

UNIVALI - UNIVERSIDADE DO VALE DO ITAJAÍ
CAMPUS SÃO JOSÉ
CIÊNCIA DE COMPUTAÇÃO - BACHARELADO

Mecanismo de Tradução de Credenciais de Autenticação para Portais de Negócios

Bolsista: André Luiz de Oliveira
Orientadora: Michelle Silva Wangham

Relatório Final

São José – 2011

SUMÁRIO

1. INTRODUÇÃO	4
2. OBJETIVOS	5
2.1 Objetivo Geral	5
2.2 Objetivos específicos.....	5
3. FUNDAMENTAÇÃO TEÓRICA.....	6
4. MATERIAIS E MÉTODOS DE PESQUISA.....	11
4.1 Estudo bibliográfico.....	12
4.2 Definição e Modelagem do Mecanismo de Autenticação e de Tradução de Credenciais.....	12
4.3 Implementação do Protótipo	14
4.4 Avaliação do Protótipo	17
4.5 Documentação e Divulgação de Resultados	17
5. RESULTADOS E DISCUSSÕES.....	17
6. CONSIDERAÇÕES FINAIS.....	21
7. SUGESTÕES PARA DESDOBRAMENTO DA PESQUISA.....	21
REFERÊNCIAS.....	22
APÊNDICE – Descrição dos Projetos do NetBeans	24

RESUMO

Nos portais de negócios, por onde circulam informações importantes para as corporações e, muitas vezes, sigilosas, garantir o gerenciamento de identidades e a autenticação única (*Single Sign On*) são requisitos desejáveis. O projeto de pesquisa teve por objetivo desenvolver um mecanismo de autenticação única para portais de negócio do tipo *marketplace*, mesmo diante de credenciais de autenticação heterogêneas, através de um mecanismo de tradução de credenciais. O projeto envolveu (1) um estudo bibliográfico sobre gerenciamento de identidades e sobre segurança em Serviços Web, (2) a análise dos mecanismos que oferecem autenticação única (SSO) em portais de negócios e do problema da falta de compatibilidade entre credenciais de autenticação, (3) a definição e modelagem do mecanismo de autenticação e de tradução de credenciais para portais de negócios, (4) a implementação de parte do protótipo do mecanismo proposto e a sua integração em um estudo de caso para verificação da sua aplicabilidade – um portal de agência de viagens.

Palavras-chave: Portal de negócio, Autenticação única, Tradução de credenciais

1. INTRODUÇÃO

A Internet provê uma poderosa infra-estrutura de comunicação e de colaboração e, devido a isto, a realização de negócios que usufruem desta cresceu muito nos últimos anos, impulsionando assim a criação de diversas formas de modelos de negócios e de redes colaborativas. Dentre estes, destacam-se os portais de negócios colaborativos do tipo *marketplace*.

O desenvolvimento dos Serviços Web (*Web Services*), como uma tecnologia modular de interoperabilidade global, coloca-os como uma peça fundamental para a área de processos de negócios baseados em XML (BOOTH et al., 2004). Conforme constatado em Wege (2002), a tecnologia de Serviços Web, tem sido amplamente utilizada como infraestrutura de comunicação, colaboração e coordenação em portais de negócio do tipo *marketplace*.

Segundo Barton (2006), os portais de negócios devem ainda oferecer (1) autenticação única (*Single Sign On* - SSO), que permite que um usuário se autentique apenas uma vez (no portal) e use desta autenticação nos demais serviços acessados via portal de negócios e (2) gerenciamento identidades e de acesso a informações e serviços disponibilizados via portal, mas que são suportados por diferentes provedores de serviços (gerenciamento de contas de usuários e de permissões de acesso). Segundo o autor, há uma tendência para que o controle de acesso seja baseado nas credencias (atributos) dos usuários.

Os portais de negócios possuem uma série de requisitos de interoperabilidade e de segurança. A interoperabilidade é necessária para tratar vários aspectos de heterogeneidade entre os participantes do portal (organizações e usuários), que incluem as diversas plataformas computacionais utilizadas, as várias políticas (administrativas, de segurança, de negócios) às quais esses participantes estão sujeitos e as diferentes tecnologias de segurança adotadas. Um suporte a essa heterogeneidade é essencial para garantir que um portal possa atender o maior número possível de participantes. A segurança, por sua vez, é fundamental para que os participantes deste ambiente colaborativo possam depositar confiança nas suas interações com outros participantes (WANGHAM ET AL, 2010).

A **autenticação única** (*Single Sign On* – SSO) traz facilidade para os usuários, pois permite que esses passem pelo de processo de autenticação uma única vez e usufrua das

credenciais obtidas por todos os serviços que desejam acessar. Garantir tal conceito dentro de um único domínio administrativo e de segurança não é algo complexo, porém garantir o SSO em uma federação de serviços (associadas a um portal de negócios) é algo desafiador (WANGHAM ET AL, 2009).

Uma federação é uma forma de associação de parceiros de uma rede colaborativa que usa um conjunto comum de atributos, práticas e políticas para trocar informações e compartilhar serviços, possibilitando a cooperação e transações entre os membros da federação (CARMODY et al, 2009). A noção de federação é construída a partir do **gerenciamento de identidades** obtido com o uso de uma Infraestrutura de Autenticação e Autorização (AAI).

Para Mello et al (2009), um importante requisito no contexto da autenticação federada é a necessidade de tradução de credenciais de autenticação, ou seja, prover um suporte a autenticação SSO mesmo diante de parceiros que usem diferentes tecnologias de segurança. Como exemplo de infraestruturas de autenticação e autorização, tem-se o Shibboleth, Liberty Alliance, OpenAM, OpenID, Microsoft Identity Metasystem (Windows CardSpace).

Dentro deste contexto, este trabalho trata de aspectos de autenticação única federada mesmo diante de diferentes tecnologias de autenticação em portais de negócios.

2. OBJETIVOS

2.1 Objetivo Geral

Prover a autenticação única (*Single Sign On* –SSO) para portais de negócio do tipo *marketplace*, mesmo diante de credenciais de autenticação heterogêneas, através de um mecanismo para tradução de credenciais de autenticação.

2.2 Objetivos específicos

De forma a alcançar o objetivo geral, os seguintes objetivos específicos foram definidos:

- analisar os mecanismos que oferecem autenticação única (SSO) em portais de negócios e o problema da falta de compatibilidade entre credenciais de autenticação;

- definir e modelar um mecanismo de autenticação e de transposição de credenciais de autenticação compatível com os requisitos de gerenciamento de identidades em portais de negócio do tipo *marketplace*;
- verificar a aplicabilidade do mecanismo proposto através da implementação de um protótipo e da sua integração a um estudo de caso - um portal de uma agência de viagens;
- verificar eficácia e eficiência do protótipo desenvolvido através de testes de software.

3. FUNDAMENTAÇÃO TEÓRICA

Os Serviços Web seguem uma arquitetura orientada a serviços (AOS) e são componentes de softwares projetados para suportar interações entre aplicações heterogêneas sobre a Internet. De acordo com Mello et al. (2006), as principais características que os tornam uma tecnologia emergente e promissora para a realização de transações comerciais, são: (1) possuem um modelo fracamente acoplado e transparente que garante a interoperabilidade entre os serviços, sem que estes necessitem ter o conhecimento prévio de quais tecnologias estão presentes em cada lado da comunicação; (2) usam um conjunto de padrões, como o HTTP e o XML, que permite a convergência de funcionalidades de negócios díspares através de linguagens e protocolos amplamente aceitos, resultando em uma redução significativa nos custos totais de desenvolvimento de aplicações de negócio; (3) usam interfaces de serviços que são auto-descritivas e baseadas no padrão XML; (4) tornam mais fácil a composição ou a combinação de diferentes provedores, visando formar serviços mais complexos e sofisticados.

Devido às características dos portais do tipo *marketplace*, que envolve vários domínios administrativos e ambientes heterogêneos, a utilização de uma tecnologia integradora tanto em nível de plataformas computacionais quanto de sistemas de informação se faz necessário. Nestes portais, as aplicações tornam-se ainda mais visíveis, expondo suas funcionalidades, seus fluxos de negócios, processos, políticas e arquiteturas internas. Este cenário torna esta área um excelente ambiente para pesquisa.

Nestes portais, geralmente, um provedor de serviços tem preocupações de segurança com relação aos provedores ou serviços com os quais este coopera durante

uma transação comercial (CARMINATI et al., 2005). É desejável que os Serviços Web estejam acessíveis apenas para os parceiros de negócios que possuam credenciais de segurança apropriadas. Isto significa que o portal tem que conhecer as políticas de segurança dos serviços parceiros, antes de iniciar a comunicação com estes para verificar se as credencias exigidas e providas são compatíveis.

Um sistema de gerenciamento de identidades consiste na integração de políticas e processos de negócios, resultando em um sistema de autenticação de usuários aliado a um sistema de gerenciamento de atributos. (WANGHAM ET AL, 2010)

Alem de gerenciar os acessos aos provedores de serviço, o gerenciamento de identidades também utiliza de um mecanismo de autorização para garantir que somente usuários autorizados acessem o que o sistema de gerenciamento permite.

Segundo Chadwick (2009), o gerenciamento de identidades consiste em um conjunto de funções e habilidades, como administração, descoberta e troca de informações, usadas para garantir a identidade de uma entidade e as informações contidas nessa identidade, permitindo assim que relações comerciais possam ocorrer de forma segura.

Os sistemas de gerenciamento de identidades podem ser caracterizados pelos seguintes elementos (WANGHAM E AL, 2010):

- **Usuário:** aquele que deseja acessar algum serviço;
- **Identidade:** conjunto de atributos de um usuário. Pode ser seu nome, endereço, filiação, data de nascimento, etc;
- **Provedor de Identidades (*Identity Provider* – **IdP**):** responsável por emitir a identidade do usuário. Após o usuário passar por um processo de autenticação, este recebe uma credencial, dita identidade, que é reconhecida como válida pelos provedores de serviço;
- **Provedor de Serviços (*Service Provider* – **SP**):** oferece recursos a usuários autorizados, após verificar a autenticidade de sua identidade e após comprovar que a mesma carrega todos os atributos necessários para o acesso.

Identidades parciais são classificações que podem ser distribuídas para um só solicitante, como CPF, ou dependendo da classificação exigida em um contexto pode ser apresentada por uma classificação parcial diferente, como o estado civil de uma

pessoa física. O sistema de gerenciamento de identidade é a ferramenta que gerencia as identidades parciais.

Segundo Wangham et al (2010), a utilização do modelo de gerenciamento de identidades centrado no usuário em portais de negócios é interessante devido a possibilidade do usuário de selecionar quais informações deseja liberar ao provedor de serviços, respeitando a privacidade dos usuários.

Dentre as soluções de gerenciamento de identidades centradas no usuário, destaca-se a solução OpenID, que é descentralizada sendo que nenhuma autoridade central aprova ou registra as partes confiantes (*relying parties*), sítios web, ou provedores OpenID (OpenID providers). Um usuário pode escolher livremente qual provedor (de identidades) OpenID usará (Google, Yahoo, etc) e pode preservar seu identificador caso deseje futuramente mudar de provedor OpenID (ver Figura 1). A distribuição de um identificador OpenID é gratuita e não é necessário qualquer tipo de registro ou aprovação de qualquer organização (BALDONI, 2010).

Segundo a especificação OpenID (2007), o framework utiliza apenas pedidos e respostas HTTP, por isso não exige nenhuma capacidade especial do software cliente (*User-Agent*), no caso um navegador web. O framework OpenID não está vinculado à utilização de *cookies* ou a utilização de qualquer outro mecanismo específico da parte confiante (também chamado de provedor de serviço) ou depende da utilização de gerenciamento de sessão do provedor OpenID. Extensões para os navegadores Web podem simplificar a interação com o usuário final, porém não são obrigatórios. A seguir, uma visão geral das trocas de mensagens do protocolo OpenID 2.0 é descrita e também é ilustrada na Figura 1 (OPENID, 2007):

1. Através do seu navegador Web, o usuário inicia o processo de autenticação apresentando um identificador para a parte confiante (o sítio web que deseja acessar e que suporta login OpenID);
2. Com base no identificador fornecido pelo usuário, a parte confiante realiza a descoberta (*Discovery*) da URL do provedor OpenID que o usuário utiliza para autenticação;
3. (Opcional e não ilustrado na Figura) A parte confiante e o provedor OpenID estabelecem uma associação - uma chave secreta é estabelecida utilizando o protocolo *Diffie-Hellman Key Exchange*. Esta associação é estabelecida para facilitar o processo de assinatura e verificação de mensagens trocadas entre o provedor OpenID e o a parte confiante.

4. A parte confiante redireciona o navegador do usuário para o provedor OpenID com um pedido de autenticação OpenID (solicitação de uma asserção ou credencial);
5. O provedor OpenID verifica se o usuário final está autorizado a executar a autenticação OpenID que deseja fazê-lo. A maneira na qual o provedor OpenID autentica o usuário final e as políticas em torno desta autenticação estão fora do escopo da especificação (p.ex: no caso do Google, é utilizada a senha associada a conta Google como forma de autenticação);
6. O provedor OpenID redireciona o navegador Web do usuário final de volta para a parte confiante com uma asserção que indica que a autenticação está aprovada (asserções positivas) ou uma mensagem de que a autenticação falhou (asserções negativas);
7. A parte confiante verifica as informações enviadas pelo provedor OpenID, que inclui a verificação da URL retornada, das informações descobertas (provedor OpenId), do *nonce* e da assinatura. Para verificação da assinatura, a parte confiante usa a chave compartilhada estabelecida durante a associação ou a solicita diretamente ao provedor OpenID.

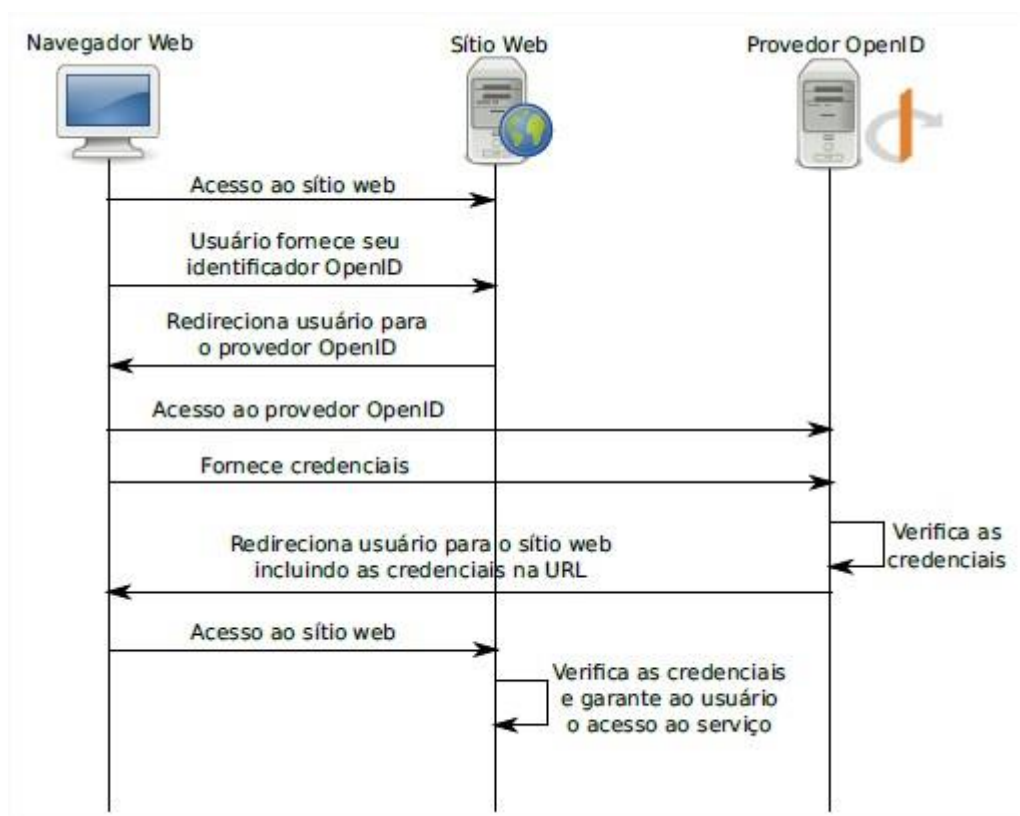


Figura 1 - Protocolo OpenID Authentication 2.0 (OPENID, 2007)

A OASIS, um órgão padronizador, lançou um conjunto de especificações para definir uma infraestrutura para troca dinâmica de informações de segurança entre parceiros de negócios, dentre estas, destacam-se as especificações relacionadas à Linguagem de Marcação para Asserções de Segurança (Security Assertion Markup Language – SAML) (OASIS, 2005).

As especificações SAML definem cinco componentes: papéis, perfis, asserções, protocolos e transporte. No componente *papéis*, além dos papéis que cada entidade pode desempenhar na infraestrutura SAML, são apresentados também os metadados que descrevem essas entidades. Os *perfis* agregam protocolos e asserções em fluxos de dados específicos para prover funcionalidades como gerenciamento de identidades e autenticação única. As *asserções* são essenciais para o gerenciamento de identidades e de contextos de segurança. Os *protocolos* são usados para requerer e transferir asserções entre as entidades. Esses protocolos podem ser mapeados em diferentes mecanismos de *transporte* (WANGHAM ET AL, 2010).

A especificação WS-Security (OASIS 2004) define mecanismos para garantir a integridade e a confidencialidade de mensagens SOAP, bem como o uso de vários tipos de credenciais de segurança que visam declarar informações de segurança (*claims*). A especificação WS-Trust (OASIS 2009b) estende a WS-Security com a definição de um protocolo para a troca e disseminação de credenciais de segurança entre diferentes domínios, bem como meios para se verificar se uma credencial é ou não confiável. Dessa forma, as partes envolvidas podem detectar e estender relações de confiança baseadas na emissão e validação de credenciais (WANGHAM ET AL, 2010).

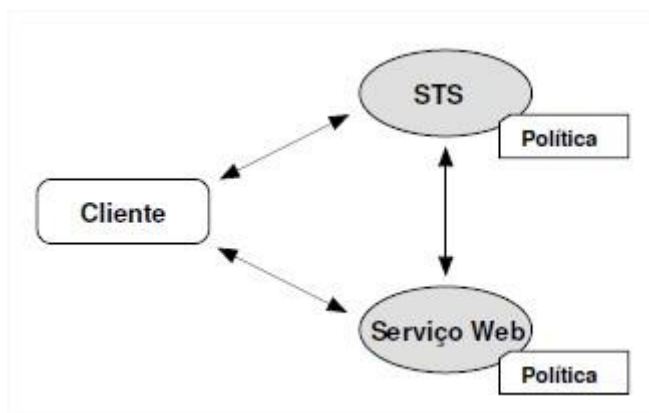


Figura 2 - Modelo de confiança da WS-Trust (WANGHAM ET AL, 2010)

A Figura 2 ilustra um cenário de mediação de confiança com base em uma terceira parte confiável que implementa um STS (OASIS, 2009). Quando o cliente deseja acessar o serviço, primeiramente obtém a política desse serviço (passo 1). Por meio dessa política, o cliente descobre o conjunto de exigências do serviço e os STSs de confiança. Caso o cliente não possua credenciais suficientes para satisfazer às exigências, pode pedir as credenciais a um dos STSs listados (passo 2). No entanto, o STS também poderá conter uma política especificando um outro conjunto de exigências e também pode apontar para outros STSs nos quais confie. Caso o cliente possua credenciais suficientes para acessar o STS de confiança do serviço, este pode requerer ao STS que emita as credenciais exigidas. O STS emitirá as credenciais e o cliente apresentará estas juntamente com a requisição ao serviço Web (passo 3). O serviço realizará a validação das credenciais passadas pelo cliente, possivelmente invocando o STS que as emitiu (passo 4), para poder permitir o acesso do cliente. (WANGHAM, 2010).

4. MATERIAIS E MÉTODOS DE PESQUISA

Nesta pesquisa, o método de experimentação foi empregado através do desenvolvimento e avaliação de um protótipo que envolve um mecanismo para tradução de credenciais de autenticação para portais de negócio do tipo *marketplace* baseados em Serviços Web. A abordagem desta pesquisa foi qualitativa, pois não fez o uso de métodos e técnicas estatísticas.

Para a construção do protótipo foram utilizados os seguintes softwares de código aberto: o ambiente de desenvolvimento NetBeans, a ferramenta de modelagem Astah, os *frameworks* METRO, JSF e RichFaces para desenvolvimento de Serviços Web, Aplicação Web e Interface Web, respectivamente, e os servidores de aplicação Glassfish e Tomcat. O protótipo implementado passou por testes de software. A gestão desse projeto se deu através de reuniões periódicas entre orientador e bolsista. A ferramenta de apoio a gestão de projetos *Cloking IT* foi utilizada para acompanhamento de todas as etapas especificadas a seguir.

4.1 Estudo bibliográfico

Esta primeira etapa foi destinada à formação do acadêmico-pesquisador, através do estudo das especificações relacionadas à arquitetura orientada a serviços, à arquitetura e desenvolvimento de portais e à segurança em Serviços Web e da literatura que trata de gerenciamento de identidades e mecanismos de autenticação única (SSO). Textos de apoio foram indicados pela orientadora e reuniões periódicas foram realizadas para auxiliar a consolidação da base teórica da pesquisa. Como indicador físico desta etapa, tem-se o texto da Seção 3. Nesta etapa, foi realizado ainda um estudo dos *frameworks* para desenvolvimento de portais de código aberto e o framework JSF foi o selecionado.

4.2 Definição e Modelagem do Mecanismo de Autenticação e de Tradução de Credenciais

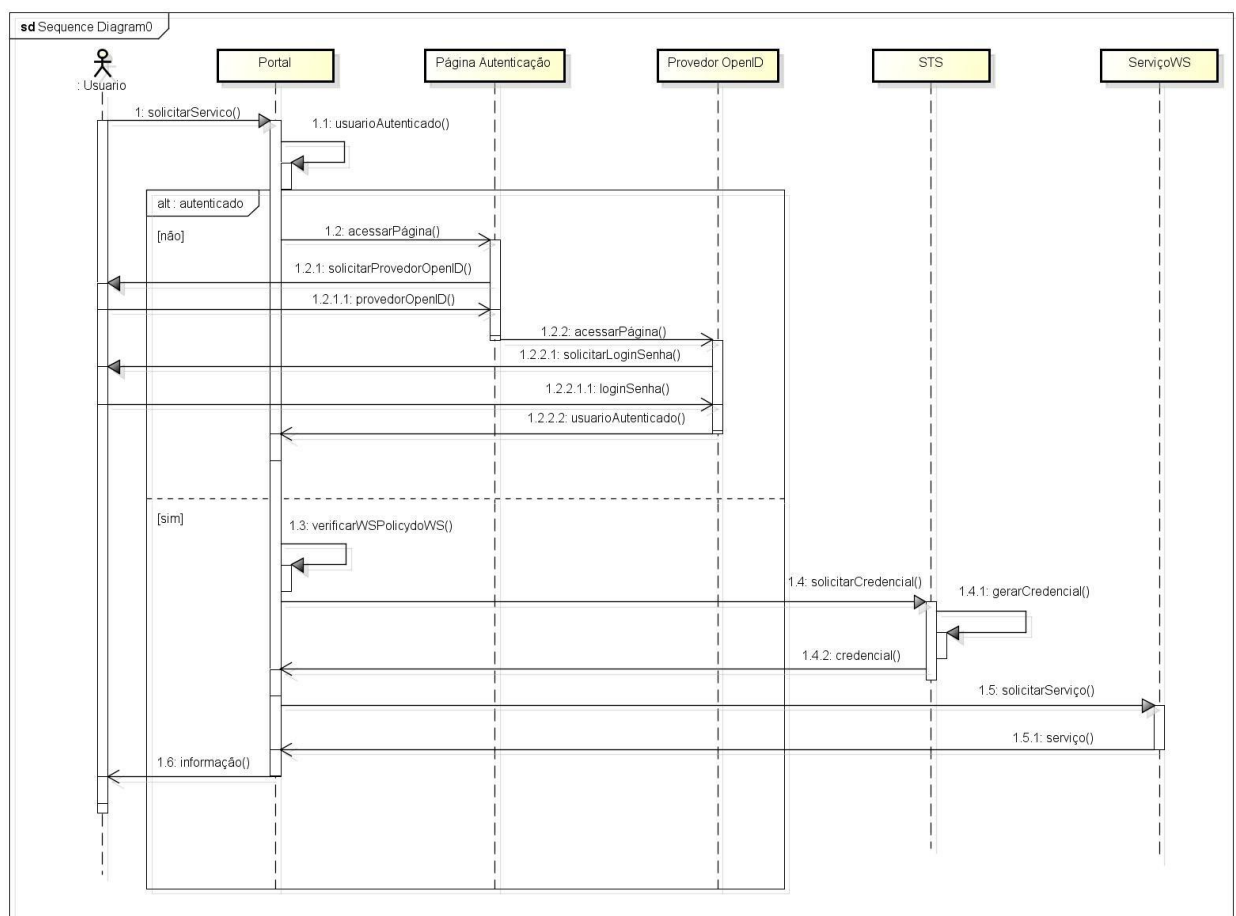
Nesta etapa, com base nas especificações de Serviços Web, nas especificações relacionadas à segurança em Serviços Web (SAML e WS-TRUST) e na especificação OPENID 2.0 a arquitetura do protótipo foi definida. Toda a modelagem do mecanismo foi realizada utilizando UML. A Figura 3 ilustra o fluxo das trocas de mensagens entre as entidades envolvidas no mecanismo de autenticação SSO e de tradução de credenciais proposto.

A seguir, uma breve descrição das entidades envolvidas:

- **Usuário:** É quem interage com o portal, fornecendo informações pessoais (provedor de identidade, login e senha) para autenticação e com isso ter acesso a serviços fornecidos pelo portal de negócio.
- **Portal de Negócios:** Com a solicitação de alguns serviços feitos pelo usuário, este é responsável por disponibilizar em uma mesma interface o acesso a serviços web que estão em diferentes provedores de serviços. Na arquitetura OpenID, o portal assume o papel de parte confiante (ou provedor de serviços). O portal assume também o papel de cliente do STS, ao solicitar a este serviço a emissão de uma credencial no formato indicado na WS-Policy do Serviço Web. Por fim, o portal assume o papel de clientes dos serviços acessíveis através do portal.
- **Serviço Web:** Um dos serviços Web que podem ser acessados através do portal que exige que o portal apresente as credenciais do usuário para acesso a seus

recursos. Este serviço possui uma WS-Policy que descreve qual tipo de credencial de autenticação este suporta.

- **Provedor OpenID:** Responsável pelo gerenciamento da identidade de usuários no fluxo. O usuário pode escolher qual provedor (de identidades) OpenID usará (Google, Yahoo, etc).
- **STS:** serviço de emissão de credenciais de autenticação que segue a especificação WS-Trust. Este serviço implementa a funcionalidade de tradução de credenciais.



powered by astah

Figura 3- Dinâmica das trocas envolvidas no mecanismo de autenticação

A seguir, tem-se a descrição do fluxo da Figura 3:

1. Solicitar Serviço – Através do seu navegador Web, o acessa o portal e tenta acessar um Serviço Web;

- 1.1. Usuário Autenticado – O Portal verifica se o usuário foi autenticado em um provedor OpenID. Se não, o portal apresenta a página de autenticação, caso contrário o usuário é redirecionado para o passo 1.3;
- 1.2. Acessar Página - O Portal apresenta a página de autenticação para o usuário;
- 1.2.1. Solicitar Provedor OpenID - A Página de autenticação solicita que o usuário escolha um provedor OpenId;
- 1.2.1.1. Provedor OpenID - Usuário seleciona provedor que deseja se autenticar;
- 1.2.2. Acessar Página - A Página de Autenticação redireciona o navegador Web do usuário para a Provedor OpenID escolhido do passo 1.2.1.1;
- 1.2.2.1. Solicitar Login Senha - O Provedor OpenID solicita ao usuário que digite seu login e senha;
- 1.2.2.1.1. Login Senha - Usuário autentica-se no provedor OpenID, fornecendo suas informações para autenticação;
- 1.2.2.2. Usuário Autenticado - Provedor OpenID envia uma mensagem para o portal, referente ao processo de autenticação do usuário;
- 1.3. Verificar WS- Policy do WS - Portal verifica na WS-Policy do serviço web qual o tipo de credencial suportado;
- 1.4. Solicitar Credencial - Portal solicita ao STS a emissão e tradução da credencial do usuário;
- 1.4.1. Gerar Credencial - STS gera a credencial;
- 1.4.2. Credencial - STS envia a credencial para o Portal;
- 1.5. Solicitar Serviço - Portal solicita serviço para o Serviço Web, juntamente com a credencial do usuário emitida pelo STS;
- 1.5.1. Serviço - ServiceWS envia para o Portal as informações do serviço;
- 1.6. Informação - Portal apresenta para o Usuário as informações solicitadas.

4.3 Implementação do Protótipo

Esta etapa de implementação foi dividida em três atividades conforme descritas a seguir:

4.3.1 - Implementação de um Portal de Negócios

A fim de verificar a aplicabilidade do mecanismo de autenticação e de tradução de credenciais proposto, um portal web foi desenvolvido em linguagem Java (plataforma Java EE 6), sendo que em sua concepção foram utilizados os frameworks

JSF e METRO e a biblioteca RichFaces.. Escolheu-se desenvolver um portal de uma agência de viagens por ser uma aplicação que possibilita a seus usuários (clientes), através de uma interface web, a consulta de informações e a reserva de recursos de hotéis, empresas aéreas e locadoras de veículos, disponibilizando desta maneira um mecanismo flexível para criação de pacotes de viagens em uma única aplicação.

JSF e RichFaces foram utilizados para compor a camada *View* do Portal, segundo o padrão de projetos *MVC*. O framework METRO foi o utilizado para a implementação do Serviço Web do Hotel e na implementação do cliente de WS que está no portal.

O framework Metro, componente do projeto Glassfish da Oracle 9, possui um extenso conjunto de bibliotecas que podem ser facilmente estendidas e implementa padrões e especificações apropriadas ao gerenciamento de identidades. Estas características o destaca como uma importante opção a ser considerada no desenvolvimento de aplicativos que incluam o gerenciamento de identidades entre seus recursos. A importância do framework Metro no cenário de Serviços Web está ligado ao fato deste conter em seu conjunto de bibliotecas, “implementações de referência” de diversas especificações da plataforma Java. Dentre estas, destaca-se a especificação JAX-WS - Java API for XML Web Services - que define as bases da criação de serviços Web com foco na linguagem Java que é parte da plataforma Java EE da Oracle.

No protótipo, os serviços web desenvolvidos, como por exemplo, o de reserva de hotéis (projeto *ServicoHotelWS*) estão no servidor de aplicação *GlassFish*, já o portal Web da Agência de Viagens (projeto *PortalServiceWS*) foi implantado no Container Web Tomcat.

Nesta etapa, nenhum mecanismo de autenticação ou de tradução de credenciais foi implantado.

4.3.2 Integração da Autenticação OpenID no Portal de Negócios

A especificação OpenID é considerada complexa, porém o *Openid4java* é uma implementação da especificação de autenticação *OpenID* que visa torna simples o uso da autenticação OpenID. No protótipo, um usuário, para acessar os serviços disponibilizados através do portal da agência de viagens, precisará passar pela autenticação SSO conforme a especificação OpenID. Os métodos implementados relacionados ao OpenID, que são da biblioteca *openid4java*, estão localizados na classe

Java OpenIdControl (localizada no control). Essa classe contém cinco métodos que correspondem ao uso da API openid4java:

- **getReturnToUrl ()** retorna a URL que o navegador redirecionará após autenticação do usuário no provedor OpenID
- **getConsumerManager ()** é usado para obter uma instância da classe principal API openid4java.
- **performDiscoveryOnUserSuppliedIdentifier ()** lida com eventuais problemas que surgem durante o processo de descoberta.
- **createOpenIdAuthRequest ()** cria a construção AuthRequest que é necessário para fazer a autenticação.

4.3.3 Implementação do Serviço Tradutor de Credenciais

O framework METRO oferece mecanismos para prover segurança para Serviços Web, dentre estes, destacam-se: WS-Policy; XWSS - XML and Web Services Security - responsável pela implementação da especificação WS-Security; e WSIT - Web Services Interoperability Technologies. responsável pela implementação das seguintes especificações: WS-Trust, WS-SecurityConversation, WS-SecurityPolicy, entre outras.

O framework Metro fornece toda funcionalidade básica para construção de um STS, definido na WS-Trust, por meio da classe *BaseSTSImpl*. Essa classe, por padrão, permite a emissão de asserções SAML 1.1 e 2.0. A implementação é extensível por meio das seguintes interfaces: *STSTokenProvider*, para suportar outros tipos de credenciais; *STSAttributeProvider*, para obter atributos de fontes variadas; e *STSAuthorizationProvider*, para emitir credenciais e declarações de autorização.

As mensagens trocadas pelo STS são bastante gerais para permitir extensões e composições futuras, cabendo a cada operação (emissão, validação, renovação, revogação, etc.) especificar quais elementos a mensagem deve conter. A mensagem de requisição é composta de um elemento XML *RequestSecurityToken* (RST) e a resposta é formada pelo *RequestSecurityTokenResponse* (RSTR). A RST contém parâmetros gerais, como o tipo de requisição, o tipo de credencial ao qual a requisição se refere, o serviço ao qual a credencial será apresentada e as exigências desse serviço. No protótipo, esta conterá os atributos obtidos do processo de autenticação OpenID. A RSTR contém, basicamente, a credencial requisitada juntamente com parâmetros da credencial (tipo, validade, contexto de utilização, etc.) descritos de forma independente do tipo de credencial, no caso do protótipo, é uma asserção SAMLv.2.

4.4 Avaliação do Protótipo

Durante o desenvolvimento do protótipo, testes de unidade foram realizados. OS erros encontrados foram corrigidos e os testes foram refeitos. Alguns testes de integração foram realizados, porém, com o protótipo não foi 100% finalizado, alguns asos de testes previstos não foram executados, dente estes, os testes de desempenho.

4.5 Documentação e Divulgação de Resultados

Os relatórios parciais, alguns resumos dos estudos realizados e o presente relatório foram redigidos nesta etapa. Pretende-se ainda até o final deste mês de Julho redigir um artigo para ser submetido em um evento nacional de iniciação científica e o resumo que será submetido para o evento de iniciação científica da UNIVALI.

5. RESULTADOS E DISCUSSÕES

A Figura 4 ilustra a página de acesso do portal. Nesta o usuário decidirá qual serviço solicitar. Podendo escolher entre *HOME*, *ACOMODAÇÕES* e *HOTEL*. *Home* é a página que dará as boas-vindas ao cliente, mostrando uma imagem convidativa para o usuário fazer seu roteiro de viagem. Enquanto Hotel tem como finalidade principal efetuar reservar e mostrar tarifário do hotel selecionado. As opções de TRANSPORTE E CONTATO não foram implementadas.

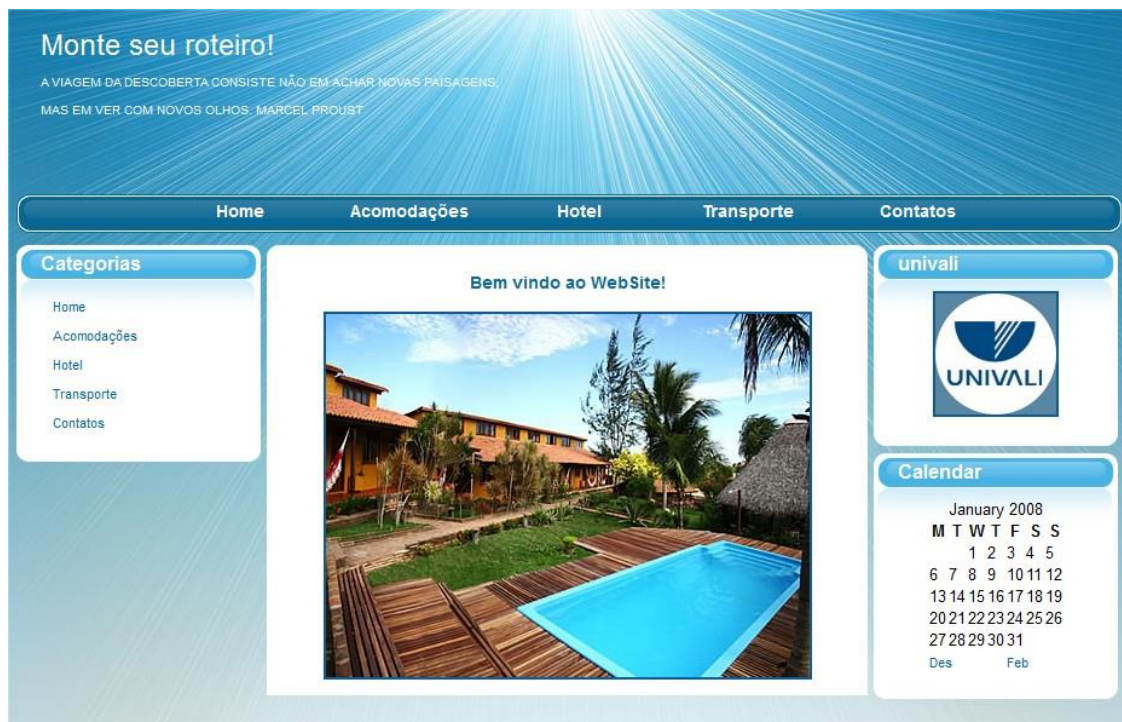
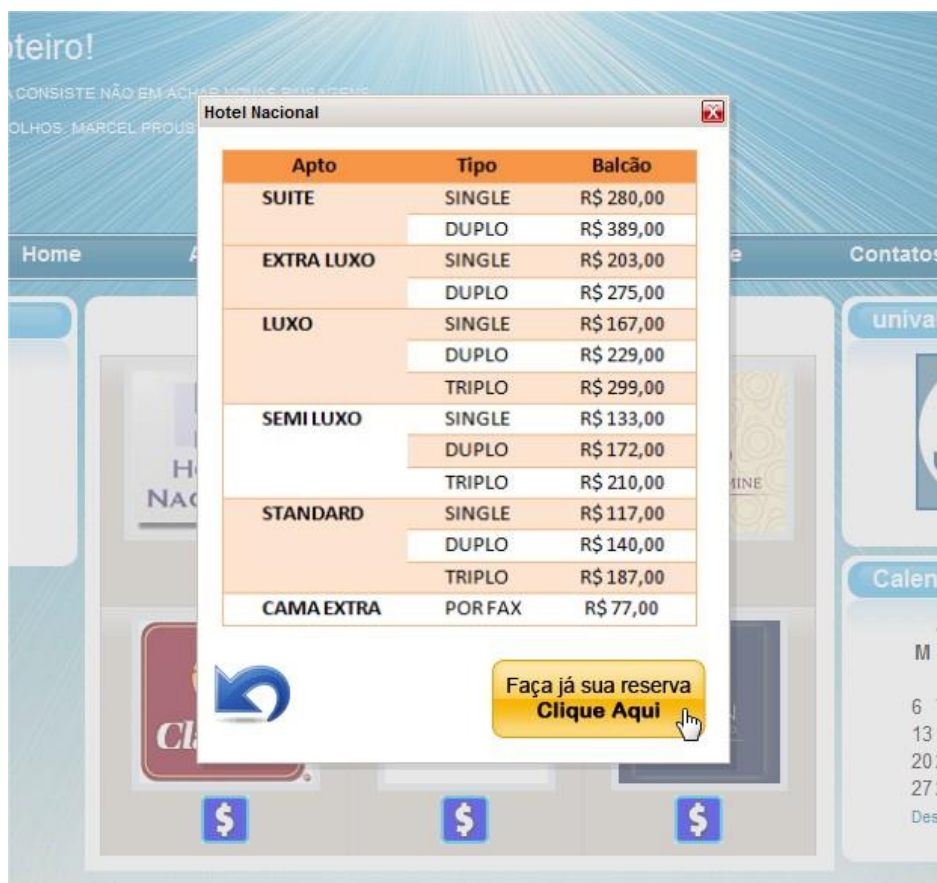


Figura 4 – Tela Inicial do Portal da Agência de Viagens

O usuário, ao selecionar no Menu o item HOTEL, uma página para a escolha de hotéis é apresentada, conforme Figura 5. Nesta página são apresentadas as logomarcas dos hotéis credenciadas no Portal. Em baixo de cada imagem tem um botão simbolizando o tarifário de cada hotel. Ao clicar neste botão, o usuário terá acesso a tabela de preços indicando o valor de diárias para cada tipo de acomodação (Figura 6). Nesta tela, o usuário tem a opção de fechar a janela utilizando o botão voltar, simbolizado com uma seta, ou fazer sua reserva clicando no botão “Faça já sua reserva/ Clique Aqui”.



Figura 5- Hóteis acessíveis via Portal (parceiros de negócio)



Apto	Tipo	Balcão
SUITE	SINGLE	R\$ 280,00
	DUPLO	R\$ 389,00
EXTRA LUXO	SINGLE	R\$ 203,00
	DUPLO	R\$ 275,00
LUXO	SINGLE	R\$ 167,00
	DUPLO	R\$ 229,00
	TRIPLO	R\$ 299,00
SEMI LUXO	SINGLE	R\$ 133,00
	DUPLO	R\$ 172,00
	TRIPLO	R\$ 210,00
STANDARD	SINGLE	R\$ 117,00
	DUPLO	R\$ 140,00
	TRIPLO	R\$ 187,00
CAMA EXTRA	POR FAX	R\$ 77,00

Figura 6 – Tarifário do Hotel

Para que o usuário faça sua reserva, o sistema solicita seu identificador *OpenID* para então verificar se o mesmo se encontra logado no provedor OpenID identificado conforme descrito pelo cliente. Não estando logado, o sistema o redirecionará para página de autenticação do provedor OpenID escolhido pelo usuário (Figura 7). Caso contrário, o mesmo não precisará se autenticar no provedor OpenID, sendo então redirecionado diretamente para a página de reservas (Figura 9).



Login:

OpenID:

ENVIAR

Figura 7- Página de Autenticação (indicação do provedor OpenID do usuário)

A Figura 8 ilustra o processo de autenticação do usuário em seu provedor OpenID.

The screenshot shows the 'myOpenID' login interface. At the top left is the 'myOpenID' logo. Below it is a green 'SIGN IN' header. A notice box states: 'You must sign in to authenticate to http://localhost:8282/PortalServiceWS/admin/reserva.faces as http://meuopenid.myopenid.com/'. The login form includes fields for 'Username' (pre-filled with the URL) and 'Password' (masked with dots). There is a 'Stay signed in' checkbox and 'Sign In' and 'Cancel' buttons. On the right, there's a section for 'OPÇÕES' with links for 'Home', 'Recuperar Conta', and 'OpenID Site Directory'. The footer contains links for 'Help', 'Privacy', and 'Language: Not set', along with copyright information for Janrain, Inc.

Figura 8 – Autenticação do usuário no Provedor MyOpenID

Na página de reservas, o usuário preencherá as lacunas com seus dados pessoais: nome, telefone, email, como também a quantidade de pessoas, tipo de apartamento, se possui criança de colo, período da estadia e as observações. Quando as informações forem preenchidas e enviadas, estas aparecerão para o serviço Web do hotel escolhido.

The screenshot displays a reservation form titled 'Reserva:'. It contains several input fields: 'Nome:', 'Telefone:', 'Email:', 'Nr. Pessoas:' (a dropdown menu showing '0'), 'Apartamento:' (a dropdown menu with 'Enter some value'), 'Criança de Colo:' (radio buttons for 'Sim' and 'Não'), and 'Período:' (a date range selector showing 'A'). Below these is a large blue text area for 'Observação:'. At the bottom of the form is a blue button labeled 'ENVIAR'. On the right side of the page, there are partial views of other sections: 'unival' and a 'Calen' (calendar) showing dates like 6, 7, 13, 14, 20, 21, 27, 28, and 'Des'.

Figura 9 – Página para efetuar a reserva no Hotel (somente para usuários autenticados)

6. CONSIDERAÇÕES FINAIS

O uso de portais de negócios na Internet têm apresentado um crescimento muito acentuado nos últimos anos (BELLAS, 2004). Para a realização de transações comerciais através de portais de negócio, por onde circulam informações importantes para as corporações e, muitas vezes, sigilosas, estabelecer o gerenciamento de identidades entre os parceiros envolvidos e traduzir credenciais de autenticação utilizadas na composição são necessidades críticas.

O mecanismo de segurança desenvolvido neste projeto oferece meios que auxiliam clientes e provedores de serviços no gerenciamento de identidades (ou credenciais) e na tradução de credencias de autenticação. Um protótipo do mecanismos de autenticação única e de tradução de credenciais foi desenvolvido, porém este ainda não está totalmente concluído. O mecanismos de tradução de credencias (STS) ainda não está totalmente funcional, porém é possível que este possa ser concluído até o final do mês de julho ou em agosto.

Destaca-se que o cronograma de implementação da solução proposta atrasou devido alguns problemas e desafios que o bolsista enfrentou no decorrer de sua pesquisa. Dentre estes, destacam-se: (1) pouca bibliografia em português e, como o aluno possui dificuldades em ler em inglês, a etapa de estudo bibliográfico foi concluída com atraso, (2) apesar de ter um bom conhecimento em Java, o bolsista não conhecia os framework JSF e METRO, o que exigiu um bom tempo para o desenvolvimento do portal de negócios e dos serviços web, (3) a tecnologia OpenID não possui documentação em português e, poucos exemplos foram encontrados que utilizavam a biblioteca openid4J, por fim, (4) o aluno encontrou muita dificuldade no desenvolvimento do serviço de tradução de credenciais a partir do STS, uma vez que para esta etapa da implementação muitas bibliotecas que não eram conhecidas pelo bolsista precisaram ser utilizadas (WSIT e openSAML).

7. SUGESTÕES PARA DESDOBRAMENTO DA PESQUISA

O mecanismo de segurança desenvolvido neste projeto deve prover meios que auxiliem clientes e provedores de serviços no gerenciamento de identidades (ou credenciais) e na tradução de credencias de autenticação. O modelo e o protótipo

desenvolvido se concentraram em somente duas tecnologias de autenticação, OpenID e SAML. O protótipo não foi finalizado, logo a primeira sugestão é finalizá-lo para que os testes possam ser concluídos. Consta-se que o serviço responsável para tradução de credenciais necessita ser aprimorado para tratar outras credenciais de autenticação como certificados X.509 e os *InformationCards*.

Outro ponto importante é que o mecanismo proposto ainda não contempla todas as necessidades exigidas em portais de negócios, como por exemplo, lidar com um processo de negócio executável (CARMINATI et al., 2005, MELLO et al, 2009). É importante que a criação sob demanda e customizada de processos de negócios (composição dinâmica de Serviços Web), realizada nos portais, seja ciente da segurança, levando em conta os riscos de segurança e os requisitos de todos os parceiros envolvidos (provedores e requisitantes), expressos em suas políticas de segurança. É desejável que processos de negócios não deixem de ser realizados por falta de compatibilidade das credenciais de autenticação entre os parceiros envolvidos.

REFERÊNCIAS

- BALDONI, R. (2010). Federated Identity Management Systems in e- Government: the Case of Italy. *Electronic Government: An International Journal*, 8(1).
- BARTON, T. et al. Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, GridShib, and MyProxy. In: ANNUAL PUBLIC KEY INFRASTRUCTURE R&D WORKSHOP, 5th, 2006, Gaithersburg, MA. **Proceedings...** Gaithersburg, National Institute of Standards and Technology Interagency Series (NISTIR 7313), 2006. p. 64-67.
- BELLAS, F. Standards for Second-Generation Portals. **IEEE Internet Computing**, v. 8, n. 4, p. 54-60, Mar. 2004.
- BOOTH, D. et al. **Web Services Architecture**. W3C Working Group Note, 2004. Disponível em: <<http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>>. Acesso em: 25 jan 2011
- CARMINATI, B.; FERRARI, E.; HUNG, P. C. K. Web service composition: A security perspective. In: INTERNATIONAL WORKSHOP ON CHALLENGES IN WEB INFORMATION RETRIEVAL AND INTEGRATION (WIRI'05), 2005, Tokyo. **Proceedings...** Los Angeles, CA, IEEE Computer Society, 2005. p. 248 -253.
- CARMODY, S., ERDOS, M., HAZELTON, W. HOEHN, B. MORGAN, T. SCAVO e D. WASLEY. **InCommon Technical Requirements and Information**. Last

update 18.12.2009. Disponível em:
<<http://www.incommonfederation.org/technical.html>>. Acesso em: 10 jul. 2011.

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

MELLO, E. R. de; WANGHAM, M. S.; FRAGA, J.; CAMARGO, E. Segurança em Serviços Web. In: **Livro de Minicursos do VI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais**. Santos: SBC, 2006. p. 1–48.

MELLO, E. R. de; WANGHAM, M. S.; FRAGA, J. S.; CAMARGO, E. T.; BOGER, D. S. . A Model for Authentication Credentials Translation in Service Oriented Architecture. In **Transactions on Computational Sciences Journal - Security in Computing**, v. 5430, p. 68-86, 2009.

OASIS (2004). Web Services Security: SOAP Message Security 1.0. OASIS. <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>.

OASIS (2005). Security Assertion Markup Language (SAML) 2.0 Technical Overview. OASIS.

OPENID (2007). Openid authentication 2.0. OPENID. http://openid.net/specs/openid-authentication-2_0.html.

OASIS (2009). WS-Trust 1.4.

WEGE, C. Portal server technology. **IEEE Internet Computing**, v.6, n.3, p. 73–77, 2002.

WANGHAM, Michelle S., DE MELLO, Emerson Ribeiro, BOGER, Davi da Silva, GUERIOS, Marlon, FRAGA, Joni da Silva. Minicurso de Gerenciamento de Identidades Federadas. **Gerenciamento de Identidades Federadas**, 2010.

APÊNDICE – Descrição dos Projetos do NetBeans

A seguir, estão descritas algumas atividades realizadas no desenvolvimento do protótipo nos projetos *ServicoHotelWS*, referente ao Serviço Web do Hotel, e no projeto *PortalWS*. A Figura X apresenta o diagrama de classes do projeto *ServicoHotelWS*.

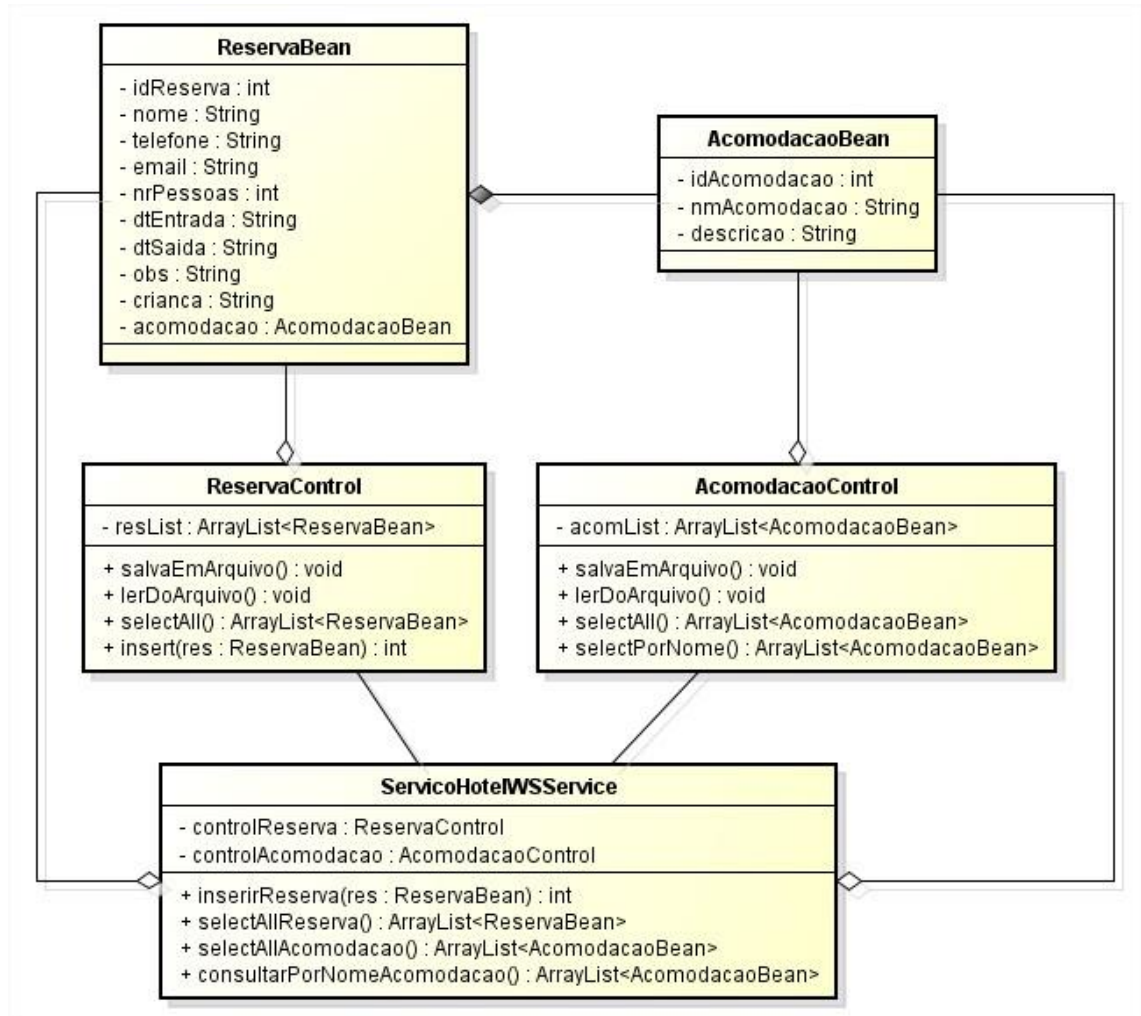
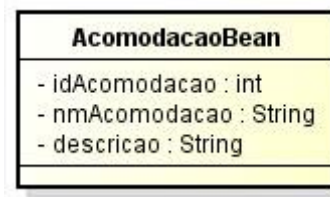


Figura XX – Diagrama de classe do *ServicoHotelWS*

Projeto *ServicoHotelWS*

- Antes de criar o serviço web, certificou-se que o GlassFish foi registrado com o IDE NetBeans. A lista de servidores registrados podem ser vistos a partir da opção do menu Ferramentas> Servidores;
- Criou-se o projeto *ServicoHotelWS* com a configuração *Java Web>Aplicação Web*, servidor GlassFish Server 3 e versão Java JEE 6;

- Criou-se classe Java *AcomodacaoBean* no pacote *bean* e a declarou como *public class AcomodacaoBean implements Serializable* e os *Gets* e *Sets* foram gerados:



- Criou-se a classe Java *ReservaBean* no pacote *bean* e a declarou como *public class ReservaBean implements Serializable* e os *Gets* e *Sets* foram gerados:



- Criou-se a classe Java *ReservaControl* no pacote *control*:



Implementação dos métodos da Classe *ReservaControl*:

```
public void salvaEmArquivo() {
    try {
        FileOutputStream out = new FileOutputStream("resList.dad");
        ObjectOutputStream objectout = new ObjectOutputStream(out);
        objectout.writeObject(resList);
        objectout.close();
        out.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

```

public void lerDoArquivo() {
    try {
        FileInputStream in = new FileInputStream("resList.dad");
        ObjectInputStream objectIn = new ObjectInputStream(in);
        resList = (ArrayList<ReservaBean>) objectIn.readObject();
        objectIn.close();
        in.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public int insert(ReservaBean res) {
    int idAux = 1;
    salvaEmArquivo();
    lerDoArquivo();
    if (!resList.isEmpty()) {
        idAux = resList.get(resList.size() - 1).getIdReserva() + 1;
    }
    res.setIdReserva(idAux);
    resList.add(res);
    salvaEmArquivo();
    return idAux;
}

public ArrayList<ReservaBean> selectAll() {
    lerDoArquivo();
    return resList;
}

```

- Criou-se a classe Java *AcomodacaoControl* no pacote *control*:

AcomodacaoControl
- acomList : ArrayList<AcomodacaoBean>
+ salvaEmArquivo() : void + lerDoArquivo() : void + selectAll() : ArrayList<AcomodacaoBean> + selectPorNome() : ArrayList<AcomodacaoBean>

Implementação dos métodos da Classe *AcomodacaoContro*:

```

public void salvaEmArquivo() {
    try {
        FileOutputStream out = new FileOutputStream("acomLista.dad");
        ObjectOutputStream objectout = new ObjectOutputStream(out);
        objectout.writeObject(acomList);
        objectout.close();
        out.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

```

```

public void lerDoArquivo() {
    try {
        FileInputStream in = new FileInputStream("acomLista.dad");
        ObjectInputStream objectIn = new ObjectInputStream(in);
        acomList = (ArrayList<AcomodacaoBean>) objectIn.readObject();
        objectIn.close();
        in.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

```

public ArrayList<AcomodacaoBean> selectAll() {
    lerDoArquivo();
    return acomList;
}

```

```

public ArrayList<AcomodacaoBean> selectPorNome(String descricao) {
    ArrayList<AcomodacaoBean> acoss =
        new ArrayList<AcomodacaoBean>();
    lerDoArquivo();
    for (AcomodacaoBean acom : acomList) {
        if (acom.getDescricao().equals(descricao)) {
            acoss.add(acom);
        }
    }
    return acoss;
}

```

- Criou-se a classe *ServicoHotelWSService* com a configuração *Web Service>Web Service* no pacote *ws*:

ServicoHotelWSService
- controlReserva : ReservaControl - controlAcomodacao : AcomodacaoControl
+ inserirReserva(res : ReservaBean) : int + selectAllReserva() : ArrayList<ReservaBean> + selectAllAcomodacao() : ArrayList<AcomodacaoBean> + consultarPorNomeAcomodacao() : ArrayList<AcomodacaoBean>

Implementação dos métodos `ServicoHotelWSService`:

```
@WebMethod(operationName = "inserirReserva")
public int inserirReserva(@WebParam(name = "reserva")
ReservaBean res) {
    return this.controlReserva.insert(res);
}
```

```
@WebMethod(operationName = "selectAllReserva")
public ArrayList<ReservaBean> selectAllReserva() {
    return this.controlReserva.selectAll();
}
```

```
@WebMethod(operationName = "selectAllAcomodacao")
public ArrayList<AcomodacaoBean> selectAllAcomodacao() {
    return this.controlAcomodacao.selectAll();
}
```

```
@WebMethod(operationName = "consultarPorNomeAcomodacao")
public ArrayList<AcomodacaoBean> consultarPorNomeAcomodacao
(@WebParam(name = "nome") String nome) {
    return this.controlAcomodacao.selectPorNome(nome);
}
```

- Com o botão direito do mouse no `WebService Serviços`
`Web>ServicoHotelWSService`, selecione propriedade e aparecerá o *URL* da *WSDL*;
- Após colocar a *URL da WSDL* no navegador, é possível verificar se o serviço foi está pronto e disponível para receber invocações.

Projeto *PortalWS*:

- Criou-se o projeto *PortalWS* com a configuração *Java Web>Aplicação Web*, servidor Tomcat 6.0 e versão Java JEE 6;
- Criou-se a classe *RegistrationBean* no pacote bean, a declarou como *public class RegistrationBean implements Serializable* e os *Gets* e *Sets* foram gerados:

RegistrationBean
- openId : String
- fullName : String
- emailAddress : String
- zipCode : String
- dateOfBirth : Date
- favoriteColor : String

- Criou-se a classe *ServicoHotelWSService* com a configuração *Web Service>Cliente para serviço Web*, Foi informado a *WSDL*

<http://localhost:8080/ServicoHotelWSService/ServicoHotelWSService?wsdl>
(URL que foi mostrada na propriedade da classe *ServicoHotelWSService* do projeto *ServicoHotelWS*):.

ServicoHotelControl
+ inserirReserva(reserva : ReservaBean) : int + selectAllReserva() : List<ReservaBean> + selectAllAcomodacao() : List<AcomodacaoBean> + consultarPorNomeAcomodacao(nome : String) : List<AcomodacaoBean>

Implementação dos métodos da Classe *ServicoHotelControl*:

```
public static int inserirReserva(ReservaBean reserva) {
    ServicoHotelWSServiceService service =
        new ServicoHotelWSServiceService();
    ServicoHotelWSService port =
        service.getServicoHotelWSServicePort();
    return port.inserirReserva(reserva);
}

public static List<ReservaBean> selectAllReserva() {
    ServicoHotelWSServiceService service =
        new ServicoHotelWSServiceService();
    ServicoHotelWSService port =
        service.getServicoHotelWSServicePort();
    return port.selectAllReserva();
}

public static List<AcomodacaoBean> selectAllAcomodacao() {
    ServicoHotelWSServiceService service =
        new ServicoHotelWSServiceService();
    ServicoHotelWSService port =
        service.getServicoHotelWSServicePort();
    return port.selectAllAcomodacao();
}

public static List<AcomodacaoBean> consultarPorNomeAcomodacao(
    String nome) {
    ws.ServicoHotelWSServiceService service =
        new ServicoHotelWSServiceService();
    ws.ServicoHotelWSService port =
        service.getServicoHotelWSServicePort();
    return port.consultarPorNomeAcomodacao(nome);
}
```

- Colocou-se na pasta lib do projeto a biblioteca *openid4java-full-0.9.5.jar*, para implementação da próxima classe;
- Criou-se a classe *OpenIdControl* no pacote control:

OpenIdControl
- ret : DiscoveryInformation - consumerManager : ConsumerManager
+ performDiscoveryOnUserSuppliedIdentifier(userSuppliedIdentifier : String) : DiscoveryInformation + createOpenIdAuthRequest(discoveryInformation : DiscoveryInformation, returnUrl : String) : AuthRequest + getConsumerManager() : ConsumerManager + getReturnToUrl() : String

```

@SuppressWarnings("unchecked")
public static DiscoveryInformation performDiscoveryOnUserSuppliedIdentifier(
    String userSuppliedIdentifier) {
    DiscoveryInformation ret = null;
    ConsumerManager consumerManager = getConsumerManager();
    try {

        List<DiscoveryInformation> discoveries =
            consumerManager.discover(userSuppliedIdentifier);
        ret = consumerManager.associate(discoveries);
    } catch (DiscoveryException e) {
        String message = "Um erro ocorreu durante a descoberta!";
        throw new RuntimeException(message, e);
    }
    return ret;
}

```

```

public static AuthRequest createOpenIdAuthRequest(
    DiscoveryInformation discoveryInformation,
    String returnUrl) {
    AuthRequest ret = null;
    try {
        ret = getConsumerManager().authenticate(
            discoveryInformation, returnUrl);
        SRegRequest sRegRequest = SRegRequest.createFetchRequest();
        sRegRequest.addAttribute("email", false);
        sRegRequest.addAttribute("fullname", false);
        sRegRequest.addAttribute("dob", false);
        sRegRequest.addAttribute("postcode", false);
        ret.addExtension(sRegRequest);
    } catch (Exception e) {
        String message =
            "Exception occurred while building AuthRequest object!";
        ret = null;
        throw new RuntimeException(message, e);
    }
    return ret;
}

```

```

private static ConsumerManager getConsumerManager() {
    try {
        if (consumerManager == null) {
            consumerManager = new ConsumerManager();
            consumerManager.setAssociations(
                new InMemoryConsumerAssociationStore());
            consumerManager.setNonceVerifier(
                new InMemoryNonceVerifier(10000));
        }
    } catch (ConsumerException e) {
        String message = "Exception creating ConsumerManager!";
        throw new RuntimeException(message, e);
    }
    return consumerManager;
}

public static String getReturnToUrl() {
    return "http://localhost:8282/PortalWS/admin/reserva.faces";
}

```

- Colocou-se na pasta lib do projeto a biblioteca *jsf-api.jar*, para implementação da próxima classe;
- Criou-se a classe Java *ReservaMB* no pacote *mb* e a declarou como *public class* *ReservaMB* implements *Serializable* e os *Gets* e *Sets* foram gerados:

ReservaMB
- reserva : ReservaBean - hotelControl : ServicoHotelControl - listaReserva : List<ReservaBean> - listaAcomodacao : List<AcomodacaoBean> - nomeAcomodacao : String
+ ReservaMB() + getItensAcomodacao() : List<SelectItem> + inserir() : String

```

public ReservaMB() {
    System.out.println(" >>>>Contrutor da Reserva <<<<");
    reserva = new ReservaBean();
    hotelControl = new ServicoHotelControl();
    listaReserva = hotelControl.selectAllReserva();
    listaAcomodacao = hotelControl.selectAllAcomodacao();
}

```



```

public List<SelectedItem> getItensAcomodacao() {
    List<SelectedItem> itens =
        new ArrayList<SelectedItem>(listaAcomodacao.size());
    for(AcomodacaoBean p : listaAcomodacao){
        itens.add(new SelectedItem(
            p.getNmAcomodacao(),
            p.getNmAcomodacao(),
            p.getDescricao(),
            true));
    }
    return itens;
}

```

```

public String inserir(){
    List<AcomodacaoBean> listAco =
        hotelControl.consultarPorNomeAcomodacao(nomeAcomodacao);
    if(listAco.size()>0){
        reserva.setAcomodacao(listAco.get(0));
    }
    int id = hotelControl.inserirReserva(reserva);
    reserva = new ReservaBean();
    listaReserva = hotelControl.selectAllReserva();
    if(id!=-1){
        System.out.println(" >>>> Reserva Inserida <<<<");
        return "index";
    }else{
        System.out.println(" >>>> ERRO - Reserva Não Inserida <<<<");
        return "index";
    }
}

```

- Configuração feita no arquivo *web>WEB-INF>faces-config.xml* para os componentes das páginas *.xhtml* acessar a classe *ReservaMB* com *managed-bean-name reservaMB*:

```

<managed-bean>
    <managed-bean-name>reservaMB</managed-bean-name>
    <managed-bean-class>mb.ReservaMB</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>

```

- Criou-se a classe Java *LoginMB* no pacote *mb* e a declarou como *public class LoginMB implements Serializable* e os *Gets* e *Sets* foram gerados:

LoginMB
- openID : String
+ autenticar() : String


```

public String autenticar() {

    ExternalContext extCon =
        FacesContext.getCurrentInstance().getExternalContext();
    HttpSession session = (HttpSession) extCon.getSession(true);
    DiscoveryInformation discoveryInformation =
        OpenIdControl.performDiscoveryOnUserSuppliedIdentifier(openID);
    session.setAttribute("discoveryInformation", discoveryInformation);
    AuthRequest authRequest =
        OpenIdControl.createOpenIdAuthRequest(
            discoveryInformation,
            OpenIdControl.getReturnToUrl());

    try {
        extCon.redirect(authRequest.getDestinationUrl(true));
    } catch (IOException ex) {
        System.out.println(ex.toString());
    }
    return "index";
}

```

- Configuração feita no arquivo *web>WEB-INF>faces-config.xml* para os componentes das páginas *.xhtml* acessar a classe *LoginMB* com *managed-bean-name loginMB*:

```

<managed-bean>
    <managed-bean-name>loginMB</managed-bean-name>
    <managed-bean-class>mb.LoginMB</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

```

Projeto *ProjectSTS*:

- Criou-se o projeto *ProjectSTS* com a configuração *Java Web>Aplicação Web*, servidor Glassfish Server 3 e versão Java JEE 6;
- Criou-se a classe *STSService* com a configuração *Web Service>Serviço de símbolo seguro(STS)* no pacote *sts* e declarou como:

STSService
- context : WebServiceContext
+ invoke(rstElement : Source) : Source
+ getMessageContext() : MessageContext

```

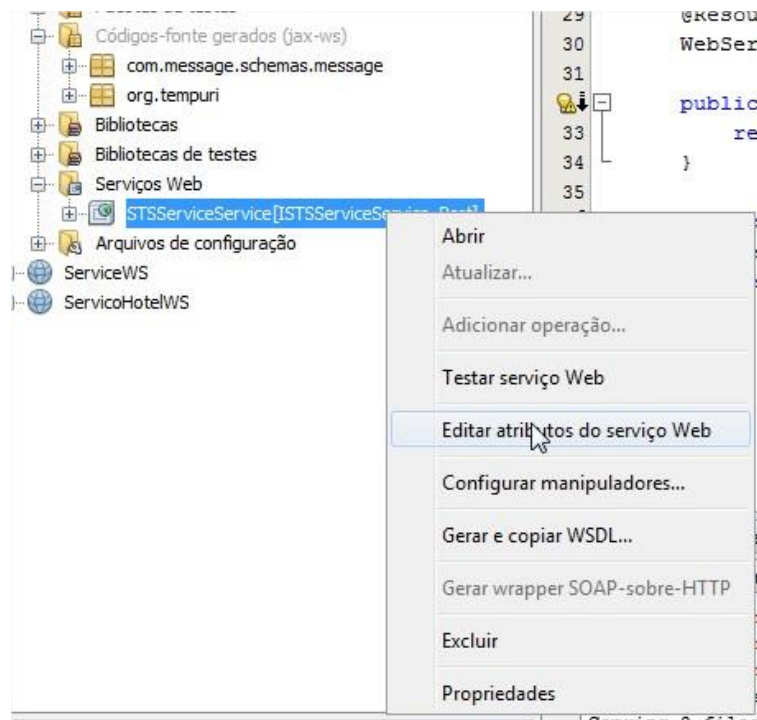
@WebServiceProvider(serviceName = "STSServiceService",
    portName = "ISTSServiceService_Port",
    targetNamespace = "http://tempuri.org/",
    wsdlLocation = "WEB-INF/wsdl/STSService/STSServiceService.wsdl")
@ServiceMode(value = Mode.PAYLOAD)
public class STSService extends com.sun.xml.ws.security.trust.sts.BaseSTSImpl
    implements Provider<Source> {

    public Source invoke(Source rstElement) {
        return super.invoke(rstElement);
    }

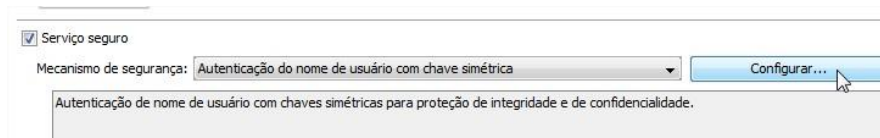
    protected MessageContext getMessageContext() {
        MessageContext msgCtx = context.getMessageContext();
        return msgCtx;
    }
}

```

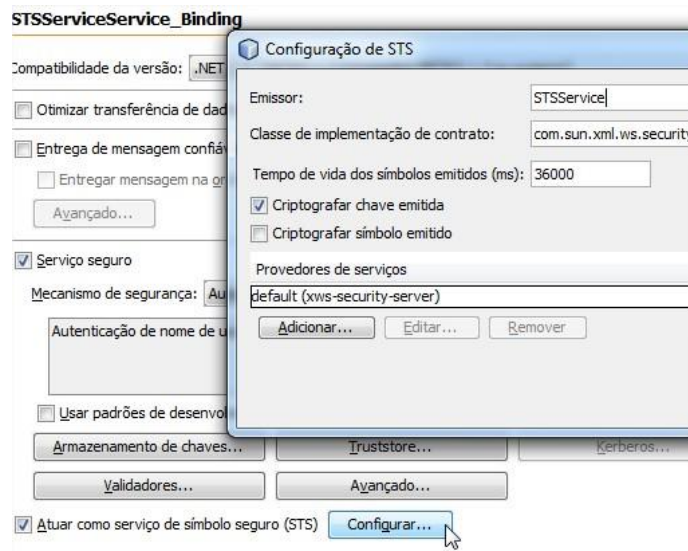
- Foi clicado com direito *STSServiceService[ISTSServiceService_Port]* e seleccionou-se em *Editar atributos do serviço Web* para configurar o STS:



- Foi seleccionada a opção *Compatibilidade de versão* para *NET. 3,5 / 1,3 Metro*;
- Foi seleccionado *Serviço Seguro*;
- Verificou-se se o mecanismo de segurança de *Autenticação nome de usuário com chave simétrica* estava seleccionado;
- Foi clicado no botão *Configurar*:



- Verificou-se se *Basic128 bit* estava selecionado. Selecionou OK.
- Verificou-se se ainda não estiver selecionada, selecione *Atuar como serviço de token de seguro (STS)*;
- Clicou-se no botão *Configurar*. No campo *Emissor*, digitou-se *STSService*. Clicou-se em *OK*:



- Clicou-se no botão *Armazenamento de chaves...*
- Clicou-se no botão *Carregar Alias*, selecionou *wssip*,
- Clicou-se em *OK*;
- Clicou-se com botão direito do mouse na guia *ProjectSTS*, foi selecionada *Propriedades*. Foi selecionada a categoria *Executar* e foi digitada o seguinte no campo URL relativo: */STSServiceService?wsdl*;
- Executou-se o projeto;
- A *WSDL* do STS apareceu no navegador com a URL: *http://localhost:8080/ProjectSTS/STSServiceService?wsdl*.