# *DSOA e Padrões de Segurança aplicados em Web Services*

**Fábio Sarturi Prass**
Faculdade Antonio Meneghetti (AMF)
Pontifícia Universidade Católica do Rio Grande do Sul (PUC-RS)
fabio@fp2.com.br

**Resumo**: As organizações enfrentam uma série de problemas para atender às exigências previstas pelas normas e modelos de segurança de software, além do aumento contínuo das exigências relacionadas à segurança em sistemas. Uma série de normas e modelos de segurança estão disponíveis na literatura a fim de conduzirem o desenvolvimento de software seguro. Para resolver esse problema tem-se os padrões que são amplamente utilizados em Engenharia de Software onde eles têm sido bem sucedidas na melhoria da análise e projeto por encapsular a experiência de muitos designers. Padrões de segurança são um desenvolvimento recente, como forma de encapsular o conhecimento acumulado sobre o *design* de sistemas de segurança. Apresenta-se aqui dois padrões para a *Web Services*: *Authentication* and *Authorization* com utilização de Aspectos.
**Palavras-chave**: padrões; segurança; *Web Services*; aspectos.

**Abstract**: Organizations face a number of problem to meet the requirements of the standards and models of security software, in increase to continued growth in requirements related to security systems. A series of standards and security models are available in the literature to lead the development of secure software. To resolve this problem has been that then Patterns are widely used in Software Engineering where they have been successfu l in improving analysis and design by encapsulating the experience of many designers. Security patterns are a recent development as a way to encapsulate the accumulated knowledge about secure systems design. We present here two patterns for web services: authentication and authorization aspects of using.
**Keywords**: patterns; security; web services; aspects.



## 1 Introdução

Em sistemas atuais com inúmeros recursos de comunicação, considerações de prosperidade e segurança são de maior interesse. Portanto, uma boa quantidade de experiência em segurança adicional é necessário para atender os requisitos não-funcionais de segurança. Uma abordagem comum para superar as lacunas de conhecimento entre os desenvolvedores é a utilização de padrões (por exemplo, padrões de projetos, padrões de segurança, padrões de sinal, padrões de especificação, etc.) Na segurança de domínio, é um desafio para capturar e transmitir informações a fim de facilitar engenharia de segurança, que pode muitas vezes ser um objetivo abstrato (PRASAD, RAMAKRISHNA e SHRAVANI, 2010).

As necessidades de segurança de um sistema dependem fortemente do

ambiente em que o sistema for implantado. Além disso, o comportamento da informação com as restrições de segurança estão incluídos em um modelo padrão de segurança. O desenvolvedor pode usar esta informação para verificar se um projeto específico ou a implementação do padrão é consistente com as propriedades essenciais de segurança.

Esta tendência ocorre com uma maior frequência em tecnologias baseadas na *Web*, dando origem a uma demanda de tecnologias que buscam segurança em aplicações *Web* e *Web Services* (CURPHEY & FOUNDSTONE, 2006). Inserida nesse contexto, a palavra-chave no desenvolvimento de software é segurança. Para implementar segurança em *Web Services* são necessárias modificações. Isto abrange desde questões culturais até a capacitação dos envolvidos em temas associados a segurança. Todo esse processo de reeducação leva tempo e não acontece simplesmente ao escrever uma política ou definir alguns *checklists* (CURPHEY & FOUNDSTONE, 2006). Em decorrência dessa situação, Schumacher (2006) propõe uma coleção de padrões de segurança que podem ser adaptados a essas dificuldades. Sendo assim, tem-se a abordagem de padrão, que se baseia num conjunto de ativos em camadas que pode ser explorada por qualquer metodologia de desenvolvimento existente.

De acordo com Schumacher (2006), padrões de segurança são soluções reutilizáveis aos problemas de segurança. Embora muitos padrões de segurança e técnicas para usá-los têm sido propostos, é complexo adaptá-los e integrá-los em cada fase do desenvolvimento de software.

Atualmente, são desenvolvidos e melhorados cada vez mais os padrões de segurança, incluindo a estes, padrões para *Web Services*, os quais são complexos e detalhados *e* tem evoluído para integrar organizações que usam tecnologias diferentes executando em sistemas heterogêneos e *frameworks*. É um componente de lógica de negócios projetado para ser acessado através de uma rede utilizando protocolos padrão. Logo não é uma tarefa simples para os desenvolvedores e usuários compreenderem seus pontos chave. Assim, um fornecedor de produtos pode usar normas para orientar o desenvolvimento de um produto e, ao expressar normas e padrões, pode compará-los e compreendê-los cada um dos seus recursos com mais precisão (ENDREI et al., 2004). Nesse sentido, para projetar, desenvolver e implantar *Web Services* seguros, arquitetos e desenvolvedores precisam compreender novas tecnologias e analisar as ameaças associadas à exposição de funcionalidade em redes inseguras (PRASAD, RAMAKRISHNA e SHRAVANI, 2010).

Sem uma definição clara de como os *Web Services* podem gerenciar e estabelecer comunicações seguras em relações de confiança com outros parceiros ou tecnologias, seria difícil realizar qualquer tipo de interação. Considerando que um sistema tradicional possui os dados necessários para conhecer a *priori* cada usuário que o utiliza e o que cada um pode fazer. Assim, um usuário também tem o conhecimento prévio de quais sistemas vai acessar, já possuindo os dados necessários para cada autenticação, e com autorização prévia para realizar um conjunto de operações em cada um deles (SCHUMACHER, 2006).

Segundo Monday (2003), para resguardar esses recursos, as organizações precisam definir políticas de segurança, que são linhas de orientação de alto nível que especificam em qual estado o sistema é considerado seguro. Essas políticas precisam ser aplicadas por meio de mecanismos de segurança e, para implementar esses mecanismos, pode-se aplicar alguns padrões, dentre os quais destaca-se o uso da *Authenticator* e

*Authorization* (FERNANDEZ & DELESSY, 2006).

Com o intuito de propor uma melhor implementação desses mecanismos de segurança, optou-se pelo uso da orientação a aspectos (AOP - *Aspect-Oriented Programming*) para tratar alguns requisitos como preocupações transversais, ou seja, aquelas que estão espalhadas por toda a aplicação (JENSEN et al., 2007). Sendo assim, a orientação a aspectos proporciona um nível maior de abstração, possibilitando tratar os requisitos de segurança separadamente dos requisitos funcionais. Nessa perspectiva, propõe-se, neste trabalho, a modelagem de requisitos de segurança, descritos como padrões de segurança, como aspectos de software. Através desta técnica, busca-se separar os requisitos de segurança dos requisitos funcionais do software e aplicá-los na implementação de *Web Service*, obtendo um nível maior de segurança.

Para atingir o objetivo proposto, o texto é dividido da seguinte forma: a seção 2 apresenta uma breve introdução a segurança em *Web Services*. Na seção 3, busca-se fazer uma introdução descritiva sobre o desenvolvimento de software orientado a aspectos. A seção 4 apresenta uma visão geral dos padrões de segurança associados ao desenvolvimento de *Web Services* e define os padrões a serem considerados no modelo proposto e sugere como representar padrões de segurança usando aspectos. A seção 5 descreve um estudo de caso com a aplicabilidade dos padrões analisados. Na seção 6 é realizada uma comparação com trabalhos relacionados e, finalmente na seção 7, são apresentadas as considerações finais, incluindo as contribuições e trabalhos futuros.

## 2 Segurança em *Web Services*

Segurança em *Web Services,* de acordo com (ENDREI et al., 2004), engloba uma série de exigências que podem ser descritas ao longo das dimensões de segurança mais conhecidas, ou seja, integridade, pela qual a mensagem deve permanecer inalterada durante a transmissão; a confidencialidade, pela qual o conteúdo de uma mensagem não pode ser visualizado quando em trânsito, exceto pelos serviços autorizados; disponibilidade, pela qual a mensagem é imediatamente entregue ao destinatário, garantindo, assim, que os usuários legítimos recebam os serviços a que têm direito. Além disso, cada *Web Service* deve proteger seus próprios recursos contra o acesso não autorizado. Esse, por sua vez, exige meios adequados para identificação, pelo qual o destinatário de uma mensagem deve ser capaz de identificar o remetente; autenticação, através do qual o destinatário de uma mensagem precisa verificar a identidade do remetente; autorização, pelo qual o destinatário de uma mensagem possui necessidade de aplicar políticas de controle de acesso para determinar se o remetente tem direito de utilizar os recursos necessários.



Basicamente, o *Web Service* faz com que os recursos da aplicação do software fiquem disponíveis sobre a rede de uma forma normalizada. Diferentes tecnologias fazem o mesmo processo, como por exemplo, os *browsers* da *Internet* acessem as páginas *Web* disponíveis utilizando por norma as tecnologias da *Internet*, HTTP e HTML. Entretanto, estas tecnologias não são bem sucedidas na comunicação e integração de aplicações. Tem-se uma motivação sobre a tecnologia *Web Service*, pois possibilita que distintas aplicações comuniquem entre si e utilizem recursos diferentes.

Visando fornecer facilidades de acesso e integração, a arquitetura de *Web Services* pode ser vulnerável, sendo necessário que a segurança seja considerada fundamental durante o projeto

de um sistema projetado nesta tecnologia. O desafio maior é, então, manter a eficiência das funcionalidades e ainda proporcionar um ambiente seguro (PRASAD, RAMAKRISHNA & SHRAVANI, 2010).

Uma das maiores vantagens de *Web Services* é compartilhar e redistribuir informações, concebido para ser completamente independente da tecnologia usada para construir aplicativos. Dessa forma, está acima de plataformas, bancos de dados e linguagens de programação, eliminando as limitações existentes entre aplicativos. Portanto, os clientes podem ser criados em qualquer plataforma e em qualquer linguagem de programação, independente de tecnologia e de linguagem nas quais foram implementados os aplicativos nos servidores (ENDREI et al. 2004).

Nesse sentido, *Web Services* expõem suas funcionalidades através de uma arquitetura orientada a serviços (SOA), que é mais aberta em virtude de sua característica distribuída e de sua natureza heterogênea quanto a plataformas de execução. Assim, existem alguns desafios relacionados à segurança em *Web Services* que devem ser explanados, como, quando um *Web Service* se conecta a um parceiro de negócios, existe a confiança que este parceiro faça a autenticação corretamente ou ateste a identidade dos usuários em sua extremidade da transação. Isso significa que um invasor que tenha obtido acesso a um fornecedor, por exemplo, poderia utilizar essa autenticação imprópria para invadir sistemas de clientes do fornecedor. Para evitar tais invasões, precisa-se vislumbrar, além das medidas de segurança em nível de aplicativo, regras de controles de acesso, autenticação e recursos de autorização, dentre outros que possam acompanhar as consultas e respostas entre as trocas de mensagens entre um cliente e seu fornecedor (KICZALES & MEZINI, 2005).

## 3 Desenvolvimento de Software Orientado a Aspectos

O Desenvolvimento de Software Orientado a Aspectos (DSOA) ou *Aspect-Oriented Software Development* (AOSD) é uma proposta de aumentar a modularização e a composição de características ou preocupações transversais em um software, representando-as e tornando-as compreensíveis, uma vez que as características transversais são partes do sistema ou dos requisitos que estão entrelaçados ou entrepostos nos demais elementos de um sistema (STEEL, NAGAPPAN & LAI, 2005).

Destaca-se, então, que a Programação Orientada a Aspectos (POA) surgiu com a finalidade de separar o código que implementa a funcionalidade da aplicação, da implementação do aspecto ou do requisito que é considerado transversal. A POA propõe esse conceito de aspectos para representar as características transversais (*crosscutting concerns*), ou seja, de uma forma geral, características transversais são partes de um sistema que estão fortemente relacionadas, entrelaçadas ou sobrepostas, influenciando ou restringindo umas as outras, tornando o sistema complexo e difícil de analisar. Assim, no desenvolvimento de um sistema baseado em aspectos, observa-se a distinção entre dois tipos de códigos: um código base que diz respeito ao propósito básico da aplicação; e um código que está espalhado dentro do código base causando o entrelaçamento e dificultando a compreensão do software (KICZALES & MEZINI, 2005).

Com o DSOA obtém-se um sistema mais modularizado e com ênfase na reusabilidade e evolução do software (MEDEIROS, 2008). Os conceitos comumente definidos são: a) *aspects*, b) *joinpoints*, c) *pointcuts*, d) *advices* e e) *Intertype Declarations*, os quais são

respectivamente definidos como:

*Aspects* são unidades modulares, designadas a implementar e encapsular características transversais por meio de instruções sobre onde, quando e como eles são invocados dentro de um sistema (KHAARI & RAMSIN, 2008).

*Joinpoints* são locais bem definidos na estrutura ou fluxo de execução de um código onde comportamentos extras podem ser adicionados, os quais são afetados e invocados pelos aspectos (KHAARI & RAMSIN, 2008).

*Pointcuts* indicam os elementos afetados por uma determinada característica transversal; são mecanismos que encapsulam os *joinpoints*. Isso é uma importante característica da POA, por possibilitar um mecanismo de inserção, em outras palavras, um caminho para relacionar algo importante em muitos lugares de um programa com uma simples declaração (KHAARI & RAMSIN, 2008).

*Advices* possuem duas partes, sendo que a primeira é o *pointcut*, que determina as regras de captura dos *joinpoints*; e, a segunda, é o código que será executado quando ocorrer o ponto de junção definido pela primeira parte. Cada *advice* possui um tipo (*after*, *before* e *around*) que descreve quando o comportamento deve ser inserido nos *joinpoints* (KHAARI & RAMSIN, 2008).

*Intertype Declarations* são mecanismos utilizados para adicionar novos tipos de dados ou de elementos ao sistema. Fornece a possibilidade de adição de novos elementos sem que seja preciso fazer alterações diretas no código (KHAARI & RAMSIN, 2008).

Existem diversos mecanismos que fazem parte da estrutura da orientação a aspectos, contudo, neste trabalho, estão sendo abordados apenas os principais conceitos.

## 4 Representando Padrões de Segurança em Web Services usando DSOA

As falhas de segurança são, muitas vezes, inerentes às falhas no sistema. Infelizmente, as falhas não podem ser evitadas completamente em sistemas de software complexos. Assim, Kiczales e Mezini (2005) apontam que ainda é importante ter planos de contingência para falhas e para garantir que o sistema de segurança não seja comprometido pelo comportamento excepcional. Alguns sistemas executam operações inseguras durante os modos de falha, a fim de fornecer determinadas funcionalidades ou para manter a compatibilidade com os antigos padrões. Nesse contexto, um invasor pode encontrar uma maneira de acionar uma falha de um sistema inseguro e tirar proveito de seu comportamento.

Dessa forma, padrões de segurança são propostos como um meio de suprir essa lacuna, pois são destinados à captura de experiências em segurança sob a forma de soluções reutilizáveis para problemas recorrentes. Acima de tudo, os padrões de segurança são desenvolvidos para serem construtivos e aderentes ao extenso conhecimento acumulado sobre segurança, fornecendo orientações para a construção e validação de sistemas seguros.



A existência de padrões de segurança não é suficiente; faz-se necessário, também, metodologias que permitam aos usuários aplicar esses padrões em situações práticas. A maioria dos projetistas são especializados em desenvolvimento de software e proficientes em algumas linguagens de programação, mas normalmente não possuem conhecimento profundo na área de segurança.

A fim de manter um controle de segurança no contexto de *Web Services*, neste artigo é proposto o uso de padrões de segurança, descritos por Schumacher et al. (2006), dentre os quais se podem citar

a*uthenticator* e *authorization,* conceituados da seguinte forma:

*Authenticator* é o processo de identificar individualmente os clientes de suas aplicações e serviços. Estes podem ser usuários finais, outros serviços, processos ou computadores. Em termos de segurança, os clientes autenticados são chamados de diretores. É ainda considerado um recurso de segurança primária, pois os mecanismos usados para autenticação frequentemente influenciam os mecanismos utilizados para permitir outros recursos de segurança, tais como a confidencialidade dos dados e autenticação da origem dos dados.

*Authorization* é o processo que gerencia os recursos e as operações que o cliente autenticado tem acesso permitido. Os recursos incluem arquivos, bases de dados, tabelas, linhas e assim por diante, com nível de sistema de recursos, tais como chaves de registro e dados de configuração. As operações incluem a realização de transações como a compra de um produto, transferência de dinheiro de uma conta para outra, ou o aumento de notação de crédito do cliente.

Esses padrões podem ser aplicados em inúmeras circunstâncias considerando a *Web Services*. O método mais simples é o de autenticação local, conforme pode ser observado na

Figura 1.

Nesse modelo, o nome de usuário e senha para cada usuário autenticável é armazenado localmente no sistema (servidor). Os usuários enviam seus nomes de usuário e senhas em texto puro para o sistema (1) que, por sua vez, compara sua informação de autenticação com o gravado em um banco de dados. Se o nome de usuário e a senha fornecida são encontrados e validados (2), o usuário é considerado autenticado. Este é basicamente o modelo usado para autenticação de *logins* em sistemas multiusuários tradicionais e tem sido replicado inúmeras vezes dentro de pacotes de diversos aplicativos.

Cada regra determina uma lista de métodos de autenticação. Cada método define os requisitos acerca de como as identidades são conferidas nas comunicações às quais a regra agregada se aplica. Os dois computadores necessitam ter, pelo menos, um método de autenticação comum ou a comunicação não terá êxito. A criação de múltiplos métodos de autenticação aumenta a possibilidade de ser descoberto um método comum entre os dois computadores.

Apenas um método de autenticação pode ser empregado entre dois computadores, independentemente do número de métodos configurados. Se existir várias regras que se apliquem ao mesmo par de computadores, é necessário configurar a lista de métodos de autenticação nessas regras para consentir que os computadores empreguem o mesmo método.



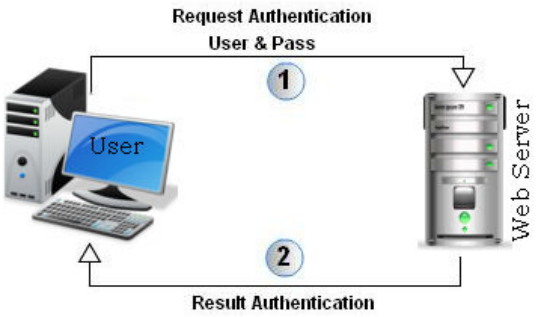


Figura 1 - Método tradicional de autenticação

Este método possui alguns detalhes que precisam ser analisados com uma maior atenção e são evidenciados abaixo:
Em muitos casos, as senhas dos usuários são armazenadas em texto simples no servidor. Quem possui acesso ao servidor de banco de dados, tem acesso a informações suficientes para representar qualquer usuário autenticável.

Nos casos em que as senhas dos usuários são armazenadas no formato criptografado no servidor, senhas de texto simples ainda são enviadas através de uma

rede insegura, possivelmente a partir do cliente para o servidor. Qualquer pessoa com acesso à rede de intervenção pode ser capaz de investigar os pares de usuário e senha e reproduzi-las para forjar a autenticação no sistema.

A autenticação não é reutilizável, ou seja, os usuários devem se autenticar em separado para cada sistema ou aplicativo que deseja acessar. Como resultado, devem digitar repetidamente suas senhas e tendem a escolher, desta forma, senhas menos segura.

Não é previsto no modelo autenticar a comunicação entre o servidor e cliente. Um sistema que representa o sistema de servidor não pode ser distinguido pelo cliente a partir do servidor real, abrindo a possibilidade de, por exemplo, um software malicioso coletar as informações (usuário e senha) e depois usá-las na autenticação de um servidor real.

Nesse contexto, a implementação de requisitos de segurança em sistemas com *Web Services* passa por alguns desafios de identificação de padrões para esse domínio. A estrutura de *Web Services* necessita de adaptações para a adequação da engenharia de segurança e dos padrões existentes. Um dos caminhos para as aplicações dos padrões de segurança, propostos por Schumacher et al. (2006), é satisfazer os requisitos de segurança desde o projeto, integrando-os nas especificações dos requisitos funcionais e não funcionais. A fim de maximizar a inteligibilidade, faz-se uso de modelos e de notações como a *Unified Modeling Language* (UML) para representar essas informações estruturais e comportamentais (O'NEILL et al., 2006).

Outra forma de representar esses requisitos é com a técnica de aspectos implementada no *Web Service*, conforme a

Figura 2, em que o aspecto é representado no servidor (2), específico para fazer as validações de segurança (3), utilizando os padrões anteriormente implementados e a orientação a aspectos. Neste tipo de cenário não há a ligação direta entre o usuário e o servidor de autenticação, o que delimita um nível de acesso mais restrito que o exemplificado no modelo anterior, ou seja, a autenticação nos serviços restritos é feita pelo *Web Service* através do aspecto que pode estar em um servidor exclusivo ou ser apenas uma aplicação.

Com a orientação a aspectos inserindo padrões e requisitos de segurança nas operações realizadas pelo servidor de autenticação, obtêm-se o benefício de deixar para o *Web Service* somente a função para a qual foi criado, isto é, a prestação de serviço entre aplicações, distribuindo informações; deixando, desse modo, para o aspecto toda a responsabilidade de autenticação, criptografia e controle de acesso a partir de um módulo de segurança que provê compartilhamento desses requisitos de segurança. Considerando que cada serviço prestado pelo *Web Service* é uma operação, o mesmo pode operar com requisitos transversais sem alterar o serviço em si. Quando é solicitada uma autenticação, o *Web Service* invoca o aspecto para fazer a validação da operação.

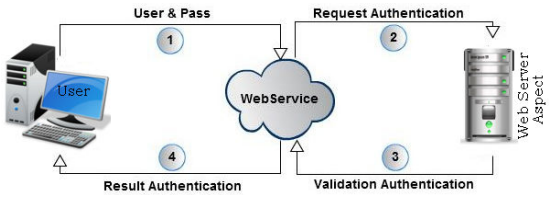


Figura 2 – Método proposto de *Authenticator* utilizando Aspectos

Ainda assim, investiga-se como a verificação das propriedades de requisitos pode ser ativada pela adição de padrões e requisitos de segurança, buscando torná-los reutilizáveis para que, quando alterados, não causem impacto negativo ao sistema, pois, dessa forma, serão

incrementados novos requisitos apenas no módulo principal responsável pela segurança. Outros detalhes sobre o modelo proposto pode ser observado no estudo de caso onde são representados os requisitos de segurança de uma operação bancária, aplicados ao uso de *Web Service* para distribuição de aplicações.

Quando, utilizando mecanismos de segurança, é considerada a autenticação, deve-se também considerar a autorização, uma vez que estes processos caminham lado a lado, e uma política de autorização significativa requer usuários autenticados. Seguindo essa perspectiva, na

Figura 3 são representados os elementos propostos pelo padrão *Authorization*, descritos a partir de um diagrama de classe. Neste modelo, percebe-se que a classe *Subject* descreve uma entidade que se empenha em acessar um recurso de alguma forma, considerando que a classe *Protection Object* representa um recurso a ser protegido. A associação entre o sujeito e o objeto define uma autenticação e a associação da classe *Right* descreve o tipo de acesso que o sujeito está autorizado a executar sobre o objeto correspondente. Através desse, uma classe pode verificar os direitos/permissões que um sujeito tem em um determinado objeto.

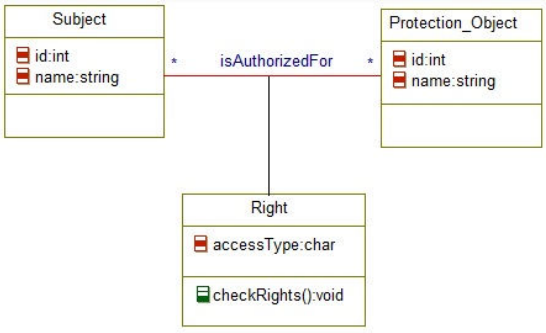


Figura 3 - Modelo de classe para *Authorization* (adaptado de Schumacher et al., 2006).

O modelo de classes do padrão *Authorization*, proposto por Schumacher et al. (2006), também pode ser representado com o uso da orientação a aspectos. Assim, analisando-se a

Figura 4, pode-se observar no diagrama de classe o uso de um estereótipo responsável por uma classe abstrata; esta, por sua vez, representa o aspecto em que são implementados os requisitos de segurança responsáveis pela autorização.

Estes requisitos de segurança determinam os direitos de acesso para cada usuário. A funcionalidade fornecida pelos dois modelos é a mesma, a diferença do uso de aspectos é que esse controle é feito por uma técnica específica e com recursos compartilhados. Baseada no reuso de requisitos de segurança e de partes do programa, facilita, assim, a leitura do código fonte, a fácil manutenção e é realizada somente nos módulos responsáveis pela segurança das aplicações, por parte dos desenvolvedores.

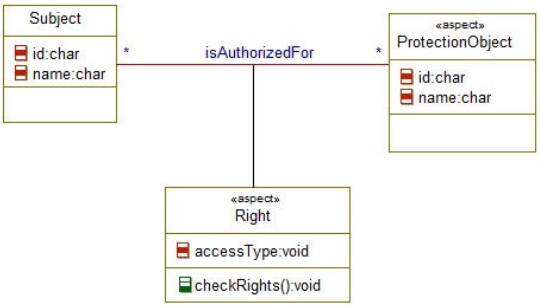


Figura 4 – Modelo de classe para o Padrão *Authorization* utilizando aspectos

## 5 Estudo de Caso

O estudo de caso elaborado neste trabalho se refere a um sistema bancário, representando, especificamente, as operações bancárias mais comumente utilizadas pelos clientes.

O diagrama de classe, elaborado neste estudo, pode ser visualizado na

Figura 5, a qual demonstra como o sistema foi projetado, ou seja, por meio de uma arquitetura em camadas de aplicações (Gerenciador Transações). Sendo estas aplicações coordenadas a partir de um

sistema de *Web Services* que faz a interação entre o Banco, suas bases de dados e seus clientes.

Nesse caso, o *Web Service* tem a função de invocar o gerenciador da camada de aplicação para executar a autenticação do cliente no sistema bancário. Da mesma forma, o *Web Service* invoca os módulos de segurança que, no modelo, estão representados usando aspectos.

No diagrama de classes, a orientação a aspectos foi representada utilizando-se o estereótipo <<*Aspecto*>>; devido à complexidade desse modelo utiliza-se o recurso dos pontos de corte e de junção que são parte da estrutura do aspecto para melhor representarmos o estudo de caso. No modelo, considera-se o aspecto como uma dependência do *Web Service*. Já o estereótipo <<*Aspecto*>> como classe abstrata é implementada para garantir a execução desse aspecto, caso alguma das operações não passe pelo *Web Service* devido a falhas ou, até mesmo, em caso de cancelamento ou fim de alguma operação pelo cliente.

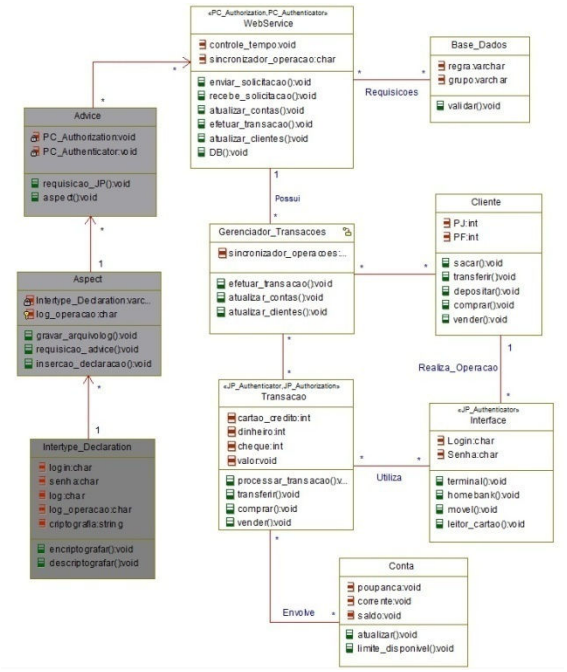


Figura 5 - Diagrama de classes do Sistema Bancário Orientado a aspectos

Nesse contexto, a arquitetura representada pelo aspecto é responsável pelo controle de segurança das operações coordenadas pelo banco, que, no estudo de caso, compartilham um mesmo módulo de segurança acessado pelos módulos de aspectos (representados pela cor cinza). Este módulo comporta, assim, as funções de autenticação, controle de acesso, controle de operações e outros requisitos que podem ser implementados posteriormente ou em tempo de execução.

O ponto de junção (*JoinPoint*) é responsável pelas requisições que são realizadas no ponto de corte (*PointCut*). No modelo proposto, têm-se apenas um *pointcut* inserindo os requisitos no *Web Service*. Mas, em um sistema maior, se faz necessário a inserção de mais *pointcuts*, de acordo com cada necessidade de abrangência desses requisitos o aspecto vai sendo implementado.

Um ponto de junção pode fazer requisição em mais de um ponto de corte, principalmente se tipos diferentes de requisitos forem implementados pelo mesmo aspecto, tem-se, então, um efeito cascata, onde os pontos de corte e de junção distribuem essas requisições coordenadas pelo mesmo aspecto. Neste caso, para evitar inconsistência e satisfazer os requisitos, utilizam-se mais pontos de corte e de junção. Esse é considerado um dos benefícios da orientação a aspectos, que possibilitou a representação e a inserção dos requisitos de segurança no modelo em questão.

As operações bancárias realizadas pelos clientes exigem autenticação, muitas vezes para cada operação exclusivamente, tais como saques, transferências dentre outras. Sendo assim, estas operações são executadas na camada de interface com o cliente (terminal bancário, acesso *web*, acesso móvel e cartões de crédito/débito) e executadas pela camada de aplicação (Gerenciador Transações). O que diferencia é a forma com que o sistema

autentica e criptografa as informações de usuário e senha, requisitando ao *Web Service* a autenticação no banco de dados. Nesse sentido, com a técnica de aspectos sendo aplicada e invocando os módulos de segurança têm-se uma melhor aplicação dos requisitos de segurança no *Web Service* e nas operações.

A classe "Cliente" possui os atributos referentes ao tipo de pessoa e ações que são executadas. A classe "Interface" comporta o primeiro requisito de segurança (login/senha) e seus atributos como formas de acesso. A classe "Transação" trata dos tipos de operações executadas. A classe "Gerenciador Transações" sincroniza as operações com as transações dos clientes e suas permissões e depende diretamente do *Web Service*. Todas essas classes possuem uma classe abstrata; o estereótipo <<*Aspecto*>> que garante a execução das autenticações. A classe "*Web Service*" mantem a interoperabilidade entre aplicações, banco, base de dados e está diretamente ligada ao aspecto através de sua estrutura de pontos de corte e junção. Essa estrutura executa as operações entre o sistema e o módulo de segurança, onde estão armazenados os atributos de segurança. O modelo representa as funcionalidades das aplicações e seus atributos sem preocupações com requisitos de segurança, deixando para o aspecto essa responsabilidade.

## 6 Trabalhos relacionados

Voelter e Groher (2007) descrevem que durante o ciclo de vida de desenvolvimento de um sistema, considerando-o como uma linha de produção em que se tem vários pontos de vista desde os requisitos até os testes finais, a modelagem é considerada um fator importante que busca a redução do tempo de produção, a reutilização, a redução de custos e seus esforços, além da padronização das atividades. O desenvolvimento de software orientado por modelos facilitou, assim, o entendimento de cada processo ou atividade, ou seja, a união com o DSOA foi considerada como um dos caminhos para obter esses benefícios, principalmente na reutilização de código nos diferentes projetos em linha de produção. Apesar de possuir certa relevância, o trabalho relacionado não considera requisitos de segurança nem relaciona padrões de segurança.

Seguindo esta linha de raciocínio, Khaari e Ramsin (2010) destacam a maturação de processos desde o estágio inicial do ciclo de vida de um software. Nesse sentido, os autores realizam um estudo sobre a técnica de orientação a aspectos e sua utilização em modelos para uma melhor visualização das preocupações comuns consideradas transversais, bem como uma análise da aplicação da orientação a aspectos em alguns setores importantes como a indústria. Destacam também alguns benefícios e algumas complexidades que podem ocorrer, como, por exemplo, neste trabalho os autores buscam por padrões para processos de software com DSOA e tratam da criação de um processo genérico que acompanhe todo o ciclo de vida de um software. Inicialmente, os autores abordam algumas questões sobre *Web Service*, mas não consideram requisitos de segurança, gestão de riscos ou padrões de segurança.



## 7 Considerações Finais

Como pontos conclusivos, destaca-se a descrição feita, neste estudo, sobre o uso da modelagem orientada a aspectos para representar requisitos de segurança, descritos como padrões em *Web Services*. Durante a pesquisa, aspectos foram usados para representar esses importantes requisitos, de forma que a sua aplicação,

ainda que não muito difundida, principalmente em *Web Service*, tem como vantagens um nível maior de abstração, inserção de novos requisitos, garantia de autenticação e controle de operações sem sobrecarregar as aplicações.

A proposta mostrou-se, então, viável, pois minimiza a complexidade de inserção dos requisitos de segurança em *Web Services*, possibilita uma abrangência em níveis diferentes de camadas de sistemas e são independentes de aplicações. Em contrapartida, exige cautela e análise mais específica na fase de projeto e implementação.

Pontua-se que alguns benefícios como a redução do número de linhas de código, reusabilidade e menor tempo de resposta do sistema podem vir a ser observados. Uma vez que os requisitos de segurança que estão espalhados por grande parte das aplicações, podem ser armazenados em um único módulo e invocado pelo aspecto nas áreas que estão sendo executadas.

Referente à manutenção e atualizações do sistema, pode-se dizer que estas se tornam menos onerosas e menos complexas, pois é realizada em alguns módulos do sistema, ou seja, o aspecto é encarregado de distribuí-las através de *Web Service* para todo o sistema.

Os padrões de segurança propostos pelos principais autores, e utilizados neste trabalho, foram satisfatórios e coerentes com as necessidades de segurança em *Web Services* e operações bancárias. Dessa forma, destaca-se que a importância desses padrões e seu reuso podem ser ampliados com a técnica de aspectos.

Nesse contexto, a orientação a aspectos vem ao encontro às propostas de tecnologias que auxiliam na proteção de sistemas e, principalmente, em *Web Services*; tecnologia bastante vulnerável no quesito segurança.

Assim, identificam-se como trabalhos futuros um estudo mais aprofundado em requisitos de segurança envolvendo a autenticação através do uso da biometria, o reconhecimento por atributos, à segurança de operações na *Cloud Computing* visando à solução de problemas de parametrização em *Web Services*, sobretudo no que se refere ao comportamento de aspectos relacionados à herança de requisitos de segurança ou de qualquer outro requisito que apresente preocupações espalhadas ou sobrepostas.

## Referências

CURPHEY, Mark. FOUNDSTONE, Rudolph Araujo. **Web Application Security Assessment Tools**. IEEE Security & Privacy, pp. 32-41, 2006.

ENDREI, Mark et al: **Patterns: service-oriented architecture and Web Services**. IBM Corporation, Riverton, NJ, 2004.

FERNANDEZ, E.B., DELESSY, N.: **Using patterns to understand and compare Web Services security products and standards**. Proceedings of the Int. Conference on Web Applications and Services (ICIW'06), Guadeloupe. IEEE Comp. Society, 2006.

JENSEN, M., GRUSCHKA, N., HERKENHÖNER, R., & LUTTENBERGER, N.: **SOA Web Services: New technologies - new standards - new attacks**. In Proceedings of the 5th IEEE european conference on Web Services (ECOWS), 2007.

KHAARI, M.; RAMSIN, R.; **Process Patterns for Aspect-Oriented Software Development**. In: IEEE International Conference and Workshops on Engineering of Computer-Based Systems; Department of Computer Engineering Sharif University of Technology Tehran-Iran, 2010.

MEDEIROS, Luisa A.: Marisa-MDD: **Uma Abordagem para a Transformação entre Modelos Orientados a Aspectos: dos Requisitos ao Projeto Detalhado**; Dissertação (Mestrado) – UFRN, Rio Grande do Norte, 2008.

MEIKO, Jensen; FEJA, Sven. **A Security Modeling Approach for Web-Service-Based Business Processes**. ecbs, pp.340-347, 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2009.

MONDAY, Paul B.: **Web Services Patterns - Java Edition**, Apress, 2003.

O'NEILL, Mark et al. **Web Services Security**. 1. ed. Berkeley: McGraw-Hill, 312p., 2003.

PRASAD, A.V. K., RAMAKRISHNA, S., SHRAVANI, D., **Security Patterns for Web Services Mining Agile Architectures**. International Journal of Computer Science and Information Technologies, vol. 1, no. 3, pp. 185-189, 2010.

PRASS, Fábio S.; FONTOURA, Lisandra. M.; RIGHI, Cleber. Aspectos com Padrões de Segurança. Um processo para desenvolvimento em Web Services. **Revista Java Magazine**, v. 93, p. 68-74, 2011.

RASHID, A., COTTENIER, T.; **Aspect-Oriented Software Development in Practice: Tales from AOSD-Europe**. In: IEEE Conf. Aspect-Oriented Software Development (AOSD 10), Europe, 2010.

SCHUMACHER, Markus et al: **Security Patterns Integrating Security and Systems Engineering**. Wiley; 1 edition, 2006.

VOELTER, M.; GROHER, I.; **Product Line Implementation using Aspect-Oriented and Model-Driven Software Development**. In: IEEE, International Software Product Line Conference, Munich-Germany, 2007.

YOSHIOKA, N; WASHIZAKI, H; MARUYAMA, K; **A Survey on Security Patterns**; National Instiute Informatics; Ritsumeikan University, 2008.

YU, Weider D., ARAVIND, Dhanya, SUPTHAWEESUK, Passarawarin, **Software Vulnerability Analysis for Web Services Software System**. iscc, pp.740-748, 11th IEEE Symposium on Computers and Communications (ISCC'06), 2006.

*Autor:*

*Fábio Sarturi Prass*: Doutorando em Ciência da Computação (PUC-RS); Mestre – Programa de Pós-Graduação em Informática da Universidade Federal de Santa Maria (UFSM), na Linha de Pesquisa de Computação Aplicada – Engenharia de Software. Atua como professor na Faculdade Antonio Meneghetti (AMF) no curso de Sistemas de Informação e é Gerente de Projetos da FP2 Tecnologia; domínio em Java, .NET e UML, Banco de Dados.

Submetido em: 30/04/2011
Revisto em: 28/06/2011
Aceito em: 13/09/2011