos provedores deve estar configurado com o valor `type="HTTPMetadataProvider"` e o elemento `backingFile` deve receber como valor um diretório com

acesso “liberado” para que a aplicação possa realizar o download dos arquivos de metadados e processá-los. Na sequência, segue a descrição dos estágios.

4.5 ESTÁGIO DE LEITURA E AGREGAÇÃO

Nesta primeira etapa serão definidas duas classes responsáveis pelas funções de leitura e agregação dos metadados obtidos dos provedores. A classe LeArquivoXML será composta pelos pacotes da W3C para criação de árvores DOM a partir de documentos XML e possuirá apenas um método, denominado leArquivoXML (String arquivo), que receberá como parâmetro uma variável do tipo String que representará o diretório onde o arquivo de metadados está armazenado. A Figura 35 representa o código da classe.

```

1 package dom;
2
3 import java.io.File;
12
13 public class LeArquivoXML {
14
15     public void leArquivoXML (String arquivo){
16         try {
17
18             File fXmlFile = new File(arquivo);
19             //DocumentBuilderFactory: define uma API que permite que aplicações obtenham
20             //um parser que crie árvores (DOM) a partir de arquivos XML
21             DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
22             DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
23             Document doc = dBuilder.parse(fXmlFile);
24
25             //Realiza a normalização do documento
26             doc.getDocumentElement().normalize();
27
28             //Retorna uma lista de elementos que estejam inclusos na tag
29             //passada (String) como parâmetro
30             NodeList nList = doc.getElementsByTagName("ds:X509Data");
31
32             System.out.println("-----");
33
34             for (int temp = 0; temp < nList.getLength(); temp++) {
35
36                 Node nNode = nList.item(temp);
37
38                 System.out.println("\nElemento analisado : " + nNode.getNodeName());
39                 if (nNode.getNodeType() == Node.ELEMENT_NODE) {
40
41                     Element eElement = (Element) nNode;
42                     //Imprime todos os dados relacionados a certificados assinantes
43                     System.out.println("Certificado assinante (BASE64) : " +
44                         eElement.getElementsByTagName("ds:X509Certificate").item(0).getTextContent());
45                 }
46             } catch (Exception e) {
47                 e.printStackTrace();
48             }
49         }
50     }
51 }
52 }
53

```

Figura 35: Classe LeArquivoXML.java

A método realiza a conversão do arquivo localizado no diretório recebido como parâmetro para um objeto do tipo File para, na sequência, convertê-lo em um DOM (classe Document) para que seja possível iterar sobre os nodos da árvore de forma a ler e imprimir os dados dos elementos desejados, conforme disposto na Figura 35.

A classe AgregadorXML também será composta por apenas um método, cujo nome é agregarXML (String origem, String destino) e recebe como parâmetro duas Strings, sendo uma com o diretório do arquivo de metadados de um determinado provedor e outra com o diretório do arquivo de metadados que “receberá” os elementos responsáveis pela descrição das entidades do provedor. Em síntese, o arquivo destino receberá os elementos do arquivo origem. A Figura 36 demonstra como esse processo é realizado em Java.

```

1 package dom;
2
3 import java.io.BufferedWriter;
26
29 * @author 'Luiz-Felipe'
34
35 public class AgregadorXML {
36
37     public static void agregarXML(String origem, String destino) throws
38     SAXException, IOException, ParserConfigurationException, TransformerException {
39
40         File documentoOrigem = new File(origem);
41         File documentoDestino = new File (destino);
42
43         DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
44         DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
45         Document docOrigem = dBuilder.parse(documentoOrigem);
46         Document doDestino = dBuilder.parse(documentoDestino);
47
48         Element origemRoot = docOrigem.getDocumentElement();
49         Element destinoRoot = doDestino.getDocumentElement();
50
51         Node filho = null;
52         while ((filho = origemRoot.getFirstChild()) != null) {
53             doDestino.adoptNode(filho);
54             destinoRoot.appendChild(filho);
55         }
56
57         TransformerFactory tFactory = TransformerFactory.newInstance();
58         Transformer transformer = tFactory.newTransformer();
59         transformer.setOutputProperty(OutputKeys.INDENT, "yes");
60
61         DOMSource dom = new DOMSource(doDestino);
62         StreamResult resultado = new StreamResult(new StringWriter());
63         transformer.transform(dom, resultado);
64
65         Writer saida = new BufferedWriter(new FileWriter ("src/test/resources/agregado-Switch-Cafe.xml"));
66         String saidaXML = resultado.getWriter().toString();
67         saida.write(saidaXML);
68         saida.close();
69     }
70
71 }
72 }
73

```

Figura 36: Classe AgregadorXML.java

Primeiramente os arquivos são convertidos em objetos do tipo File para posteriormente serem convertidos em objetos da classe Document (DOM). Essa ação é necessária para que fosse possível percorrer seus elementos de forma a obter os elementos responsáveis pela descrição das entidades do provedor.

Para o processo de agregação, foi criado um laço iterativo que irá percorrer os elementos (classe Element) do objeto Documento e que irá verificar se o elemento seguinte ao raiz era diferente de nulo. Caso não fosse, o documento agregado “adotaria” esse elemento, incorporando-o em sua estrutura. Após a iteração, um novo arquivo XML será criado localmente, para que as próximas etapas sejam realizadas. A Figura 37 destaca o processo descrito nesse parágrafo.

```
51     Node filho = null;
52     while ((filho = origemRoot.getFirstChild()) != null) {
53         doDestino.adoptNode(filho);
54         destinoRoot.appendChild(filho);
55     }
```

Figura 37: Laço responsável pela agregação dos elementos do arquivo.

É importante ressaltar que a classe responsável pela agregação dos elementos pode ser considerada a mais importante do projeto, visto que, caso sua funcionalidade seja executada corretamente, os elementos responsáveis pela descrição das entidades pertencentes a um determinado provedor, estarão dispostos também no novo arquivo, junto de outras entidades, possibilitando assim que o acordo de chaves seja realizado e o acesso ao dados seja efetuado com sucesso.

4.6 ASSINATURA XML

Assinatura digital é uma assinatura eletrônica que comprova que um indivíduo criou ou concorda com um documento assinando digitalmente, como a assinatura do próprio punho comprova a autoria de um documento escrito.

A assinatura será realizada no padrão XMLDSig, o qual é específico para arquivos do tipo XML, podendo ser aplicada em três tipos básicos: *detached* (o arquivo e a assinatura estão em arquivo separados), *enveloped* (o documento XML e a assinatura se encontram no

mesmo arquivo, de forma sequencial) e *enveloping* (o documento e a assinatura estão contidos em um “envelope” XML). Neste trabalho, a assinatura gerada será uma *enveloped*, utilizando um repositório do tipo PKCS#12, o qual permite o armazenamento de chaves privadas juntamente com o certificado de chave pública.

Para que fosse possível a execução do processo de assinatura digital, primeiramente foi necessário gerar um repositório JKS, importando o certificado digital e, posteriormente, acessar esse repositório para obter o aliás do certificado. As Figuras 38 e 39 demonstram as ações enumeradas:

```
C:\Users\LuizFelipe\Documents>keytool -importkeystore -srckeystore certificado.p
fx -srcstoretype pkcs12 -destkeystore certrepo.jks -deststoretype JKS
Informe a senha da área de armazenamento de chaves de destino:
Informe novamente a nova senha:
Informe a senha da área de armazenamento de chaves de origem:
Entrada do alias b95e6d13-c3d1-11e3-aa18-005056c00008 importada com êxito.
Comando de importação concluído: 1 entradas importadas com êxito, 0 entradas fa
lharam ou foram canceladas
C:\Users\LuizFelipe\Documents>_
```

Figura 38: Geração do repositório JKS.

```
C:\Users\LuizFelipe\Documents>keytool -list -keystore "C:\Users\LuizFelipe\Docum
ents\certrepo.jks" -storepass *****
Tipo de área de armazenamento de chaves: JKS
Fornecedor da área de armazenamento de chaves: SUN
Sua área de armazenamento de chaves contém 1 entrada
b95e6d13-c3d1-11e3-aa18-005056c00008, 23/06/2014, PrivateKeyEntry,
Fingerprint (SHA1) do certificado: 9F:D8:5F:13:38:DC:58:1A:DD:9B:37:3E:59:08:52:
B4:67:6A:C4:86
```

Figura 39: Obtenção do valor do aliás do certificado digital.

A classe *AssinadorXMLDsig* utiliza os pacotes de criptografia do java (*Javax XML Crypto*) e de conversão de arquivos XML para árvores DOM (W3C). Possui quatro métodos, cujas funções são:

- **getChavePrivada():** método responsável por obter a chave privada do repositório PKCS#12 para que a assinatura digital seja efetuada. A partir do arquivo JKS, busca pelo aliás do certificado e, fornecendo a senha, obtém a chave privada. A Figura 40 representa o método descrito.

```

68 //Método criado para obter a chave privada do certificado, baseado em seu alias
69 public static PrivateKey getChavePrivada() throws Exception {
70     InputStream entrada = new FileInputStream(arquivoJKS);
71     ks.load(entrada, senhaDoRepositorio);
72     entrada.close();
73     Key chavePrivada = (Key) ks.getKey(alias, senhaDoRepositorio);
74     if (chavePrivada instanceof PrivateKey) {
75         System.out.println("Chave Privada encontrada!");
76         return (PrivateKey) chavePrivada;
77     }
78     return null;
79 }

```

Figura 40: Método para obter a chave privada.

- **getValidade(X509Certificate cert):** um certificado digital pode estar válido, expirado ou inválido. Esse método, a partir do certificado recebido no parâmetro, verifica qual o estado do certificado.

```

116 //Método utilizado para verificar a validade do certificado, recebendo um objeto do tipo
117 //X509Certificate como parâmetro
118 //Um certificado digital pode estar válido, expirado ou inválido
119 public static String getValidade(X509Certificate cert) {
120     try {
121         cert.checkValidity();
122         return "Certificado válido!";
123     } catch (CertificateExpiredException e) {
124         return "Certificado expirado!";
125     } catch (CertificateNotYetValidException e) {
126         return "Certificado inválido!";
127     }
128 }

```

Figura 41: Verificação da validade do certificado.

- **getCertificado():** método responsável pela definir o certificado a ser utilizado para a assinatura a partir do seu alias, selecionando-o no seu repositório JKS. A Figura 42 mostra o método destacado.

```

130 //Método utilizado para selecionar o certificado a ser utilizado para assinatura a
131 //partir do seu alias, selecionando-o dentro do JKS
132 public static void getCertificado() throws Exception {
133     InputStream dado = new FileInputStream(arquivoJKS);
134     //Este método (getInstance()) percorre uma lista de provedores de
135     //segurança registrados, começando com os "preferidos"
136     //Como retorno, obtem-se um novo objeto KeyStore encapsulando a implementação
137     //KeyStoreSpi do primeiro provedor que suporta a especificação
138     ks = KeyStore.getInstance("JKS");
139     //Carrega o KeyStore
140     ks.load(dado, senhaDoRepositorio);
141     //Obtem o certificado utilizado o método getCertificate passando o alias como
142     //parâmetro e realizando a conversão de tipo
143     cert = (X509Certificate) ks.getCertificate(alias);
144     //Verifica a validade do certificado e imprime
145     String retorno = AssinadorXMLDSig.getValidade(cert);
146     System.out.println(retorno);
147 }

```

Figura 42: Seleção do certificado.

- assinarDocumento (String localDocumento):** método responsável por assinar o documento, recebe como parâmetro uma String que irá representar o diretório do arquivo XML que o usuário deseja assinar. Novamente o arquivo XML será convertido em um objeto do tipo Document para que seja possível percorrê-lo de forma a criar os elementos que representam a assinatura digital. A partir dos métodos citados anteriormente, são obtidas as propriedades necessárias para efetuar a assinatura digital (certificado digital e posteriormente chave privada) e o documento é efetivamente assinado criando um novo elemento em seu corpo, conforme mostra a Figura 43:

```

154066 <Signature xmlns="http://www.w3.org/2000/09/xmldsig#"><SignedInfo>
154067 <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
154068 <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
154069 <Reference URI="#AAITest-20140627185922"><Transforms><Transform
154070 Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
154071 <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/></Transforms>
154072 <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
154073 <DigestValue>KxzdiIdlQZVpltxL99xxBv/mmVw=</DigestValue></Reference></SignedInfo>
154074 <SignatureValue>U9dGwIDm52eSG+qNpi72PceXSHEtIy9M/qv8fX7VvWwKOKCH77Ac40V/gYJh3Fi5N+BamkFDJWQUC
154075 6TclMprONA57ASEDrXz97/DDTRyK8bBW6ULMpk77DWqowO+qzEX6MGCC3DRQ1KemzhYVB3dhco9s
154076 WYjrij87AF60nei4pI3Nze6h0a5e1Ycx1Txcj+D9H4/1etdx3IHXqGhXojMXOXvvdcojqNVcnsMMe
154077 IwkME68EF6L0393Pwoe1P3mxYneBC6rhv7Ij874Pw59iy6KtTMBTfZh4SMfUrRcljHx9X09AYOwo
154078 wXWqEq8I1v9X78QmJt/rBwlgsVsFTeUeKEiH4w==</SignatureValue><KeyInfo><X509Data><X509Certificate>

```

Figura 43: Arquivo XML assinado.

A Figura 44 representa o método utilizado para efetuar as operações destacadas acima:

```

150 //Método que realiza a assinatura do documento
151 public static void assinarDocumento(String localDocumento) throws Exception {
152     //Criação de uma nova factory
153     DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
154     //Passando o valor true como parâmetro, parser irá prover suporte para XML
155     dbf.setNamespaceAware(true);
156     //Passa o diretório XML como parâmetro e obtém um DOM
157     Document doc = dbf.newDocumentBuilder().parse(new FileInputStream(localDocumento));
158     System.out.println("Documento convertido em DOM.");
159     //Retorna uma XMLSignatureFactory que suporta o mecanismo de processamento XML requisitado ("DOM")
160     XMLSignatureFactory sig = XMLSignatureFactory.getInstance("DOM", new org.jcp.xml.dsig.internal.dom.XMLDSigRI());
161     ArrayList transformados = new ArrayList();
162     //Criam-se as transformações de acordo com os parâmetros
163     //Assinatura XML do tipo ENVELOPED
164     Transform enveloped = sig.newTransform(Transform.ENVELOPED, (TransformParameterSpec) null);
165     Transform c14n = sig.newTransform(C14N_TRANSFORM_METHOD, (TransformParameterSpec) null);
166     //Inclui as transformações no ArrayList criado
167     transformados.add(enveloped);
168     transformados.add(c14n);
169     //Retorna uma NodeList de todos os elementos do documento na tag encaminhada como parâmetro
170     NodeList elements = doc.getElementsByTagName("EntitiesDescriptor");
171     org.w3c.dom.Element el = (org.w3c.dom.Element) elements.item(0); //Define o item 0
172     //Define o atributo "ID" do atributo root (EntitiesDescriptor) como atributo Id do arquivo XML em questão
173     //Maiores detalhes: http://stackoverflow.com/questions/17331187/xml-dig-sig-error-after-upgrade-to-java7u25
174     el.setAttribute("ID", true);
175     String id = el.getAttribute("ID");
176     //String id = el.getElementsByTagName("EntitiesDescriptor");
177     //Cria uma nova referência
178     Reference r = sig.newReference("#".concat(id), sig.newDigestMethod(DigestMethod.SHA1, null),
179     transformados, null, null);
180     //Cria uma nova SignedInfo com os métodos especificados para canonicalização e assinatura, além das listas de referências
181     si = sig.newSignedInfo(sig.newCanonicalizationMethod(CanonicalizationMethod.INCLUSIVE, (C14NMethodParameterSpec) null),
182     sig.newSignatureMethod(SignatureMethod.RSA_SHA1, null), Collections.singletonList(r));
183     //Retorna uma KeyInfoFactory
184     KeyInfoFactory keyInfoFactory = sig.getKeyInfoFactory();
185     List certificadosX509 = new ArrayList();
186     //Adiciona o certificado a lista X509
187     certificadosX509.add(cert);

188     X509Data xd = keyInfoFactory.newX509Data(certificadosX509);
189     ki = keyInfoFactory.newKeyInfo(Collections.singletonList(xd));
190     //Cria-se um contexto para assinatura DOM passando como parâmetro a chave privada do certificado e o documento
191     DOMSignContext contextoDeAssinaturaDOM = new DOMSignContext(getChavePrivada(), doc.getDocumentElement());
192     XMLSignature assinaturaXML = sig.newXMLSignature(si, ki);
193     //Assina o contextoDeAssinaturaDOM - Assinatura ENVELOPED no padrão XMLDSig
194     assinaturaXML.sign(contextoDeAssinaturaDOM);
195     //Define e saída
196     OutputStream saida = new FileOutputStream("src/test/resources/saidaXMLAssinado.xml");
197     TransformerFactory transformerFactory = TransformerFactory.newInstance();
198     Transformer arquivoXMLAssinado = transformerFactory.newTransformer();
199     arquivoXMLAssinado.transform(new DOMSource(doc), new StreamResult(saida));
200     System.out.println ("Assinatura XMLDSig efetuada com sucesso. Verifique o diretório de saída.");
201 }
202

```

Figura 44: Método para efetuar a assinatura XMLDSig.

4.7 DOWNLOAD E UPLOAD DOS METADADOS

Estágios cujas respectivas funções estão diretamente relacionadas a Internet. Ambas as classes possuem somente em um método em sua composição, conforme descrito na sequência:

- **BaixaArquivo.java:** composta pelo método download (String [] urls, String pathLocal), que recebe como parâmetro um conjunto de URLs que serão acessadas e o diretório em que o arquivo baixado será armazenado. Percorrendo o array de Strings contendo os endereços que devem ser acessados, cada um destes é transformado em um objeto do tipo URL e seu conteúdo é lido byte a byte para posteriormente salvá-lo no diretório determinado.

```

1 package arquivo;
2
3 import java.io.File;
7
10 * @author 'Luiz-Felipe'
15
16
17 public class BaixaArquivo {
18     public static File download (String [] urls, String pathLocal) {
19         String urlString = "";
20         for (int i = 0; i < urls.length; i++){
21             try {
22                 urlString = urls[i];
23                 System.out.println (stringUrl);
24                 //Encapsula a URL num objeto java.net.URL
25                 URL url = new URL(stringUrl);
26
27                 //O arquivo local terá o mesmo nome descrito na URL
28                 //String nomeArquivoLocal = url.getPath();
29                 String nomeArquivoLocal = urlString.substring(stringUrl.lastIndexOf("/"));
30                 System.out.println (nomeArquivoLocal);
31                 //O método openStream, além de realizar a conexão, cria streams de leitura
32                 InputStream is = url.openStream();
33                 FileOutputStream fos = new FileOutputStream(pathLocal+nomeArquivoLocal);
34
35                 //Leitura e gravação efetuada Byte a Byte
36                 int umByte = 0;
37                 while ((umByte = is.read()) != -1){
38                     fos.write(umByte);
39                 }
40                 is.close();
41                 fos.close();
42
43                 File f = new File(pathLocal+nomeArquivoLocal);
44
45
46                 } catch (Exception e) {
47                     e.printStackTrace();
48                 }
49             }
50         }
51         System.out.println ("##### ARQUIVO BAIXADO COM SUCESSO #####");
52         return null;
53     }
54 }
--

```

Figura 45: Código da classe responsável por efetuar o download dos arquivos.

- **UploadArquivo.java:** o upload do arquivo para ficar disponível na Web, será via FTP (*File Transfer Protocol*), o qual é o padrão da pilha TCP/IP para transferência de arquivos, sendo um protocolo genérico independente do sistema operacional. A transferência de arquivo ocorre por livre arbítrio, tendo em conta as restrições de acesso e propriedades dos mesmos. A classe que implementa o upload do arquivo é composta por três métodos: `conectaFTP` (String host, String usuário, String senha), `uploadArquivo` (String diretorioLocalDoArquivo, String nomeDoArquivo, String diretorioHost) e `encerraConexao()`. As três funções em conjunto realizam a conexão com o servidor para transferência de arquivos, determinam o arquivo que será enviado, além de seu respectivo diretório na rede e, por fim, finalizam a conexão. A Figura 46 demonstra como a classe foi codificada:

```

1 package arquivo;
2
3 import java.io.File;
13
16 * @author 'Luiz-Felipe'
21
22 public class UploadArquivo {
23
24     //Cria-se um cliente FTP
25     FTPClient ftp = null;
26 //Método para conexão com o FTP passando como parâmetro o host, usuario e senha do servidor
27 public void conectaFTP (String host, String usuario, String senha) throws Exception{
28     //Cria-se um novo cliente
29     ftp = new FTPClient();
30     ftp.addProtocolCommandListener(new PrintCommandListener(new PrintWriter(System.out)));
31     int reply;
32     ftp.connect(host);
33     reply = ftp.getReplyCode();
34     if (!FTPReply.isPositiveCompletion(reply)) {
35         ftp.disconnect();
36         throw new Exception("Erro na conexão com o servidor FTP");
37     }
38     //Passa o usuario e senha para o login
39     ftp.login(usuario, senha);
40     //Defini o tipo de arquivo para transferencia (binário, nesse caso)
41     ftp.setFileType(FTP.BINARY_FILE_TYPE);
42     ftp.enterLocalPassiveMode();
43 }
44 //Método para upload do arquivo
45 public void uploadArquivo(String diretorioLocalDoArquivo, String nomeDoArquivo, String diretorioHost)
46     throws Exception {
47     try(InputStream input = new FileInputStream(new File(diretorioLocalDoArquivo))){
48         //Define o local onde o arquivo será salvo
49         this.ftp.storeFile(diretorioHost + nomeDoArquivo, input);
50     }
51 }
52 //Método para encerrar a conexão
53 public void encerraConexao(){
54     if (this.ftp.isConnected()) {
55         try {
56             this.ftp.logout();
57             this.ftp.disconnect();
58         } catch (IOException f) {
59
60         }
61     }
62 }

```

Figura 46: Código da classe responsável por efetuar o upload dos arquivos.

4.8 RESULTADOS

A partir das classes criadas no decorrer deste trabalho, ao final foi criado um teste com a função de executar todas as tarefas com o objetivo de criar uma funcionalidade, ou seja, operar de maneira a um *pipeline*. Os objetos das classes foram criados para que seus respectivos métodos fossem invocados, conforme demonstra a Figura 47.

```

1 package testeEstagios;
2
3 import publicacao.*;
4
5
6
7 public class SimulaPipeline {
8
9 public static void main (String [] args) throws Exception {
10
11     //Instanciam-se os objetos referentes aos estágios criados para implementar uma "funcionalidade"
12     BaixaArquivo baixarArquivo = new BaixaArquivo();
13     UploadArquivo u = new UploadArquivo();
14     AssinadorXMLDSig assinador = new AssinadorXMLDSig();
15     AgregadorXML agregador = new AgregadorXML();
16
17     //Primeiramente é realizado o download dos Metadados de duas federações diferentes
18     String [] urls = {"http://metadata.aai.switch.ch/metadata.aaitest.xml",
19                     "https://ds.cafe.rnp.br/metadata/cafe-metadata.xml"};
20     baixarArquivo.download(urls,"src/test/resources");
21
22     //Na sequencia é realizada a leitura do arquivo baixado, com o objetivo de enumerar
23     //os certificados assinantes
24     //leitor.leArquivoXML("src/test/resources/metadata.aaitest.xml");
25     //Realiza a agregação dos metadados
26     agregador.agregarXML("src/test/resources/metadata.aaitest.xml",
27                          "src/test/resources/cafe-metadata.xml");
28
29
30     //OPCIONAL - Realiza a assinatura no padrão XMLDSig
31     try {
32     //     assinador.getCertificado();
33     //     assinador.assinarDocumento("src/test/resources/agregado-Switch-InCommon.xml");
34     // } catch (Exception e) {
35     //     e.printStackTrace();
36     // }
37
38     //Por fim, realiza o UPLOAD do arquivo via FTP para disponibilização
39     u.conectaFTP("chiaradia.comyr.com", "a2519534", "l Luiz2503");
40     u.uploadArquivo("src/test/resources/metadado-Agregado-Teste.xml",
41                   "metadado-Agregado-Teste.xml", "/public_html/tcc/");
42     u.encerrarConexao();
43     System.out.println("Transferência finalizada.");
44     System.out.println ("Arquivo disponibilizado em " +
45                          " http://chiaradia.comyr.com/tcc/metadado-Agregado-Teste.xml");
46
47 }
48 }

```

Figura 47: Código da classe responsável por efetuar o upload dos arquivos.

Analisando os dados dispostos no console da IDE Eclipse, foi possível constatar que o download dos arquivos foi efetuado com sucesso e os processos seguintes, como agregação e

upload, também ocorreram sem a sinalização de erros. A Figura 48 demonstra a saída do console após efetuar a compilação do código.

```
<terminated> SimulaPipeline [Java Application] C:\Program Files (x86)\Java\jdk1.7.0_25\bin\javaw.exe (18/07/2014 15:53:
http://metadata.aai.switch.ch/metadata.aaitest.xml
/metadata.aaitest.xml
https://ds.cafe.rnp.br/metadata/cafe-metadata.xml
/cafe-metadata.xml
##### ARQUIVO BAIXADO COM SUCESSO #####
220----- Welcome to Pure-FTPd [privsep] -----
220-You are user number 11 of 500 allowed.
220-Local time is now 14:54. Server port: 21.
220-This is a private system - No anonymous login
220 You will be disconnected after 3 minutes of inactivity.
USER a2519534
331 User a2519534 OK. Password required
PASS
230-OK. Current restricted directory is /
230-9 files used (0%) - authorized: 10000 files
230 41158 Kbytes used (2%) - authorized: 1536000 Kb
TYPE I
200 TYPE is now 8-bit binary
PASV
227 Entering Passive Mode (31,170,163,110,170,173)
STOR /public_html/tcc/metadado-Agregado-Teste.xml
150 Accepted data connection
226-10 files used (0%) - authorized: 10000 files
226-42851 Kbytes used (2%) - authorized: 1536000 Kb
226-File successfully transferred
226 31.548 seconds (measured here), 53.68 Kbytes per second
QUIT
221-Goodbye. You uploaded 1694 and downloaded 0 kbytes.
221 Logout.
Transferência finalizada.
Arquivo disponibilizado em      http://chiaradia.comyr.com/tcc/metadado-Agregado-Teste.xml
```

Figura 48: Console da IDE Eclipse durante a execução do código.



Figura 49: Arquivo disponibilizado.














Nome	Data de modificaç...	Tipo	Tamanho
 agregado-Switch-Cafe	01/07/2014 10:20	Documento XML	1.692 KB
 agregado-Switch-InCommon	27/06/2014 14:56	Documento XML	11.165 KB
 cafe-metadata	18/07/2014 15:54	Documento XML	746 KB
 certificado.pem	24/06/2014 10:13	Arquivo PEM	5 KB
 certificado	23/06/2014 10:46	Troca de Informaç...	4 KB
 certrepo	23/06/2014 22:39	Arquivo JKS	3 KB
 InCommon-metadata	27/06/2014 14:56	Documento XML	10.055 KB
 metadado-Agregado-Teste	18/07/2014 15:54	Documento XML	1.694 KB
 metadata.aaitest	18/07/2014 15:53	Documento XML	950 KB
 metadata	20/06/2014 15:30	Documento XML	5.587 KB
 privatekey.key	17/06/2014 10:57	Arquivo KEY	1 KB
 publickey.key	17/06/2014 10:57	Arquivo KEY	1 KB
 saidaXMLAssinado	27/06/2014 14:56	Documento XML	11.168 KB

Figura 50: Arquivos no diretório local.

Analisando o arquivo gerado que foi disponibilizado, é possível constatar que foram agregados os elementos responsáveis por descrever as entidades que possuem acesso aquele provedor, possibilitando, pois, o acesso dos usuários de um provedor ao outro, devido à realização do acordo de chaves.

5. CONCLUSÕES E TRABALHOS FUTUROS

A mudança para o paradigma da computação em nuvem hoje é uma realidade e cada vez mais usuários estão aderindo, seja criando ambientes virtuais inteiros, até simplesmente utilizando os serviços para armazenamento de arquivos. No entanto, devido a disponibilização dos serviços na rede, questões de segurança necessitam cada vez mais de soluções, para o usuário pode confiar no serviço que usa.

Dentro dessa realidade, surge o gerenciamento de identidade visando, ao mesmo tempo, tanto o controle de acesso quanto a facilidade de mecanismos como o *Single Sign-On*, no qual só é necessário apresentar a identidade do usuário uma única vez para poder acessar diversos recursos.

Nesse contexto, surgem as federações, compostas por provedores de serviço e identidade. Nelas, há grandes concentrações de informações, principalmente em meios acadêmicos.

O framework Java Metadata Aggregator foi desenvolvido com o objetivo de agir de forma eficiente promovendo a comunicação entre diferentes provedores de serviço e identidade. Contudo, durante o seu desenvolvimento, a equipe responsável chegou à conclusão que haviam criado uma ferramenta bastante genérica, com diversos casos de uso sendo descobertos a medida que novos usuários começaram a fazer uso.

Vale ressaltar que atualmente a plataforma se encontra na versão 0.8, com previsão para nova versão ainda nesse ano de 2014. Seu *roadmap* é baseado em questões levantadas por usuários em fóruns de discussão e sua documentação é alimentada por usuários em uma Wiki.

Esse trabalho teve como objetivo documentar de forma mais didática e abrangente a ferramenta, apresentando suas principais funções e configurações, baseado no estudo do código fonte do Metadata Aggregator e dos registros realizados por membros da equipe do projeto ou indivíduos dispostos a compartilhar suas experiências.

A partir desse levantamento, foi possível implementar uma aplicação que opere de maneira semelhante, criando um conjunto de estágios para a formação de uma funcionalidade. A diferença se encontra no fato da aplicação desenvolvida ser mais intuitiva e organizada de forma que o usuário saiba facilmente pelo que cada classe é responsável. Ao final, foi disponibilizado com sucesso um arquivo XML, no qual os elementos responsáveis por descrever cada entidade do provedor estivessem todos agregados, possibilitando, conforme já mencionado anteriormente, efetuar o acordo de chaves.

6. REFERÊNCIAS BIBLIOGRÁFICAS

ABNT – Associação Brasileira de Normas Técnicas. **ABNT NBR ISO/IEC 27002 – Tecnologia da informação – Técnicas de Segurança – Código de prática para a gestão de segurança da informação.** ABNT, 2005.

Camarinha-Matos, L. M., Afsarmanesh, H., e Ollus, M. (2008). **Methods and Tools for Collaborative Networked Organizations**, chapter Eco-lead And Cno Base Concepts, pages 3–32. Springer.

Chadwick, D. (2009). **Federated identity management. Foundations of Security Analysis and Design V**, pages 96–120.

Chadwick, D. e Inman, G. (2009). **Attribute aggregation in federated identity.** IEEE Computer, pages 44–53.

Damiani, E., di Vimercati, S. D. C., e Samarati, P. (2003). **Mana-ging multiple and dependable identities.** In IEEE Internet Computing, pages 29–37. IEEE.

DE SOUZA, Terezinha Batista; CATARINO, Maria Elizabete; DOS SANTOS, Paulo Cesar. **Metadados: catalogando dados na Internet.** Transinformação, v. 9, n. 2, 2012.

FELICIANO, Guilherme et al. **Gerência de Identidades Federadas em Nuvens: Enfoque na Utilização de Soluções Abertas.** Short course, XI SBSeg, Brazil, 2011.

FOWLER, Martin. **UML distilled: a brief guide to the standard object modeling language.** Addison-Wesley Professional, 2004.

FRANCISCO, R. **A importância de um plano de continuidade do negócio da organização.** Estado de Santa Catarina. Instituto Superior Tupy, 2004 (Monografia)

Jøsang, A. e Pope, S. (2005). **User centric identity management**. In AusCERT Asia Pacific Information Technology Security Conference 2005.

MELL, Peter; GRANCE, Timothy. **The NIST definition of cloud computing (draft)**. NIST special publication, v. 800, n. 145, p. 7, 2010.

OASIS (2005). **Assertions and Protocols for the SAML 2.0**. OASIS.

SHIBBOLETH. **Disponível em** <<https://shibboleth.net/>>. Acesso em Fevereiro de 2014.

SPRING. **Disponível em** <<http://spring.io/>>. Acesso em Maio de 2014.

SWITCH AAI-Federation. **Disponível em** <<http://www.switch.ch/aai/support>>. Acesso em Novembro de 2013.

SVN-SHIBBOLETH. **Disponível em** <<https://svn.shibboleth.net/java-metadata-aggregator>>. Acesso em Março de 2014.

WANGHAM, Michelle S. et al. **Gerenciamento de identidades federadas**. Minicurso-SBSeg 2010-Fortaleza-CE, 2010.

ANEXO(S) E APÊNDICE(S)

A.1 ARTIGO

Proposta de uma Aplicação Para Agregação e Publicação de Metadados

Luiz F. C. Chiaradia¹, Carla M. Westphall¹

¹Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brasil

{luizfc, carlamw}@inf.ufsc.br

***Abstract.** Share informations between different providers of a federation is a very complex task due the existing privacy agreements and the conventions used in each. The Internet2 consortium developed a Java framework to allow the development of applications in order to perform the processing of the federations' metadata. This paper presents a detailed study of Metadata Aggregator Framework and a proposed application that operates similarly aiming download, aggregation and publication of metadata obtained from providers located in federations.*

***Resumo.** O compartilhamento de informações entre os diferentes provedores de uma federação é uma tarefa bastante complexa, devido aos acordos de privacidade existentes, além das convenções utilizadas em cada uma delas. O consórcio Internet2, visando facilitar essa tarefa, desenvolveu um framework em Java que permitiria o desenvolvimento de aplicações com a finalidade de realizar o processamento dos metadados de cada federação. Este artigo apresenta um estudo detalhado do framework Metadata Aggregator e uma proposta de aplicação que opere de forma semelhante, objetivando o download, agregação e publicação de metadados obtidos de provedores localizados em federações.*

1.Introdução

A migração para os serviços de nuvem permitem ao usuário uma maior flexibilização e uma considerável redução de custos, principalmente relacionados à infraestrutura. Entretanto certas questões devem ser abordadas durante esse processo, principalmente relacionadas à segurança da informação, visto a necessidade de definir quem poderá ter acesso ou não aos arquivos disponibilizados e que tipo de acesso, ou seja, quais permissões serão concedidas. A importância do gerenciamento de identidade e acesso em um ambiente de computação em nuvem cresce na medida em que serviços que necessitem utilizar autenticação e controle de acesso a usuários surgem. Uma gestão na qual todos os usuários possuem identidades e papéis definidos é necessária a fim de que falhas, como interceptação e/ou quebra de sigilo por exemplo, sejam minimizadas durante o processo de armazenamento e compartilhamento de informações na nuvem. Foi constatado que a adoção do modelo de federações e consequentemente de sistemas de gerenciamento de identidade baseados nesse modelo já é uma realidade e utilizados em larga escala. Entretanto, existem obstáculos quando se

deseja realizar a compartilhamento de informações entre diversas federações, devido à enorme demanda de tempo para negociação das políticas de privacidade e liberação de atributos, além das convenções utilizadas em cada uma delas em seus arquivos de metadados.

2.Gerenciamento de Identidade Digital

2.1.Identidade Digital

De acordo com [WINDLEY, 2003] identidades digitais são coleções de dados que representam atributos (características associadas), preferências (desejos) e traços (características permanentes) de uma entidade. Uma identidade digital irá conter dados que descreveram de forma singular algo (chamado de sujeito ou entidade na linguagem de identidade digital) ou alguém. Além disso, possuirá informações sobre as relações com outras entidades que o sujeito possui. Funciona como um meio de autenticação e autorização e são utilizadas em cenários como redes sociais, comércio eletrônico e e-mail, por exemplo. Mesmo com todos os benefícios, principalmente no estabelecimento de um canal seguro para comunicar informações entre meios humanos e tecnológicos, a gestão da identidade digital levanta novos problemas legais, como o roubo de identidade, por exemplo. Sendo assim, existe a necessidade da criação de uma legislação em que haja o equilíbrio entre a liberdade individual, privacidade, proteção de dados pessoais e obrigações do indivíduo para com o Estado.

2.2.Gerenciamento de Identidades

O amadurecimento da Internet, que alcançou índices mais elevados de desempenho e confiabilidade, permitiu a otimização do processo de comunicação e, conseqüentemente, uma maior eficácia nas interações por meio das redes colaborativas. Uma rede colaborativa pode ser definida como “uma rede que consiste de várias entidades autônomas, heterogêneas e geograficamente distribuídas, que colaboram para encontrar um objetivo comum e compatível e cujas interações são suportadas pelas redes de computadores” [Camarinha-Matos et al. 2008]. A rede deve seguir uma série de requisitos de interoperabilidade, devido às diferenças entre os ambientes dos membros da rede, e segurança, para que as informações possam trafegar com confiança. A autorização é feita por meio de uma terceira parte confiável, também chamada de autoridade de autenticação, que irá emitir credenciais para apresentação e posterior gerenciamento de identidade para controle de acesso. [JOSANG & POPE, 2005] definem o gerenciamento de identidades como um sistema integrado de políticas, processos de negócios e tecnologias que permitem às organizações o tratamento e manipulação de identidades de seus usuários.

2.3.Sistemas de Gerenciamento de Identidade

Em qualquer cenário real, quem definirá quais informações podem ou não serem reveladas é o próprio indivíduo, enquanto no mundo digital, essa tarefa é desempenhada por um sistema de gerenciamento de identidades, utilizado para administrar, descobrir e trocar informações visando garantir a identidade de uma entidade, permitindo relações de confiança entre os membros de uma rede colaborativa. Um sistema de gerenciamento de identidades será

composto por quatro elementos, os quais são:

- **Usuário:** aquele que deseja acessar algum serviço;
- **Identidade:** nome, filiação, data de nascimento, ou seja, qualquer atributo do usuário;
- **Provedor de Identidades (Identity Provider – IdP):** após todo o processo de autenticação, o usuário receberá uma credencial, reconhecida pelos provedores de serviço;
- **Provedor de Serviços (Service Provider - SP):** após a verificação da identidade e comprovação de todos os atributos necessários para o acesso, o SP oferecerá recursos para o usuário autorizado.

Os sistemas de gerenciamento de identidade seguem modelos que apresentam formas diferentes de interação e disposição dos elementos citados anteriormente. Seus modelos são classificados em tradicional, centralizado, centrado no usuário e centralizado.

2.4.Sistemas de Gerenciamento de Identidade Baseados no Modelo de Identidade Federada

Fundamentados sobre a distribuição da tarefa de autenticação dos usuários por múltiplos provedores de identidades, estando estes dispostos em diferentes domínios administrativos, representados por uma empresa ou universidade, por exemplo, e composto por usuários. Sua abordagem visa a otimização das trocas de informações relacionadas a identidade por meio de relações de confiança construídas nas federações. Por meio de acordos entre as entidades é garantido o reconhecimento de identidades emitidas por um domínio em provedores de serviços de outros domínios, permitindo assim, a criação de Federações que permitem a redução de contratos bilaterais entre usuários e provedores de serviços.

3.Shibboleth

Fundamentado sobre padrões abertos como SAML e XML, a ferramenta Shibboleth foi criada para tratar de desafios relacionados ao gerenciamento de identidades e controle de acesso em instituições acadêmicas. Enfatiza a privacidade dos atributos do usuário, sendo que sua liberação para os provedores de serviço é controlada por uma política de privacidade definida pela instituição de origem e suas preferências pessoais. Segundo [CHADWICK 2009], todo o processo de autenticação é executado na instituição de origem do usuário, por meio de seu provedor de identidade, fazendo uso de mecanismos de autenticação presentes nessa instituição, podendo ser feita por nome de usuário e senha, X.509, etc. O fluxo de funcionamento do *Shibboleth* é representado na Figura 1 e descrito a seguir:

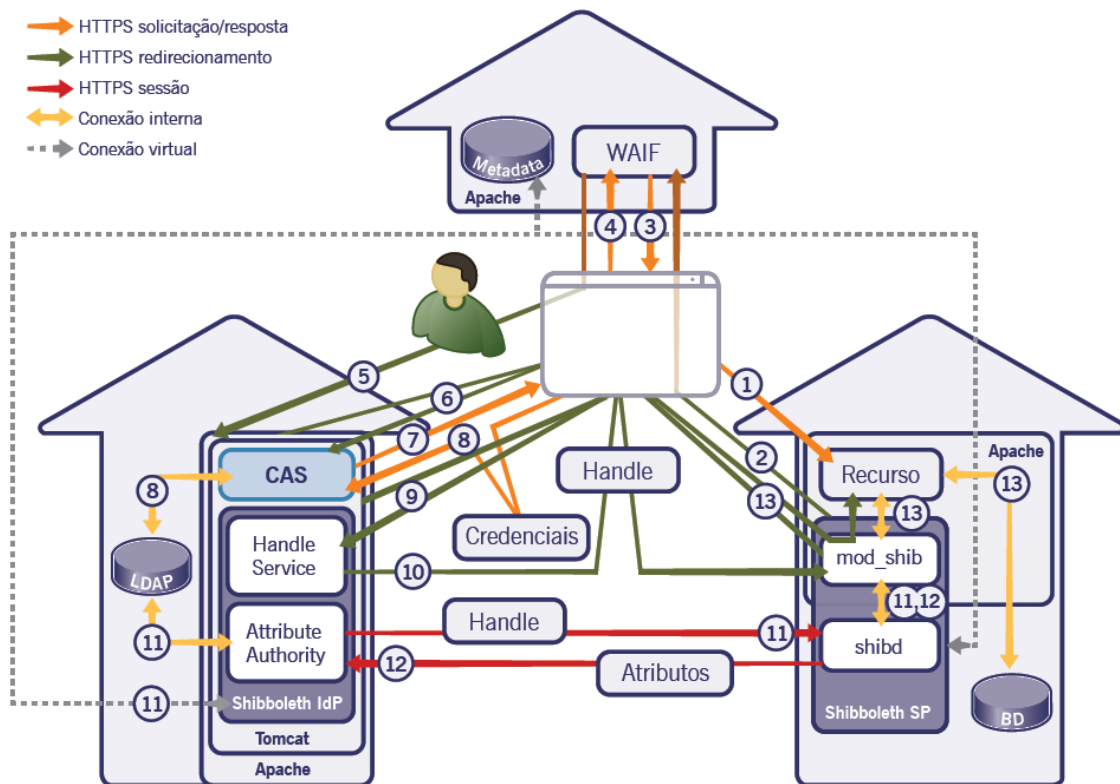


Figura 1: Funcionamento do Shibboleth [SWITCH AAI-Federation]

- **Passos 1, 2 e 3:** o usuário realiza a autenticação selecionando a sua instituição de origem.
- **Passos 4 e 5:** após o envio da requisição do usuário, o WAYF responde com um redirecionamento HTTP para o provedor de identidade do usuário.
- **Passos 6 e 7:** deverá inserir seus dados em um sistema de autenticação *Single Sign-On*. Uma vez que o usuário disponibiliza seus dados;
- **Passo 8:** é enviada uma solicitação para o sistema de autenticação (CAS), que irá verificar os dados no diretório LDAP.
- **Passos 9 e 10:** os provedores de serviço e identidade realizam as trocas dos *handles* para confirmar a autenticação do usuário.
- **Passos 11 e 12:** após a sessão HTTPS ser estabelecida entre o *shibd* e o *Attribute Authority*, é encaminhado os atributos permitidos para o provedor de serviço.
- **Passo 13:** o usuário recebe um *cookie* de sessão *Shibboleth* e é redirecionado para o recurso, que utilizará os atributos para prover um nível de autorização mais granular.

4. Metadata Aggregator

4.1. Metadados

Metadados, no que diz respeito ao Shibboleth, referem-se a dados de configuração, comumente no formato XML, utilizados para permitir que provedores de serviço (SP) e identidade (IdP) se comuniquem [INTERNET2, 2014]. Existem diversos esquemas de metadados definidos por diferentes softwares ou especificações, entretanto, o Shibboleth é baseado na especificação SAML 2.0 Metadata normatizada pela Oasis. De um modo geral, sua estrutura é composta por entidades ou grupos, papéis e contatos e organizações

4.2. Definição

O Shibboleth Metadata Aggregator (MDA) é um framework desenvolvido em Java sobre a plataforma Spring Framework. A partir de uma sequência de estágios previamente configurados, realiza o processamento de uma coleção de itens. Esses estágios são implementados como *JavaBeans*, que é uma especificação que define uma API e dita regras de configuração e comunicação entre componentes e convenções de programação, ou seja, são instâncias de classes Java que possuem algumas propriedades que são configuradas anteriormente. Cada uma dessas etapas configuradas realiza uma determinada tarefa, que irá depender diretamente de uma tarefa realizada por outra etapa para constituir uma funcionalidade propriamente dita. No contexto do MDA, o conjunto de estágios operando em conjunto para a criação de uma funcionalidade recebe o nome de “Pipeline”, mesmo termo utilizado em ambientes UNIX.

O Metadata Aggregator pode ser considerado uma ferramenta bastante genérica, o que possibilita a descoberta de novos usos para ela a medida em que é usada. De uma forma geral, ela atende qualquer caso em que um usuário que deseja ler, transformar, escrever ou procurar dados. De acordo com seus desenvolvedores [SHIBOLLETH, 2014], os seguintes casos de uso foram responsáveis pelo seu desenvolvimento:

- Uma ferramenta *Command Line* que possibilita a leitura de uma grande quantidade de metadados no padrão SAML, valide sua estrutura, filtra as entidades baseado em regras previamente definidas e assine esses arquivos digitalmente.

- Uma ferramenta *Command Line* associada a um web service que possibilita a comunicação interfederações.
- Um *web service* que opere em conjunto com os provedores de serviço e identidade, recebendo as requisições a fim de efetuar as operações mais complexas, permitindo que aqueles possam ser configurados de maneira mais simples, segura e eficaz.

4.3. Configurações

No contexto do Metadata Aggregator, um pipeline é definido como uma coleção de estágios que irá realizar uma determinada função. Cada estágio é formado por um conjunto de classes, as quais possuem seus respectivos métodos para realizar suas tarefas. A seguir, a relação de estágios disponíveis para a configuração de um *Pipeline* que podem ser mencionadas são:

- **Source Stage:** normalmente considerado o estágio inicial, é composto por classes que permitem que um determinado arquivo seja lido e preparado para sua posterior manipulação.
- **DOM Processing Stage:** por meio da utilização da tecnologia *Document Object Model*, DOM, para interação com objetos HTML e XML, este estágio contém classes que manipulam os arquivos de metadados em formato XML.
- **SAML Processing Stage:** opera de maneira semelhante ao estágio anterior, com a diferença de que os arquivos manipulados devem estar organizados no padrão SAML.
- **MDRPI Processing Stage:** implementa funcionalidades relacionadas à especificação *SAML V2.0 Metadata Extensions for Registration and Publication Information*, utilizada para criar uma estrutura de arquivos de Metadados a fim de que todos os atores possam interagir.
- **Item Metadata Stage:** estágio responsável pelo processo de marcação (*tag*) dos itens de uma determinada coleção.
- **Pipeline Structure Stage:** estágio cuja função está relacionada a criação de estruturas avançadas de *Pipelines*, como a combinação de resultados de diferentes *pipelines*, por exemplo.
- **Outros:** estágios responsáveis por funções alternativas, como serialização de itens e execução de scripts de compilação, por exemplo.

Para configurar um *Pipeline* é necessário definir seu identificador (id) e seus estágios (*stages*). Com relação a código, um *Pipeline* é definido como qualquer classe que implemente a interface `net.shibboleth.metadata.pipeline.Pipeline`.

5. Proposta

O consórcio Internet2 desenvolveu um framework em Java denominado Metadata

Aggregator, cuja função é permitir o desenvolvimento de aplicações que permitam a leitura de metadados de múltiplas fontes e, posteriormente, verifica-los, filtrá-los e manipulá-los [INTERNET2, 2014]. A documentação que acompanha o pacote disponível para download não possui exemplos de como utilizar o framework e sua Wiki explica de forma bastante clara o relacionamento entre as classes, contudo não existem modelos de usos para as mesmas. Além disso, soma-se o fato do código possuir uma lógica bastante singular, sendo constituído de estruturas que são pouco utilizadas no cotidiano, fato que foi possível constatar pela análise do mesmo e nas questões levantadas nos fóruns de discussão.

Para a solução do problema destacado, será proposto o desenvolvimento de uma aplicação *stand alone* na linguagem Java, que irá realizar o download de arquivos de metadados de diferentes federações para, na sequência, percorrer os estágios de “Leitura e Agregação”, “Assinatura” e “Upload via FTP”. Ao final desse processo, um arquivo de metadados com todas as propriedades dos utilizados no início da operação, estará disponível em um endereço da Web.

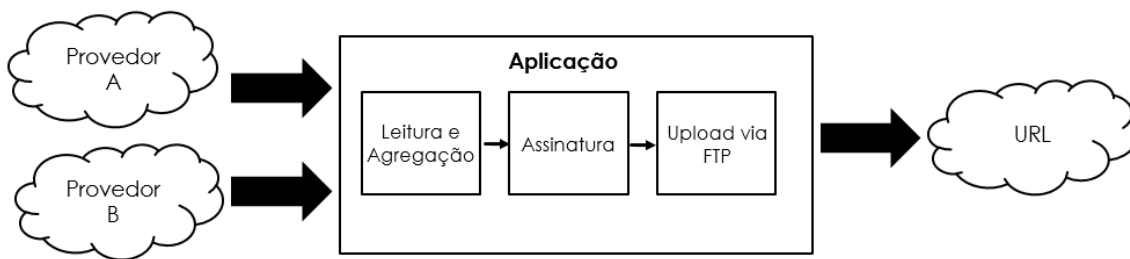


Figura 2: Modelo da aplicação proposta

Aplicações *stand alone* são completamente autossuficientes, ou seja, não necessitam de nenhum software auxiliar, como um interpretador por exemplo, sob o qual terão de ser executados. Objetivando o uso intuitivo dos recursos oferecidos pela aplicação, além de facilitar o entendimento da mesma, optou-se por separar cada estágio em uma determinada classe, a qual recebe o nome de sua tarefa.

5.1. Estágio de Leitura e Agregação

Nesta primeira etapa serão definidas duas classes responsáveis pelas funções de leitura e agregação dos metadados obtidos dos provedores. A classe `LeArquivoXML` será composta pelos pacotes da W3C para criação de árvores DOM a partir de documentos XML e possuirá apenas um método, denominado `leArquivoXML (String arquivo)`, que receberá como parâmetro uma variável do tipo `String` que representará o diretório onde o arquivo de metadados está armazenado.

A classe `AgregadorXML` também será composta por apenas um método, cujo nome é `agregarXML (String origem, String destino)` e recebe como parâmetro duas `Strings`, sendo uma com o diretório do arquivo de metadados de um determinado provedor e outra com o diretório do arquivo de metadados que “receberá” os elementos responsáveis pela descrição das entidades do provedor. Em síntese, o arquivo destino receberá os elementos do arquivo

origem.

5.2.Assinatura XML

A assinatura será realizada no padrão XMLDSig, o qual é específico para arquivos do tipo XML, podendo ser aplicada em três tipos básicos: *detached* (o arquivo e a assinatura estão em arquivo separados), *enveloped* (o documento XML e a assinatura se encontram no mesmo arquivo, de forma sequencial) e *enveloping* (o documento e a assinatura estão contidos em um “envelope” XML). Neste trabalho, a assinatura gerada será uma *enveloped*, utilizando um repositório do tipo PKCS#12, o qual permite o armazenamento de chaves privadas juntamente com o certificado de chave pública.

5.3.Download e Upload

Estágios cujas respectivas funções estão diretamente relacionadas a Internet. Ambas as classes possuem somente em um método em sua composição, conforme descrito na sequência:

- **BaixaArquivo.java:** composta pelo método `download` (`String [] urls`, `String pathLocal`), que recebe como parâmetro um conjunto de URLs que serão acessadas e o diretório em que o arquivo baixado será armazenado. Percorrendo o array de Strings contendo os endereços que devem ser acessados, cada um destes é transformado em um objeto do tipo URL e seu conteúdo é lido byte a byte para posteriormente salvá-lo no diretório determinado.
- **UploadArquivo.java:** o upload do arquivo para ficar disponível na Web, será via FTP (*File Transfer Protocol*), o qual é o padrão da pilha TCP/IP para transferência de arquivos, sendo um protocolo genérico independente do sistema operacional. A transferência de arquivo ocorre por livre arbítrio, tendo em conta as restrições de acesso e propriedades dos mesmos. A classe que implementa o upload do arquivo é composta por três métodos: `conectaFTP` (`String host`, `String usuário`, `String senha`), `uploadArquivo` (`String diretorioLocalDoArquivo`, `String nomeDoArquivo`, `String diretorioHost`) e `encerraConexao()`. As três funções em conjunto realizam a conexão com o servidor para transferência de arquivos, determinam o arquivo que será enviado, além de seu respectivo diretório na rede e, por fim, finalizam a conexão.

6.Resultados

A partir das classes criadas no decorrer deste trabalho, ao final foi criado um teste com a função de executar todas as tarefas com o objetivo de criar uma funcionalidade, ou seja, operar de maneira a um *pipeline*. Os objetos das classes foram criados para que seus respectivos métodos fossem invocados, conforme demonstra a Figura 43.

```

1 package testeEstagios;
2
3 import publicacao.*;
4
5
6 public class SimulaPipeline {
7
8
9 public static void main (String [] args) throws Exception {
10
11     //Instanciam-se os objetos referentes aos estágios criados para implementar uma "Funcionalidade"
12     BaixaArquivo baixarArquivo = new BaixaArquivo();
13     UploadArquivo u = new UploadArquivo();
14     AssinadorXMLDSig assinador = new AssinadorXMLDSig();
15     AgregadorXML agregador = new AgregadorXML();
16
17     //Primeiramente é realizado o download dos METADADOS de duas federações diferentes
18     String [] urls = {"http://metadata.aai.switch.ch/metadata.aaitest.xml",
19                     "https://ds.cafe.rnp.br/metadata/caffe-metadata.xml"};
20     baixarArquivo.download(urls,"src/test/resources");
21
22     //Na sequencia é realizada a leitura do arquivo baixado, com o objetivo de enumerar
23     //os certificados assinantes
24     //leitor.leArquivoXML("src/test/resources/metadata.aaitest.xml");
25     //Realiza a agregação dos metadados
26     agregador.agregarXML("src/test/resources/metadata.aaitest.xml",
27                          "src/test/resources/caffe-metadata.xml");
28
29
30     //OPCIONAL - Realiza a assinatura no padrão XMLDSig
31     try {
32         assinador.getCertificado();
33         assinador.assinarDocumento("src/test/resources/agregado-Switch-InCommon.xml");
34     } catch (Exception e) {
35         e.printStackTrace();
36     }
37
38     //Por fim, realiza o UPLOAD do arquivo via FTP para disponibilização
39     u.conectaFTP("chiaradia.comyr.com", "a2519534", "luiz2503");
40     u.uploadArquivo("src/test/resources/metadado-Agregado-Teste.xml",
41                   "metadado-Agregado-Teste.xml", "/public_html/tcc/");
42     u.encerrarConexao();
43     System.out.println("Transferência finalizada.");
44     System.out.println ("Arquivo disponibilizado em " +
45                          " http://chiaradia.comyr.com/tcc/metadado-Agregado-Teste.xml");
46
47 }
48 }

```

Figura 3: Código da classe responsável por efetuar o upload dos arquivos.

Analisando o arquivo gerado que foi disponibilizado, é possível constatar que foram agregados os elementos responsáveis por descrever as entidades que possuem acesso aquele provedor, possibilitando, pois, o acesso dos usuários de um provedor ao outro, devido à realização do acordo de chaves.

Analisando os dados dispostos no console da IDE Eclipse, foi possível constatar que o download dos arquivos foi efetuado com sucesso e os processos seguintes, como agregação e upload, também ocorreram sem a sinalização de erros. A Figura 4 demonstra a saída do console após efetuar a compilação do código.

```

<terminated> SimulaPipeline [Java Application] C:\Program Files (x86)\Java\jdk1.7.0_25\bin\javaw.exe (18/07/2014 15:53)
http://metadata.aai.switch.ch/metadata.aaitest.xml
/metadata.aaitest.xml
https://ds.cafe.rnp.br/metadata/caffe-metadata.xml
/caffe-metadata.xml
##### ARQUIVO BAIXADO COM SUCESSO #####
220----- Welcome to Pure-FTPd [privsep] -----
220-You are user number 11 of 500 allowed.
220-Local time is now 14:54. Server port: 21.
220-This is a private system - No anonymous login
220 You will be disconnected after 3 minutes of inactivity.
USER a2519534
331 User a2519534 OK. Password required
PASS
230-OK. Current restricted directory is /
230-9 files used (0%) - authorized: 10000 files
230 41158 Kbytes used (2%) - authorized: 1536000 Kb
TYPE I
200 TYPE is now 8-bit binary
PASV
227 Entering Passive Mode (31,170,163,110,170,173)
STOR /public_html/tcc/metadado-Agregado-Teste.xml
150 Accepted data connection
226-10 files used (0%) - authorized: 10000 files
226-42851 Kbytes used (2%) - authorized: 1536000 Kb
226-File successfully transferred
226 31.548 seconds (measured here), 53.68 Kbytes per second
QUIT
221-Goodbye. You uploaded 1694 and downloaded 0 kbytes.
221 Logout.
Transferência finalizada.
Arquivo disponibilizado em http://chiaradia.com.br/tcc/metadado-Agregado-Teste.xml

```

Figura 4: Console da IDE Eclipse durante a execução do código.

Analisando o arquivo gerado que foi disponibilizado, é possível constatar que foram agregados os elementos responsáveis por descrever as entidades que possuem acesso aquele provedor, possibilitando, pois, o acesso dos usuários de um provedor ao outro, devido à realização do acordo de chaves.














Nome	Data de modificaç...	Tipo	Tamanho
 agregado-Switch-Cafe	01/07/2014 10:20	Documento XML	1.692 KB
 agregado-Switch-InCommon	27/06/2014 14:56	Documento XML	11.165 KB
 cafe-metadata	18/07/2014 15:54	Documento XML	746 KB
 certificado.pem	24/06/2014 10:13	Arquivo PEM	5 KB
 certificado	23/06/2014 10:46	Troca de Informaç...	4 KB
 certrepo	23/06/2014 22:39	Arquivo JKS	3 KB
 InCommon-metadata	27/06/2014 14:56	Documento XML	10.055 KB
 metadado-Agregado-Teste	18/07/2014 15:54	Documento XML	1.694 KB
 metadata.aaitest	18/07/2014 15:53	Documento XML	950 KB
 metadata	20/06/2014 15:30	Documento XML	5.587 KB
 privatekey.key	17/06/2014 10:57	Arquivo KEY	1 KB
 publickey.key	17/06/2014 10:57	Arquivo KEY	1 KB
 saidaXMLAssinado	27/06/2014 14:56	Documento XML	11.168 KB

Figura 5: Arquivos no diretório local.

7. Conclusões e Trabalhos Futuros

A mudança para o paradigma da computação em nuvem hoje é uma realidade e cada vez mais usuários estão aderindo, seja criando ambientes virtuais inteiros, até simplesmente utilizando os serviços para armazenamento de arquivos. No entanto, devido a disponibilização dos serviços na rede, questões de segurança necessitam cada vez mais de soluções, para o usuário pode confiar no serviço que usa. Dentro dessa realidade, surge o gerenciamento de identidade visando, ao mesmo tempo, tanto o controle de acesso quanto as facilidade de mecanismos como o *Single Sign-On*, no qual só é necessário apresentar a identidade do usuário uma única vez para poder acessar diversos recursos.

Nesse contexto, surgem as federações, compostas por provedores de serviço e identidade. Nelas, há grandes concentrações de informações, principalmente em meios acadêmicos. O framework Java Metadata Aggregator foi desenvolvido com o objetivo de agir de forma eficiente promovendo a comunicação entre diferentes provedores de serviço e identidade. Contudo, durante o seu desenvolvimento, a equipe responsável chegou à conclusão que haviam criado uma ferramenta bastante genérica, com diversos casos de uso sendo descobertos a medida que novos usuários começaram a fazer uso. Vale ressaltar que atualmente a plataforma se encontra na versão 0.8, com previsão para nova versão ainda nesse ano de 2014. Seu *roadmap* é baseado em questões levantadas por usuários em fóruns de discussão e sua documentação é alimentada por usuários em uma Wiki.

Esse trabalho teve como objetivo documentar de forma mais didática e abrangente a ferramenta, apresentando suas principais funções e configurações, baseado no estudo do

código fonte do Metadata Aggregator e dos registros realizados por membros da equipe do projeto ou indivíduos dispostos a compartilhar suas experiências. A partir desse levantamento, foi possível implementar uma aplicação que opere de maneira semelhante, criando um conjunto de estágios para a formação de uma funcionalidade. A diferença se encontra no fato da aplicação desenvolvida ser mais intuitiva e organizada de forma que o usuário saiba facilmente pelo que cada classe é responsável. Ao final, foi disponibilizado com sucesso um arquivo XML, no qual os elementos responsáveis por descrever cada entidade do provedor estivessem todos agregados, possibilitando, conforme já mencionado anteriormente, efetuar o acordo de chaves.

References

- Camarinha-Matos, L. M., Afsarmanesh, H., e Ollus, M. (2008). Methods and Tools for Collaborative Networked Organizations, chapter Eco-lead And Cno Base Concepts, pages 3–32. Springer.
- Chadwick, D. (2009). Federated identity management. *Foundations of Security Analysis and Design V*, pages 96–120.
- Chadwick, D. e Inman, G. (2009). Attribute aggregation in federated identity. *IEEE Computer*, pages 44–53.
- Damiani, E., di Vimercati, S. D. C., e Samarati, P. (2003). Managing multiple and dependable identities. In *IEEE Internet Computing*, pages 29–37. IEEE.
- DE SOUZA, Terezinha Batista; CATARINO, Maria Elizabete; DOS SANTOS, Paulo Cesar. Metadados: catalogando dados na Internet. *Transinformação*, v. 9, n. 2, 2012.
- FELICIANO, Guilherme et al. Gerência de Identidades Federadas em Nuvens: Enfoque na Utilização de Soluções Abertas. Short course, XI SBSeg, Brazil, 2011.
- FOWLER, Martin. UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional, 2004.
- FRANCISCO, R. A importância de um plano de continuidade do negócio da organização. Estado de Santa Catarina. Instituto Superior Tupy, 2004 (Monografia)
- Jøsang, A. e Pope, S. (2005). User centric identity management. In *AusCERT Asia Pacific Information Technology Security Conference 2005*.
- MELL, Peter; GRANCE, Timothy. The NIST definition of cloud computing (draft). NIST special publication, v. 800, n. 145, p. 7, 2010.
- OASIS (2005). Assertions and Protocols for the SAML 2.0. OASIS.
- SHIBBOLETH. Disponível em <<https://shibboleth.net/>>. Acesso em Fevereiro de 2014.
- SPRING. Disponível em <<http://spring.io/>>. Acesso em Maio de 2014.
- SWITCH AAI-Federation. Disponível em <<http://www.switch.ch/aai/support>>. Acesso em Novembro de 2013.
- SVN-SHIBBOLETH. Disponível em <<https://svn.shibboleth.net/java-metadata-aggregator>>. Acesso em Março de 2014.
- WANGHAM, Michelle S. et al. Gerenciamento de identidades federadas. Minicurso-SBSeg 2010-Fortaleza-CE, 2010.