GLOBAL SCIENCE AND TECHNOLOGY (ISSN 1984 - 3801)

ARQUITETURA AAA EM SISTEMAS WEB BASEADOS EM REST

André Luiz Amorim Berenguel
Leonardo Ribeiro Queirós
Márcia Izabel Fugisawa Souza
Maria das Dores Rosa Alves

Resumo: Este artigo mostra como um serviço web baseado em REST pode utilizar um sistema de AAA de modo transparente para um cliente que seja desde um browser a um servidor de aplicações web. Para tal, definimos os componentes da arquitetura com duas formas de troca de mensagens: seqüência agente e seqüência push. Nos dias atuais, é cada vez maior a demanda de separação de funcionalidades de aplicações em serviços web. A proposta deste artigo se mostra interessante para este cenário.

Palavras-chave: serviço web, segurança, protocolo.

Abstract: This paper presents how a web service based on REST can use an AAA system in a clear way for a client who can be since a web browser up to an application server. For that, it was defined the architecture components with two ways of message exchanging: agent sequency and push sequency. Nowadays, it is bigger the demand for separation of funcionalities in web services applications. This article's proposal is relevant to this scenario.

Key-words: web service, security, protocol.

1. Introdução

A *World Wide Web*, ou simplesmente web, é o sistema de maior sucesso na internet. Seu fundamento principal está baseado na transferência de documentos de hipermídia interligados entre si por hiperlinks [1]. Esta propriedade de interligação também é chamada de **encadeamento**. Deste modo, a partir de um documento, é possível atingir vários outros documentos da web (popularmente, esta ação é conhecida como "navegar na web"). Outra propriedade que faz a web ser um grande sucesso é o **endereçamento**. A partir de um identificador universal conhecido como URI (*Uniform Resource Identifier*), é possível alcançar diretamente qualquer documento na web [2].

Para dar suporte à transferência destes documentos de hipermídia, surgiu o protocolo HTTP (*HyperText Transfer Protocol)* [3, 4]. Este protocolo de camada de aplicação fornece uma **interface uniforme** para se transferir os documentos na web. Quaisquer clientes e servidores que implementem o HTTP podem se comunicar e transferir documentos. Por padrão, o HTTP é um protocolo que utiliza o TCP/IP e possui a porta 80 como porta de conexão. Por definição, o HTTP é um protocolo que possui **falta de estado**, ou seja, a

Recebido em 29/04/2009 e Aprovado em 07/05/2009.

resposta de uma requisição feita a um servidor é independente das requisições anteriores. O servidor não guarda quaisquer informações sobre uma requisição realizada.

Essas quatro propriedades: encadeamento, endereçamento, interface uniforme e falta de estado, fizeram com que a web tivesse um projeto simples, fato principal de seu sucesso. Podemos, então, definir a web como sendo baseada em um protocolo sem estado, uniforme de transferência de documentos endereçáveis e interligados entre si.

Um conceito mais abstrato da web veio com a definição do REST (*Representational State Transfer* - Transferência de Estado Representacional). O termo surgiu na tese de doutorado de Roy Fielding, um dos principais autores da especificação do protocolo HTTP [5]. Resumidamente, sua principal característica está em fornecer uma interface uniforme para acesso a representações de recursos através de um endereçamento também uniforme. Recursos podem ser generalizados como qualquer coisa, como um copo ou um livro. Cada recurso deve ser endereçável por um identificador uniforme. Representações são maneiras de visualizar tais recursos de modo que suas representações possam ser transferidas como documentos de hipermídia. Por exemplo, a representação de um livro pode ser uma imagem com as páginas do livro, um arquivo texto ou ainda um *stream* de áudio com a narrativa do livro. A web pode ser vista como uma instanciação dos fundamentos REST, já que utiliza uma interface uniforme de transferência de documentos, o protocolo HTTP e o endereçamento uniforme, a URI.

Baseados na filosofia REST e na web, surgiram os serviços web baseados em REST. Estes serviços web se diferenciam dos serviços web baseados na arquitetura SOA (*Service Oriented Architecture*) pelo fato utilizarem o cabeçalho HTTP como envelope de meta-dados, ao contrário do SOA, que utiliza o HTTP apenas como um meio para transportar um envelope em seu próprio formato. A arquitetura SOA é voltada para o estilo RPC de sistemas distribuídos e não segue uma interface uniforme. Cada serviço web possui sua própria interface definida por um arquivo WSDL (*Web Service Definition Language*). Em serviços web baseados em REST, a interface uniforme dos métodos é dada pelos métodos definidos no HTTP, sendo os principais o *GET*, *POST*, *DELETE* e *PUT*. Com esses quatro métodos, é possível realizar qualquer operação sobre os recursos. Assim, serviços *web* baseados em REST são orientados a recursos.

Para simplificar a comunicação, neste artigo, nos referenciaremos a servidores de serviços *web* baseados em REST simplesmente como servidores de serviço *web*. Os servidores *web* que provêm conteúdo para clientes humanos (*browsers*) na forma de páginas HTML chamaremos de servidor de aplicação *web*. Quando um servidor centraliza todas as funcionalidades do sistema de modo que não necessita de um servidor de serviço *web*, chamá-lo-emos de servidor de conteúdo *web*.

Nesse artigo, iremos abordar um modelo para *web* onde servidores de aplicação web e servidores de serviço *web* podem interagir juntos para fornecer serviços a um browser que implementa a funcionalidade AJAX. Na seção 2, iremos analisar como se dá a interação entre um cliente e um servidor de conteúdo *web*. Na seção 3, mostraremos uma proposta de um modelo onde um servidor de conteúdo *web* é dividido em um servidor de aplicação *web* e um servidor de serviço *web*, tanto para um modelo simples sem utilização de recursos AAA (*Authentication*, *Authorization*, *Accounting*) quanto um mais complexo que utiliza. Para esse último caso, apresentaremos um modelo para o esquema seqüência agente e seqüência *push* do *framework* AAA [6]. A implementação e validação desses dois esquemas é mostrada na seção 4. Por fim,

concluímos o artigo na seção 5.

2. Cenário Atual

Na internet atual, o cenário mais simples de transferência de hipermídia ocorre com um cliente, normalmente um *browser*, realizando uma requisição para um servidor de conteúdo *web*. O cliente faz uma requisição HTTP por um determinado recurso e o servidor responde enviando a representação desse recurso. O recurso é endereçado por um URI. Nesse caso, o servidor não realiza um processo de AAA para verificar se o cliente pode ou não requisitar método sobre o recurso.

Um cenário mais complexo ocorre quando o cliente necessita de autenticação para realizar uma requisição. O cliente normalmente realiza isto de duas maneiras:

- O cliente envia o par usuário/senha para o servidor e recebe um *token* de sessão (chave de sessão) que identifica pelo qual é possível mapear o estado do cliente no lado do servidor. Esse *token* é setado no cliente como um *cookie* e em cada requisição que necessite AAA, esse *token* é enviado junto. O servidor pode, então, realizar o AAA descobrindo o estado do cliente através deste *token*.

- O cliente utiliza o mecanismo de autenticação padrão do HTTP [7], que pode ser o HTTP *Basic* ou HTTP *Digest*. Também há outros mecanismos, mas não são padrões do HTTP.

Em ambos os casos, o usuário envia uma credencial na forma de usuário/senha para se autenticar no servidor.

3. Cenários com servidor de serviço *web*

Ao contrário do caso mostrado na seção anterior, nesta seção vamos mostrar uma proposta onde parte das funcionalidades requeridas pelo cliente está num servidor de aplicação web e parte está num servidor de serviço web. Este cenário está se tornando cada vez mais comum hoje em dia, pois há uma grande necessidade das corporações separar as mais diversas funcionalidades em serviços web distintos. A seguir, analisaremos os modos de seqüência de mensagens nos quais os sistemas podem se relacionar com os serviços web para prover funcionalidades ao cliente.

3.1 Modos de seqüência de mensagens

Os cenários que analisaremos possuem três componentes: o cliente (*browser*), o servidor de aplicação *web* e o servidor de serviço *web*. A seqüência das mensagens pode se dar de duas formas: seqüência agente e seqüência *push*, as quais abordaremos a seguir.

3.1.1 Seqüência Agente

Nesse tipo de seqüência, como mostra a Figura 3.1, o servidor de aplicação *web* funciona como um agente do servidor de serviço *web*. O cliente envia uma requisição para o servidor de aplicação *web* que, por sua vez, baseado em suas políticas, permite ou não o acesso ao servidor de serviço *web*. O servidor de aplicação *web* ainda pode formatar os dados recebidos do servidor de serviço *web* para exibi-los de forma amigável ao cliente. Note ainda que o servidor de aplicação é quem realiza o processo de AAA. O servidor de serviço *web* simplesmente confia no servidor de aplicação e permite que todas as operações sejam realizadas.

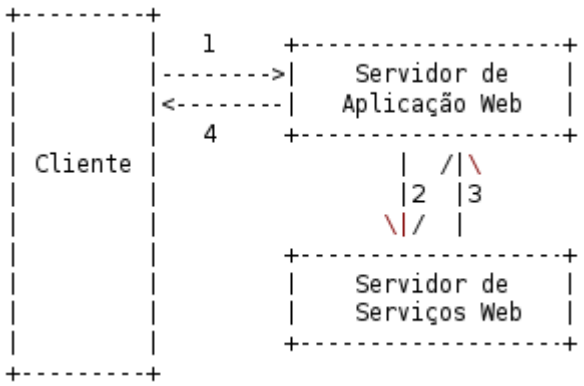

Figura 3.1: Seqüência Agente

3.1.2 Seqüência *Push*

Diferentemente do caso anterior, na seqüência de mensagem *push* o cliente interage tanto com o servidor de aplicação *web* quanto com o servidor de serviço *web* como mostra a Figura 3.2. Do ponto de vista de desempenho, esse caso é o ideal, já que não há triangulação na comunicação como a que era feita entre o cliente e o servidor de serviço *web* na seqüência agente. Contudo, dois problemas surgem: o cliente deve ser capaz de interpretar os dados do servidor *web* e, durante o processo de AAA, o servidor de serviço *web* tem, de alguma maneira, que saber que o cliente passou por tal processo. Veremos isto na seção 3.2.

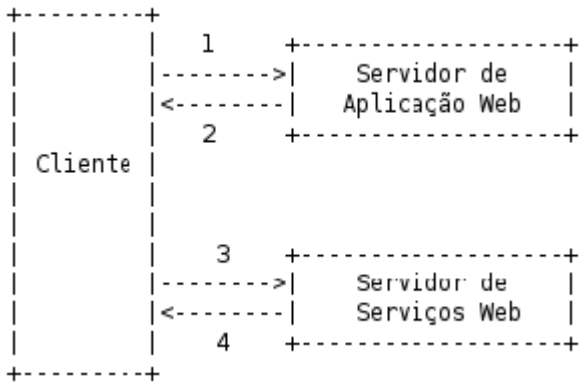

Figura 3.2: Seqüência *Push*

3.2 Solução genérica

Para satisfazer ambos os casos de seqüência de mensagens, iremos propor um protocolo de AAA, onde as três partes participam do processo. Para dar um entendimento mais geral, vamos exemplificar um caso de AAA com mensagens do tipo seqüência *push*, com uma pequena interação entre os servidores *web* como mostra a Figura 3.3. Depois, adaptaremos este caso para a seqüência agente.

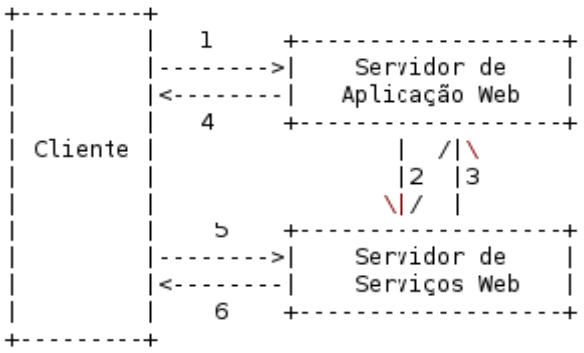

Figura 3.3: Processo de AAA

Na Figura 3.3, o cliente inicialmente envia suas credenciais para o servidor de aplicação de modo seguro (mensagem 1). O servidor então realiza a autenticação e, se esta for realizada com sucesso, o servidor de aplicação cria uma lista de privilégios que o cliente pode realizar no servidor de serviço *web* baseando-se no seu perfil para enviar uma mensagem de concessão de privilégios para o servidor de serviço *web* (mensagem 2). Nessa mensagem, ainda são enviadas informações do usuário para que servidor de serviço *web* possa identificá-lo unicamente. Conflitos entre usuário com mesmo identificador de diferentes aplicações são desfeitos pelo fato do servidor de serviço *web* relacioná-lo com a aplicação. Note que o servidor de serviço *web* somente aceita concessão de privilégios de alguns servidores de aplicação. É necessário que o servidor de aplicação se autentique de alguma forma no servidor de serviço *web*.

O servidor de serviço *web* então guarda as informações dos privilégios associados ao usuário da aplicação e mapeia um *token* de sessão criado aleatoriamente com estas informações. Esse *token* é retornado como resposta ao servidor de aplicação *web* (mensagem 3). Esse servidor simplesmente repassa esse *token* para o cliente (mensagem 4).

A partir deste momento, o cliente tem condições de enviar uma mensagem de *HELLO* para servidor de serviço *web*. Para isto, ele monta uma mensagem contendo autenticação HTTP [7]. Como usuário, ele utiliza seu próprio nome de usuário associado com um id de aplicação fornecido pela própria aplicação, e como senha utiliza o *token* recebido do servidor de aplicação web. Se a resposta do *HELLO* for sucesso, o cliente tem a certeza de que pode realizar as operações baseadas no seu perfil. Toda próxima mensagem que o cliente enviar para o servidor de serviço *web* deverá estar autenticada da mesma maneira que foi autenticada no *HELLO*.

Podemos generalizar este caso para um modelo de seqüência agente. Do ponto de vista do cliente, ele estaria mais simplificado, pois tanto faz para ele se há ou não um servidor de serviço *web* por trás do servidor de aplicação. Ele se comunica apenas com o servidor de aplicação, que teria que fazer o papel do cliente do caso anterior. É possível obter este modelo se colocarmos as funcionalidades do cliente dentro do bloco do servidor de aplicação e o cliente de verdade se comunicando com o servidor de aplicação.

4. Implementação e Validação

Implementamos um caso desta proposta no laboratório de "Organização e Tratamento da Informação Eletrônica" do CNPTIA, para o projeto Agência de Informação da Embrapa, onde o servidor de aplicação *web* é um sistema Gestor de Conteúdo de Informações que contém uma janela de autenticação contendo campos para usuário e senha e o servidor de serviço *web* é um *web-service* implementado em Java com base no *framework* Restlet 1.1RC2 [8]. Este *web-service* é utilizado para controlar um vocabulário de termos técnicos do setor de agricultura.

Para validação, realizamos um teste com um cliente sendo um *browser* Mozilla Firefox 3.03, um. Baseando-nos na Figura 3.3, tivemos as seguintes mensagens trocadas, todas sobre HTTP.

- **Mensagem1:** Requisição POST sobre HTTP seguro contendo os parâmetros usuario=fulano e senha=123mudar.

- **Mensagem2:** Requisição POST sobre HTTP seguro com dupla autenticação entre o servidor de aplicação *web* e o servidor de serviço *web*, através de certificados assinados. Tal requisição conteve um XML na seguinte forma:

```
<? xml v e r s i on ="1.0" e n c o d i n g ="UTF -8"?>
  <concession>
     <username>fulano</username>
      <privileges>
        <privilege>INSERT_TERM</privilege>
        <privilege>CHANGE_STATUS_TERM</privilege>
      </privileges>
  </concession>
```

- **Mensagem 3:** Resposta à mensagem 2 contendo um *token* de sessão gerado aleatoriamente no corpo da mensagem **HTTP** da seguinte forma: rcoUl+olVvpOfuR0XzJdkfmE7bU=

- **Mensagem 4:** Resposta à mensagem 1 contendo o mesmo conteúdo da mensagem 3.

- **Mensagem 5:** Requisição *GET* no recurso de *HELLO* do servidor de serviço *web*. Parâmetros de autenticação HTTP são passados como o usuário sendo "fulano/id_aplicacao" e senha sendo "rcoUl+olVvpOfuR0XzJdkfmE7bU=". Utilizamos HTTP Digest sobre HTTP não seguro.

- **Mensagem 6:** Resposta à mensagem 5 sem conteúdo, apenas com status HTTP de sucesso 200.

Esse processo todo significa que o cliente poderá apenas realizar operações de inserir termo e alterar o status do termo no servidor de serviço *web*. Do ponto de vista do cliente, ele realiza duas requisições em seqüência. Para que o usuário não precise interferir, implementamos um *script* baseado em AJAX para pegar a resposta da primeira requisição (mensagem 4) e montar a segunda requisição (mensagem 5). Ao realizar o processo de *HELLO* no servidor de serviço *web*, o *browser* automaticamente guarda em cache o par usuário e senha da mensagem 5. Deste modo o *browser*, realizou um processo de AAA no servidor de aplicação e de serviço *web* de modo transparente.

5. Conclusão

Nesse trabalho, mostramos como é possível separar funcionalidades de um servidor de conteúdo *web* em um servidor de serviço *web* sem ter a necessidade de replicar a funcionalidade de autenticação. Esse modelo se mostrou interessante e viável em nossas implementações.

Do ponto de vista de segurança, esta implementação se mostra adequada já que há criptografia nas mensagens com informações cruciais (mensagens 1, 3 e 4) e autenticação mútua no processo de concessão de privilégios (mensagens 2 e 3). Na mensagem 5, é possível utilizar tanto HTTP *Basic* sobre HTTP seguro quanto HTTP *Digest* para obter segurança. Se um cliente malicioso resolver realizar operações que não correspondam a seus privilégios, o *web-service* responde com uma mensagem de status 403 (*Forbidden*).

6. Referências

BERNERS-LEE, T.; CAILLIAU, R. **Worldwideweb**: Proposal for a hypertext project. 2000. Disponível em: <http://www.w3.org/Proposal>. Acesso em maio de 2001.

BERNERS-LEE, T.; FIELDING, R.; FRYSTYK, H. **Hypertext Transfer Protocol** – HTTP/1.0. IETF, maio 1996. RFC 1945 (Informational). (Request for Comments, 1945). Disponível em: <http://www.ietf.org/rfc/rfc1945.txt>. Acesso em: Maio de 1996.

FIELDING, R. et al. **Hypertext Transfer Protocol** – HTTP/1.1. IETF, jun. 1999. RFC 2616 (Draft Standard). (Request for Comments, 2616). Updated by RFC 2817. Disponível em: <http://www.ietf.org/rfc/rfc2616.txt>. Acesso em: Junho de 1999.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. Tese (Doutorado) — University of California, Irvine, 2000. Disponível em: <http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm>.

FRANKS, J. et al. HTTP **Authentication: Basic and Digest Access Authentication**. IETF, jun. 1999. RFC 2617 (Draft Standard). (Request for Comments, 2617). Disponível em: <http://www.ietf.org/rfc/rfc2617.txt>. Acesso em: Junho de 1999.

LOUVEL, J. BOILEAU, T. Restlet. **Noelios Technologies**, set. 2008. Disponível em: <http://www.restlet.org>. Acesso em: Setembro de 2008.

MEALLING, M.; DENENBERG, R. **Report from the Joint W3C/IETF URI Planning Interest Group**: Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations. IETF, ago. 2002. RFC

3305 (Informational). (Request for Comments, 3305). Disponível em: <http://www.ietf.org/rfc/rfc3305.txt>. Acesso em: Agosto de 2002.

VOLLBRECHT, J. et al. **AAA Authorization Framework**. IETF, ago. 2000. RFC 2904 (Informational). (Request for Comments, 2904). Disponível em: <http://www.ietf.org/rfc/rfc2904.txt>. Acesso em: Agosto de 2000.