Arquitetura Genérica para Projeto e Implementação de Criptografia com Programação Orientada a Aspectos

Carlos Seiki Fujii[1], Rodolfo Barros Chiaramonte[1], Valter Vieira de Camargo[2,1]

[1] Fundação de Ensino "Eurípides Soares da Rocha"
Centro Universitário "Eurípides de Marília" – UNIVEM
Marília, São Paulo, Brasil

[2] Departamento de Computação da Universidade Federal de Lavras (DCC-UFLA)
Campus Universitário, Lavras, Minas Gerais, Brasil

fujiicarlos@hotmail.com, chiaramonte@univem.edu.br, valtercamargo@hotmail.com

Resumo. A segurança é um interesse primordial para determinadas aplicações computacionais que compartilham informações ou realizam operações confidenciais por meio de uma rede e o uso de criptografia é uma forma de garantir essa segurança. Apesar de existirem inúmeras aplicações com criptografia, o estudo de criptografia com programação orientada a aspecto ainda é pouco explorado. Além disso, as implementações disponíveis de criptografia com programação orientada a objetos causam entrelaçamento e espalhamento de código de diferentes interesses. Neste artigo, o objetivo é propor uma arquitetura genérica para a implementação de criptografia com aspectos. Essa arquitetura faz uso de um padrão proposto em um trabalho existente e propicia melhores níveis de manutenibilidade e reúso para as aplicações.

Palavras-chave: Padrão Capturador de Dados, Criptografia com Aspectos, Criptografia como um aspecto.

Abstract. Security is an important concern for some applications which share information or perform confidential operations throughout a network. So, cryptography is an alternative for ensuring this security. Although a lot of research has been conducted on this issue, the study of cryptography with aspect-oriented programming is still a recent topic. Besides, the available implementations of cryptography with object-oriented programming lead to tangling and scattering problems. In this paper, the aim is to propose a generic architecture for implementing cryptography with aspects. This architecture includes a pattern proposed in a previous work which propitiates better reuse and maintainability levels.

Keywords: Data Catcher Pattern, Cryptography with aspects, Cryptography as an aspect

[1] Este trabalho é parcialmente apoiado pelo CNPq: Projeto PROSUL-Proc (Latin-AOSD) 490478/2006-9.

1. Introdução

A programação orientada a aspectos (POA) surgiu no final da década de 90 com o objetivo de modularizar um sistema de software de forma mais adequada do que era possível com a orientação a objetos (POO) [8;13]. Segundo os criadores da POA, os conceitos e abstrações da orientação a objetos não são adequados para modularizar todos os tipos de interesses existentes em um sistema de software – os interesses base ou funcionais ficam bem modularizados, mas o mesmo não ocorre com os interesses transversais, como requisitos não-funcionais e restrições globais. Isso leva a problemas de entrelaçamento e espalhamento de códigos com diferentes interesses. Dessa forma, foram propostos novos conceitos e abstrações para permitir o encapsulamento dos interesses transversais em unidades de programação específicas para esse fim. Exemplos dos novos conceitos e abstrações são: aspectos, conjuntos de junção (*pointcuts*), pontos de junção (*join points*) e adendos (*advices*) [8].

Atualmente, em conseqüência da necessidade por agilidade, economia e comodidade o compartilhamento de informações e as transações de operações por uma rede vêm crescendo. Infelizmente, quando essas informações trafegam por uma rede podem ser observadas por terceiro, que nem sempre são bem intencionados. Assim surge a necessidade de proteger as informações por meio de algum método de segurança[11]. A criptografia é utilizada para garantir uma parcela de segurança em aplicações computacionais. Existem várias maneiras de implementar aplicações com criptografia [**Erro! Fonte de referência não encontrada.**1;4;10] A mais utilizada é por meio da programação orientada a objetos (POO) que proporciona a decomposição de sistemas complexos em componentes modulares e funcionais. No entanto, a POO possui limitações na modularização de interesses transversais [7;1], e a criptografia é um exemplo de interesse transversal. Assim, quando POO é usada na implementação de criptografia, o código fonte desse interesse transversal torna-se espalhado e entrelaçado com o código fonte da aplicação, causando problemas como dificuldade de entendimento, manutenção, reúso e adição de outros interesses.

Visando a amenizar problemas como esses, a Programação Orientada a Aspectos (POA) oferece conceitos e abstrações mais eficientes para a modularização de um sistema, e assim possibilita o desenvolvimento dos interesses base e transversais em módulos isolados [8]. A motivação principal para a condução desse estudo são as limitações existentes quando se utiliza o paradigma orientado a objetos para a implementação de criptografia. Quando isso ocorre, o código do interesse de criptografia fica espalhado e entrelaçado pelos módulos funcionais da aplicação. Isso faz com que o entendimento do sistema e conseqüentemente, seus níveis de manutenção e reúso sejam prejudicados. Além disso, outras motivações para o estudo aqui descrito são: o pouco aprofundamento encontrado na literatura com relação à aplicação de POA com criptografia [7;1] e lacunas presentes nesses trabalhos relacionados em relação à tentativa de generalizar a parte de aspecto com criptografia.

O objetivo principal deste trabalho é apresentar uma arquitetura genérica para implementação de criptografia com aspectos. A utilização dessa arquitetura deve melhorar a qualidade de uma aplicação em termos de modularização e, conseqüentemente, com relação à manutenibilidade e reusabilbidade. Para que essa arquitetura pudesse ser elaborada, adaptou-se uma biblioteca de criptografia existente [4] para o contexto da POA utilizando um padrão de projeto [6] proposto em trabalho

existente. A utilização desse padrão permite que essa arquitetura torne-se genérica para que possa ser reutilizada em outros contextos.

2. Conceitos Fundamentais

O pacote Cripto, implementado em Linguagem Java, pertence ao sistema SISCO, ambos desenvolvidos por Chiaramonte [4]. Esse pacote possui três interfaces básicas:

A Interface "Algoritmo" possui os métodos mais genéricos que são comuns entre os algoritmos criptográficos utilizados, sendo eles: `doFinal()` responsável por finalizar uma operação envolvendo criptografia; `update()` responsável por realizar uma operação parcial de criptografia; e `getOutputSize()` que retorna qual o tamanho necessário para armazenar o resultado após uma chamada ao método `doFinal()`. A Interface "Simétrico" foi criada para agrupar os algoritmos simétricos de criptografia (Exemplo: AES, DES, IDEA). Ela é composta dos métodos: `defineChave()` que é responsável por definir a chave que será utilizada na operação; `criaChave()` responsável por criar uma chave aleatória; `obtemChave()` que retorna qual a chave atualmente utilizada; `defineModo()` para definir se será realizada a criptografia ou decriptografia; `limpaChave()` responsável por apagar a chave atualmente em uso; e `temChave()` para verificar se existe alguma chave definida para algoritmo. A Interface "Assimétrico", criada para os algoritmos assimétricos é semelhante à utilizada para os algoritmos simétricos, no entanto, existe uma grande diferença no método de definição de chaves uma vez que devem existir métodos específicos para manipulação da chave.

O padrão Capturador de Dados proposto por Camargo e Maseiro [2] é útil no projeto e implementação de Frameworks Orientados a Aspectos [3], pois auxilia na generalização dos adendos, diferentemente da maioria dos trabalhos que procuram somente generalizar os conjuntos de junção. Com o uso desse padrão de projeto, o engenheiro de aplicação tem maior flexibilidade para identificar um ponto de junção no código-base para realizar o entrecorte por parte dos aspectos. Uma das maiores utilidades do padrão é facilitar e tornar mais genérica a captura de objetos e/ou valores do código-base fornecendo alternativas de composição.

Na Figura 1 é mostrado um diagrama de classes do padrão em que cada classe com o estereótipo <<aspect>> representa um aspecto. O estereótipo <<hook>> representa métodos-gancho que devem ser sobrepostos pelo engenheiro de aplicação. Sua finalidade é facilitar a documentação e a identificação dos pontos de extensão dos aspectos. Os aspectos abstratos `Aspecto` e `AspectoIntermediario` são responsáveis pelos conjuntos de junção abstratos e adendos. O método abstrato `getPointObject()` é utilizado para obter um objeto do contexto de execução que possa ser utilizado para capturar o dado desejado. Esse método é redefinido em cada um dos aspectos especializados (`This`, `Target` e `Arguments`) com o objetivo de retornar diferentes objetos do contexto de execução, fornecendo assim três alternativas de composição [2]. Cada alternativa de composição é representada por um aspecto abstrato, sendo eles: `WithReturn`, `This`, `Target` e `Arguments`. O processo de escolha por uma dessas alternativas de composição ocorre quando o desenvolvedor cria um aspecto concreto que estende algum desses aspectos abstratos. O aspecto abstrato `WithReturn`, é uma alternativa que deve ser estendida se o objeto

a ser capturado do código base for um retorno de um método ou uma chamada. O aspecto `Target` deve ser estendido quando o objetivo for capturar do código-base o objeto `This` do ponto de junção fornecido no aspecto concreto. O aspecto `This` deve ser estendido quando o objetivo for capturar o objeto `This` do código-base e o aspecto `Arguments` deve ser estendido quando o objetivo for capturar algum parâmetro de um ponto de junção.

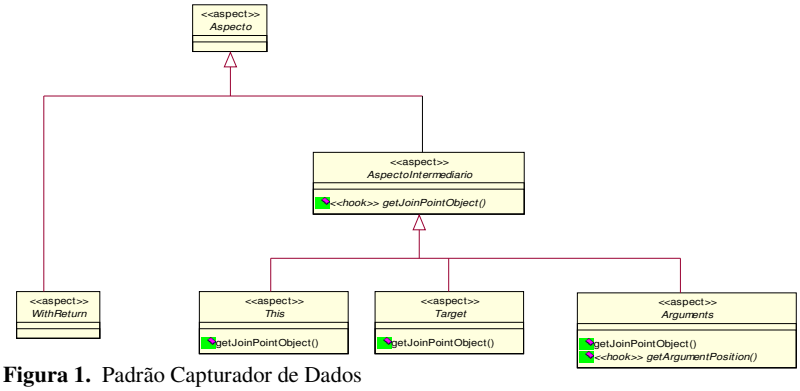

Figura 1. Padrão Capturador de Dados

3. Arquitetura Proposta

Nesta seção é mostrada a arquitetura proposta, a qual foi criada e evoluída a partir de um conjunto de diretrizes formuladas por Fujii *et.al.*[5]. Essa arquitetura é composta de três níveis horizontais com a finalidade de isolar o interesse transversal de criptografia de um lado e a aplicação base do outro.

Na arquitetura proposta, a camada de aspectos genéricos foi desenvolvida com a finalidade de agir como intermediária entrecortando pontos da aplicação base e utilizando a biblioteca de criptografia [4]. Diferentemente de outros trabalhos relacionados à POA, o objetivo dessa arquitetura é não incluir o código do interesse transversal de criptografia dentro dos aspectos, como ocorre com a maior parte dos trabalhos [1;7]. Ao contrário disso, o objetivo aqui é apenas a utilização dos aspectos como conectores entre as duas partes OO.

Para exemplificar essa arquitetura, na Figura 2 é mostrado um exemplo de sistema bancário com uso de criptografia utilizando a arquitetura proposta. Pode-se observar a separação do sistema bancário em três camadas distintas. A camada 1 corresponde à aplicação base e é formada pelas classes `CaixaEletronico`, `Servidor` e `BancoDeDados`. Sua função primária é realizar operações bancárias por meio de um caixa eletrônico que está interligado a um servidor que retorna as solicitações pedidas pelo caixa eletrônico consultando um banco de dados. Na camada 2 é realizada a conexão ente as camadas 1 e 3 e é formada pelos aspectos abstratos referentes ao padrão Capturador de Dados [2] e um aspecto concreto que é responsável por definir qual alternativa de composição foi escolhida. Os aspectos abstratos `WithReturn` e `AspectoIntermediario`, contidos na camada de aspecto, possuem chamadas a métodos de classes que realizam a criptografia, ou seja, fazem a conexão com a

camada 3. O aspecto abstrato `Aspecto` possui conjuntos de junção abstratos que selecionam os pontos de junção.

A fim de ilustração, o aspecto abstrato `This` foi o escolhido dentre as quatro alternativas como a melhor opção para concretização. Isso pode ser visto pois existe um aspecto concreto chamado ilustrativamente de `AspectoConcreto` estendendo o aspecto `This`. Esse aspecto concreto entrecorta pontos específicos das classes `CaixaEletronico` e `Servidor` onde se requer o uso da criptografia, ou seja, ele realiza a conexão com a camada 1.

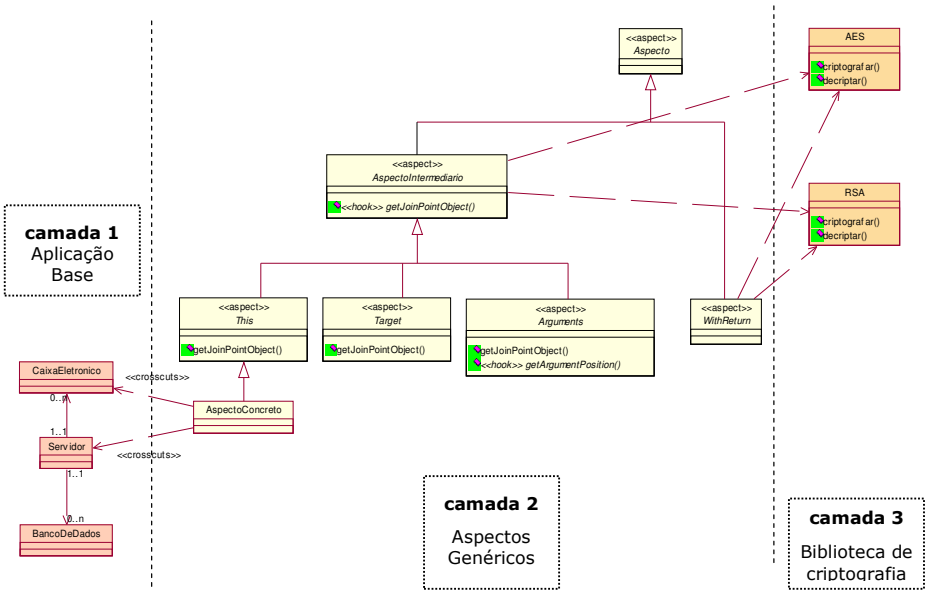

Figura 2. Exemplo de sistema desenvolvido utilizando a arquitetura proposta

Na camada 3, também como ilustração, são apresentadas duas classes que são responsáveis pela realização da criptografia. Por meio do exemplo do sistema bancário, evidencia-se uma situação real para melhor esclarecer a influência da estrutura em um sistema. A situação em questão é uma solicitação de saldo por um usuário. Quando o usuário solicita seu saldo em um caixa eletrônico as seguintes ações são geradas: 1) para validar a solicitação de saldo, o cliente precisa informar sua senha correta; 2) a senha precisa ser conferida no servidor para ser validada, contudo não é seguro o tráfego de uma senha bancária por uma rede; 3) antes do envio da senha ao servidor, o aspecto `AspectoConcreto` entrecorta a classe `CaixaEletronico` para criptografar a senha; 4) ainda na camada de aspectos é feita uma chamada ao método `criptografar()` das classes `AES` ou `RSA` e finalmente, é aplicada a criptografia na senha; 5) a camada de aspecto devolve a senha, agora criptografada, e o fluxo de execução à classe `CaixaEletronico`; 6) a classe `CaixaEletronico` envia a senha criptografada à classe `Servidor`; 7) antes da classe `Servidor` validar a senha, a camada de aspecto a entrecorta e realiza o processo inverso da criptografia, ou seja, pega a senha criptografada, faz

chamadas ao método `decriptar()`, decripta a senha e a devolve à classe `Servidor`. 8) a classe `Servidor` confere a senha no banco de dados e caso seja válida, ele devolve o valor do saldo à classe `CaixaEletronico`; 9) e por fim, para o envio do valor do saldo, acontece um processo semelhante ao ocorrido no envio da senha. Com a diferença que agora quem envia um dado é a classe `Servidor` para a classe `CaixaEletronico`.

4. Estudo de Caso

O estudo de caso que foi desenvolvido consiste em um sistema composto de uma aplicação base, de uma camada de aspectos genéricos e uma biblioteca de criptografia. A aplicação base é um sistema cliente/servidor, desenvolvido em Linguagem Java, para a realização compras e vendas on-line pela Internet. A camada de aspectos genéricos utiliza o Padrão Capturador de Dados [2] para a biblioteca de criptografia utilizou-se o pacote Cripto [4]. O objetivo do estudo de caso é verificar a possibilidade e os problemas encontrados na tentativa do acoplamento de uma biblioteca de criptografia já finalizada em uma aplicação sem fazer alterações tanto na aplicação quanto na biblioteca de criptografia.

A aplicação base utilizada nesse estudo de caso é um sistema simples de compras on-line. Nesse sistema, o usuário informa seus dados por meio de um applet (representado no sistema pela classe `ClienteGUI`), os quais são enviados ao servidor (classe `Server`) para que sejam armazenados em um banco de dados e posteriormente consultados se necessário. A classe `ClienteGUI` representa a interface com o usuário. O usuário faz o download desse *applet* a partir de uma página web. Assim que `ClienteGUI` é iniciado ele faz a conexão, conforme sua restrição de segurança, com a classe `Server`. A classe `Server` recebe o pedido de conexão e após aceitá-lo, fica aguardando algum pedido de `ClienteGUI`. Esse pedido aguardado pelo `Server` pode ser um cadastro enviado pela interface para concretizar uma compra ou apenas um consulta ou alteração do cadastro já existente.

Além das classes `ClienteGUI` e `Server`, a aplicação base também contém outras classes, entre elas as classes `Client` e `DBConnection`. A classe `Client` possui métodos para o cadastramento, consulta ou alteração dos dados do usuário no banco de dados e a classe `DBConnection` realiza a conexão do servidor com o banco de dados.

A camada de aspectos é formada por uma hierarquia de aspectos de acordo com o padrão Capturador de Dados, e é composta de 14 aspectos e uma classe. Dentro da camada de aspectos, existe uma divisão em duas partes: uma relacionada à criptografia e outra relacionada à decriptografia. Cada uma dessas partes foi projetada com o uso do padrão. Na Figura 3 é mostrada a parte relacionada à criptografia.

Na parte da criptografia, a hierarquia começa com o aspecto abstrato `AbstractEncryption` que contém um conjunto de junção abstrato. Seu código fonte é apresentado na Figura 4.

Abaixo do aspecto `AbstractEncryption`, existem os aspectos abstratos, `IntermediaryEncryption` e `EncryptionReturn`. Ambos possuem adendos do tipo `around` e chamadas a métodos das classes AES e RSA da biblioteca Cripto de

Chiaramonte [4] para a realização da criptografia. Esses aspectos fazem a conexão entre camada de aspectos e a biblioteca de criptografia.

O aspecto `IntermediaryEncryption` possui um método abstrato denominado `getJoinPointObject()` que possui um retorno do tipo `Object` e um parâmetro do tipo `JoinPoint`.

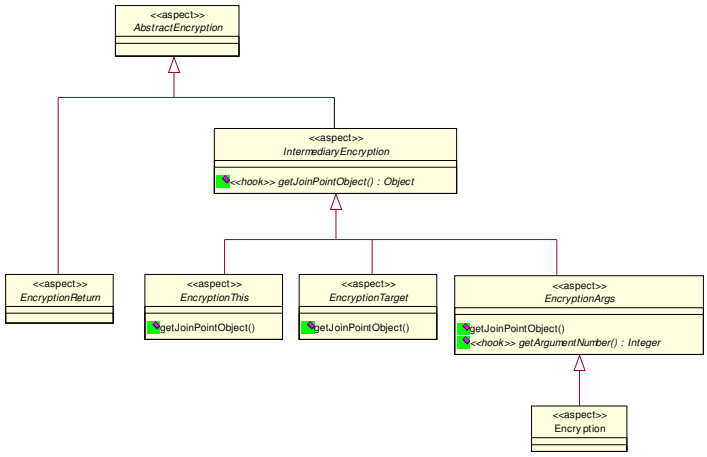

Figura 3. Diagrama de classe da parte de criptografia da camada de aspectos genéricos

```
public abstract aspect AbstractEncryption {
    public abstract pointcut callCriptografaMsg();  }
```

Figura 4. Aspecto abstrato: `AbstractEncryption`

Os últimos aspectos da hierarquia são `EncryptionThis`, `EncryptionTarget` e `EncryptionArgs`. Todos eles concretizam o método `getJoinPointObject()` da classe `IntermediaryEncryption()`. O `EncryptionThis` é uma alternativa que consistem em estender o aspecto `IntermediryEncryption` e fornecer um ponto de junção que a partir de seu objeto this, pode-se obter o valor a ser modificado/criptografado. Esse ponto de junção pode ser uma chamada ou uma execução de um método, ou ainda pode ser um acesso a um atributo.

No final da hierarquia está o aspecto concreto `Encryption`. Esse aspecto é muito importante porque ele mostra qual alternativa de composição do padrão foi escolhida e também declara o ponto de junção da aplicação que deve ser entrecortado para capturar o objeto a ser criptografado. Neste estudo de caso, o aspecto abstrato `EncryptionArgs` foi o escolhido para ser concretizado pelo aspecto `Encryption`. Essa escolha foi feita porque o ponto de junção indicado para ser entrecortado foi o `writeObject()` que possui um argumento do tipo `Vector` (ponto de junção da criptografia: "`output.writeObject(v);`"). Na Figura 5 é apresentado o aspecto concreto `Encryption`.

O uso do designador `within` no ponto de junção do aspecto mostrado na Figura 5 é em conseqüência da necessidade de discriminar as classes que serão entrecortadas pela camada de aspectos genéricos. Visto que a classe `FileCryptography` também

possui o mesmo método indicado como ponto de junção, mas o qual não é de interesse no entrecorte, assim foi preciso excluí-la da relação de classes afetadas pelo conjunto de junção dos aspectos `Encryption` e `Decryption`.

```
public aspect Encryption extends EncryptionArgs{
       public pointcut callCriptografaMsg(): call(* *.writeObject(..)) &&
                !within(FileCryptography);
       public int getArgumentNumber(){
                int argumentNumber = 0;
                return argumentNumber; }}
```

Figura 5. Aspecto concreto: `Encryption`

A outra parte da camada de aspectos genéricos está relacionada com a decriptografia, mostrada na Figura 6. A parte de decriptografia é muito semelhante à parte de criptografia, porém possui algumas diferenças. A primeira das diferenças são as chamadas a métodos relacionados a decriptografia e não a criptografia, a segunda diferença é a presença de uma classe chamada `FileCryptograhy` que realiza ações pertinentes a criptografia assimétrica. E a terceira diferença é a escolha pela concretização do aspecto `DecryptionReturn` dentre as quatro alternativa possíveis.

`DecryptionReturn` foi escolhido para ser concretizado devido ao ponto de junção da decriptografia retornar o dado que se deseja modificar. Esse ponto de junção é o método `readObject()` (Ponto de junção da decriptografia: "`v = (Vector) input.readObject();`"). A concretização do `DecryptionReturn` é feita no aspecto `Decryption`.

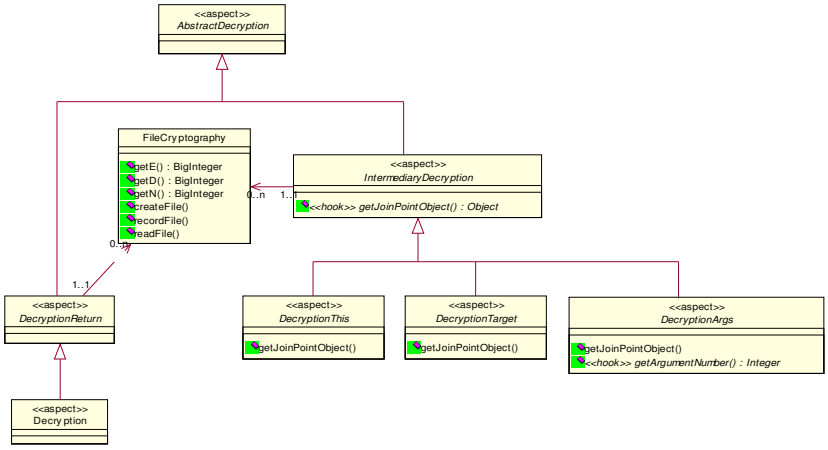

Figura 6. Diagrama de classe da parte de decriptografia da camada de aspectos genéricos

5. Trabalhos Relacionados e Conclusão

O uso da criptografia com POA ainda é pouco explorado, visto que poucos pesquisadores têm trabalhado nessa linha [1;7]. Boström [1] descreve 5 passos realizados para a inclusão do interesse transversal de Criptografia em uma aplicação

que inicialmente não foi desenvolvida com esse interesse em mente: 1) Construir uma classe em Java para criptografar e decriptar; 2) Identificar os pontos de junção; 3) Construir conjuntos de junção para os pontos de junção identificados no passo 2: 4) Adicionar adendos que atuem sobre os conjuntos de junção definidos no passo 3 e 5) Testar a aplicação. Huang *et al*. [7] descrevem em AspectJ [13], uma biblioteca reusável e genérica com funções de segurança denominada JSAL (*Java Security Aspect Library*). Durante a construção da biblioteca utilizaram-se pacotes de segurança JCE e JAAS, que são popularmente difundidos. Também foram utilizados aspectos abstratos e conjuntos de junção abstratos para obter bons níveis de generalidade e com isso contribuir para a reusabilidade dos aspectos.

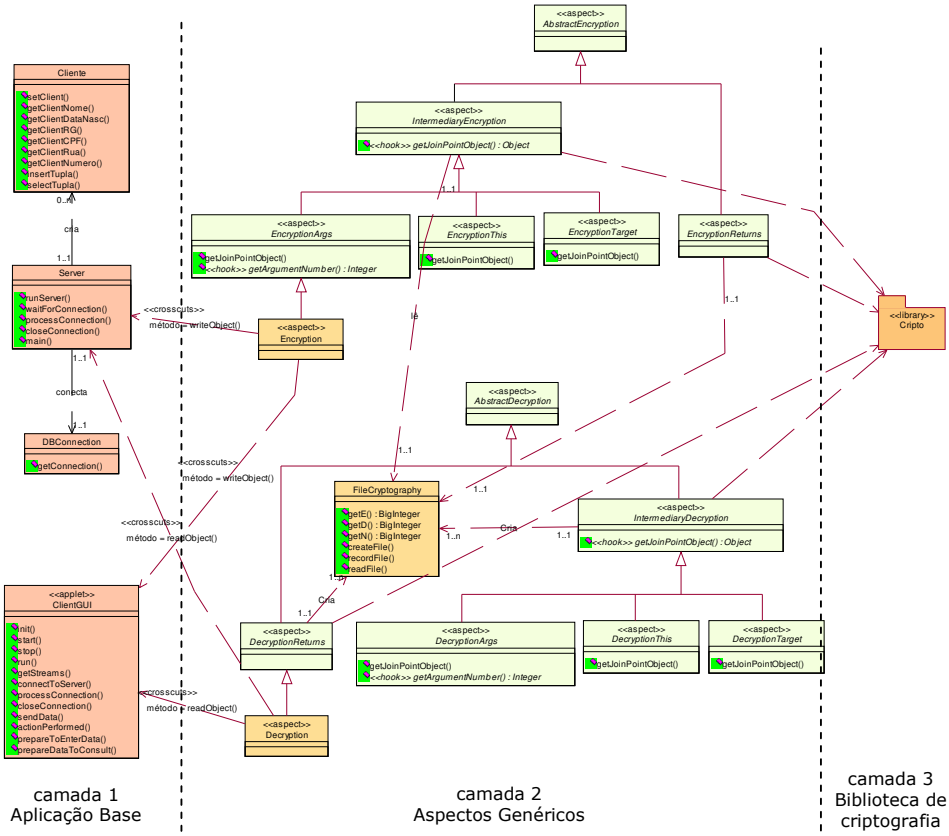

Figura 7. Diagrama de classe da aplicação da arquitetura proposta

Este trabalho apresenta uma proposta de arquitetura genérica para projeto e implementação de criptografia com programação orientada a aspectos. O objetivo é que a arquitetura genérica apresentada seja utilizada por outros desenvolvedores que tenham interesse em modularizar o interesse transversal de criptografia com aspectos.

A utilização dessa arquitetura, juntamente com o padrão Capturador de Dados [2] propicia melhor modularização do sistema e, conseqüentemente, melhores índices de manutenção e reúso.

A elaboração da camada de aspectos genéricos melhora a modularização do interesse transversal de criptografia, elimina o entrelaçamento e o espalhamento de código quando comparado com sua versão OO. Conseqüentemente, tende facilitar o acoplamento da biblioteca de criptografia a outros códigos-base sem que esses tenham consciência da sua presença, podendo elevar os níveis de reúso e manutenibilidade da aplicação final.

Um outro ponto positivo da arquitetura proposta é a possibilidade de evoluí-la para se tornar um Framework Orientado a Aspectos. Com base na experiência dos autores com o desenvolvimento desse tipo de framework [3], acredita-se que poucas evoluções na arquitetura podem se feitas para considerá-la um framework orientado a aspectos.

Referências Biblográficas

1. Boström, G. Database Encryption as an Aspect. In Belgian AOSD Workshop, 1st, 2004, Vrije Universiteit Brussel, Bélgica
2. Camargo, V. V.; Masiero, P. C. A Pattern to Design Crosscutting Frameworks. In: Proceedings of the Annual ACM Symposium on Applied Computing (ACM-SAC´08), 23, 2008, Fortaleza, Brasil.
3. Camargo V.V., Masiero, P.C. Frameworks Orientados a Aspectos. In: Proceedings of Simpósio Brasileiro de Engenharia de Software (SBES), 2005, Uberlândia, MG, Brasil.
4. Chiaramonte, R. B. Sico: Um Sistema Inteligente de Comunicação de Dados com Suporte Dinâmico a Segurança. 2006. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Marília, 2006.
5. Fujii, C. S; Chiaramonte, R. B.; Camargo, V. V. Camada de Aspecto para Modularização de Criptografia. In: Simpósio Internacional de Iniciação Cientifica da Universidade de São Paulo, 15, 2007, São Carlos
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J. – Design Patterns – Elements of Reusable of Object Oriented Software, 11. Addison-Wesley. 1995.
7. Huang, M.; Wang, C.; Zhang, Z. Toward a Reusable and Generic Security Aspect Library. AOSD 2004 Workshop - AOSD Technology for Application-level Security (AOSDSEC), 2004, Lancaster, UK
8. Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C. V.; Loingtier, J.; Irwin, J. Aspect-Oriented Programming. In: Published in proceedings of the European Conference on Object-Oriented Programming (ECOOP), Springer-Verlag LNCS 1241, junho de 1997, Xerox Palo Alto Research Center, Technical Report, Finland, p. 25
9. Stallings, W. Cryptography and Network Security - Principles and Practice. 2. ed. Prentice Hall, 1998
10. Cryptography and Network Security - Principles and Practice. 4. ed. Prentice Hall, 2005. 592p
11. Terada, R. Segurança de Dados- Criptografia em Redes de Computador. 1. ed. São Paulo: Edgard Blücher, 2000.
12. TKOTZ, Viktoria. Criptografia Numaboa, http://www.numaboa.com/content/view/157/57/
13. Winck, D. V.; Goetten Junior, V. AspectJ: Programação Orientada a Aspectos com Java. São Paulo: Novatec Editora, 2006