UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Samuel Cristhian Schwebel

Frasedare: Framework Orientado a Objetos para Segurança de Dados em Repouso

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Luiz Carlos Zancanella
Orientador

Florianópolis, Dezembro de 2005

Frasedare: Framework Orientado a Objetos para Segurança de Dados em Repouso

Samuel Cristhian Schwebel

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação na Área de Concentração de Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Raul Sidnei Wazlawick, Dr.

Banca Examinadora

_____

Luiz Carlos Zancanella, Dr. (Orientador)

_____

Frank Augusto Siqueira, Dr.

_____

Júlio da Silva Dias, Dr.

_____

Ricardo Pereira e Silva, Dr.

Agradecimentos

Em primeiro lugar a Deus, por todas as bençãos recebidas.

A minha amada esposa Micheli, pela compreensão e apoio.

A minha mãe e pai, que sempre buscaram me fortalecer nas dificuldades.

Ao prof. orientador Luiz Carlos Zancanella, pela paciência e auxilio.

Ao amigo Flávio Exterkoetter pelo incentivo e contribuições.

A todos que contribuiram direta ou indiretamente para a conclusão deste trabalho.

Sumário

Lista de Abreviaturas

AC – Autoridade Certificadora
AES – *Advanced Encryption Standard*
AR – Autoridade de Registro
API – *Application Programming Interface*
CAPI – *Cryptographic Application Programming Interface*
COM – *Component Object Model*
CSF – *Cryptographic Support Facility*
CSI – *Computer Security Institute*
DES – *Data Encryption Standard*
DSA – *Digital Signature Algorithm*
ECC – *Elliptic Curve Cryptography*
ECDSA – *Elliptic Curve Digital Signature Algorithm*
FIPS – *Federal Information Processing Standard*
GCS – *Generic Crypto Service*
GSS – *Generic Security Service*
ICP – Infra-estrutura de Chaves Públicas
ITU – *International Telecommunications Union*
JCA – *Java Cryptography Architecture*
JCE – *Java Cryptography Extension*
JFC – *Java Foundation Classes*
MFC – Microsoft *Foundation Classes*
NIST – *National Institute of Standards and Technology*
OO – Orientação à Objeto
RFC – *Requests for Comments*
SHA – *Secure Hash Algorithm*
TI – Tecnologia da Informação
UML – *Unified Modeling Language*

Lista de Figuras

Lista de Tabelas

Resumo

Este trabalho apresenta a construção do framework Frasedare, um framework orientado a objetos de subsistema que implementa a segurança de dados em repouso, isto é, segurança de dados contidos em dispositivos de armazenamento. Um framework orientado a objetos é uma solução reusável para um domínio específico baseada no paradigma da orientação a objetos. O domínio específico do framework Frasedare é a segurança de dados em repouso, que consiste em garantir a segurança em termos de sigilo, integridade, autenticação e não-repúdio. Estas garantias são alcançadas através da implementação de mecanismos de criptografia em subclasses estendidas do framework Frasedare. A construção do framework Frasedare contemplou: a fase de *desenvolvimento*, compreendendo as atividades de análise de domínio, projeto arquitetural, projeto do framework, implementação e teste; a fase de *uso*, com o desenvolvimento de subsistemas que estenderam o framework Frasedare, os quais foram utilizados em uma aplicação teste e em um sistema desenvolvido e comercializado por uma empresa, assim sendo constatada a aplicabilidade do framework Frasedare e; por fim, a fase de *evolução e manutenção*. O framework Frasedare tem como objetivo principal facilitar a inclusão de segurança de dados em repouso nas aplicações.

Palavras Chaves

Frameworks Orientado a Objetos, Banco de Dados, Segurança, Criptografia.

Abstract

This work presents the construction of the Frasedare framework, an object-oriented framework of sub-system which implements the security of data at rest, that is, security of data contained in storage devices. An object-oriented framework is a reusable solution for a specific domain based in the paradigm of the orientation of objects. The specific domain of the Frasedare framework is the security of data at rest, which consists of guaranteeing the security in terms of secrecy, integrity, authentication and not-repudiation. These guarantees are reached through the implementation of cryptography mechanisms in subclasses extended of the Frasedare framework. The construction of the Frasedare framework contemplated: the *development* phase, comprising the activities of domain analysis, architectural project, framework project, implementation and testing; the *usage* phase, with the development of sub-systems that extended the Frasedare framework, which had been used in a test application and a system developed and commercialized by a company, thus being evidenced the applicability of the Frasedare framework; and, finally, the *evolution and maintenance* phase. The Frasedare framework has as main objective to facilitate the inclusion of security of data at rest in the applications.

Key-words

Object-Oriented Frameworks, Data Base, Security, Cryptography.

1. Introdução

Este trabalho apresenta uma forma, flexível e extensível, de facilitar a inclusão de segurança de dados em repouso[1] no desenvolvimento de aplicações.

Neste contexto, a segurança de dados em repouso significa garantir o sigilo, integridade, autenticação e não-repúdio, e a facilidade de inclusão destas garantias de segurança em uma aplicação, pode ser alcançada através de uma solução reusável que implemente os meios para se garantir a segurança. Quando se trata de soluções reusáveis, se destacam os frameworks orientados a objetos. Um framework orientado a objetos é um conjunto de classes cooperantes, que fornecem um projeto reutilizável, flexível e extensível para um domínio específico. O domínio para o framework apresentado neste trabalho é a segurança de dados em repouso de uma aplicação.

Em um framework algumas partes já estão definidas e prontas para o reuso, outras são flexíveis e extensíveis. A flexibilidade fornecida pelo framework é obtida através da possibilidade de se definir características específicas da aplicação em pontos já determinados do framework, que podem ou devem ser configurados; e a extensibilidade é a capacidade de se estender alguma classe do framework, adicionando novas funcionalidades necessárias para uma aplicação.

Desta forma, o desenvolvedor terá como adicionar em uma aplicação, recursos de segurança de dados em repouso sem a necessidade de um conhecimento profundo sobre o assunto. Porém, o desenvolvedor com conhecimento mais profundo em frameworks e segurança, poderá adicionar suas funcionalidades específicas através da extensibilidade proporcionada pelo framework.

[1] Segurança de Dados em Repouso – Neste trabalho em muitos casos foi utilizada a expressão “dados em repouso” para diferenciar de dados que se encontram nos meios de transmissão e redes, ou seja “dados em trânsito” BURNET & PAINE (2001). Como sinônimo, poderia ser utilizado “dados armazenados”.

1.1. Motivação

No contexto atual, os setores privado e público são dependentes de TI (tecnologia de informação) para fazerem desde suas funções básicas até as mais complexas e críticas. Para o uso ideal da TI nos atuais ambientes de sistemas e redes interconectadas, cada vez mais distribuídos e abertos, deve-se ter uma atenção especial quanto à segurança de dados em trânsito (redes, transmissões de dados) e dos dados em repouso (armazenados).

A segurança de dados deve ser considerada para informações sensíveis, que possuam um valor elevado, ou que representam um prejuízo elevado se forem vulneráveis a divulgação desautorizada ou ainda a modificação não detectada durante a transmissão ou quando armazenadas. Existem meios de segurança para se proteger os dados contra comprometimento e alteração intencional ou acidental.

Com determinados meios de segurança, como por exemplo, mecanismos de criptografia, pode-se obter funcionalidades para comunicação segura, onde uma informação é cifrada antes de ser transmitida e decifrada depois que o destino a recebe. Também os mecanismos de criptografia proporcionam funcionalidades para segurança de dados em repouso, por cifrar os dados antes de serem enviados para um meio de armazenamento (banco de dados, arquivo) e por decifrar os dados quando forem recuperados deste meio de armazenamento. Estes últimos – mecanismos de criptografia para segurança de dados em repouso – são um dos objetivos de estudo deste trabalho.

Como uma forma de entender melhor a necessidade de aplicar segurança aos dados em repouso, abaixo será descrito um exemplo que pode ser encontrado de outras formas e dentro de outras aplicações, porém com as mesmas necessidades: a segurança de dados em repouso, alcançada através da implementação de meios de segurança.

Considere uma clínica médica possuindo vários profissionais da área, os quais registram os prontuários de seus pacientes em uma aplicação, esta aplicação armazena estes prontuários em um banco de dados central. Esta clínica também possui uma equipe de TI, que desenvolve e mantém seus aplicativos. Dentro deste contexto, surgem alguns questionamentos.

Considerando que o pessoal de TI tem acesso total às informações armazenadas no banco de dados, surge a primeira questão. Um prontuário, por uma questão de ética, deve ser mantido em sigilo. Por este motivo, ninguém além do paciente e do profissional da área de saúde que atendeu o paciente e registrou o prontuário, poderia ter acesso à informação. Uma segunda questão é quanto à integridade dos dados (informações), pois se existe uma forma de outra pessoa, além do próprio profissional da área de saúde, acessar estes dados, então também poderá alterá-los. Outra questão, a terceira, é a necessidade de se provar se os dados originais foram alterados, e por quem foi alterado; aqui entra outra necessidade de segurança importante, a autenticação.

Além das questões citadas acima, existe uma quarta questão que diz respeito ao que normalmente é chamado de não-repúdio. Dentro do exemplo aqui discutido, o não-repúdio faz com que o profissional da área de saúde que atendeu o paciente e registrou o prontuário, não tenha como negar a autoria do mesmo. Isto garante ao paciente uma forma de provar a responsabilidade de quem o atendeu.

Esta breve descrição acima é somente um exemplo no qual se pode detectar a necessidade de se aplicar a segurança de dados em repouso. Dentro deste mesmo contexto ainda pode-se considerar as administradoras de planos de saúde. Pode-se ainda citar outros exemplos que necessitam de segurança de dados em repouso, como os dados pessoais de clientes, tais como o número de cartão de crédito, CPF, informações sobre contas bancárias e datas de aniversário, que normalmente são armazenados em bancos de dados das corporações. Existem leis e regulamentos em grande parte dos países que determinam e definem diretrizes para se garantir a segurança e privacidade de dados pessoais.

Muitas vezes, os desenvolvedores não se preocupam com estas questões de segurança, em outras, não detém o conhecimento técnico necessário para atender os requisitos de segurança para os dados em repouso.

Desta forma, este trabalho tem com motivação principal, fornecer uma solução reusável com o objetivo de facilitar a adição dos serviços de segurança para dados em repouso nas aplicações, sem que o desenvolvedor necessite de conhecimento profundo em segurança. Porém, também fornecerá ao desenvolvedor que detém um conhecimento

mais especializado, a possibilidade de adicionar novas funcionalidades através da flexibilidade e extensibilidade fornecidas.

Com isso, têm-se duas questões bem definidas. A primeira questão é alcançar uma forma de facilitar a implementação, e a segunda é o domínio que deve ser atendido, ou seja, aplicar os serviços de segurança para dados em repouso.

Uma forma de se obter a facilidade na implementação de aplicações é fazer uso de soluções reusáveis, como componentes e frameworks.

Para este contexto, aplicam-se os frameworks orientados a objetos. Por serem conjuntos de classes cooperantes que formam projetos – flexíveis e extensíveis – reutilizáveis para domínios específicos, um framework orientado a objetos que implemente a segurança nos dados trará para o desenvolvedor a facilidade – por reutilizar projeto e implementação – em adicionar segurança de dados em suas aplicações, sem ter que projetar e implementar desde o início todas as funcionalidades requeridas.

1.2. Objetivo Geral

O principal objetivo deste trabalho é a construção de uma solução reusável que seja flexível e extensível, e que forneça funcionalidades de segurança de dados em repouso, em aplicações novas e existentes.

1.3. Objetivos Específicos

Para alcançar o objetivo geral mencionado acima foram estabelecidos os seguintes objetivos específicos:

- Identificar as necessidades comuns em aplicações já existentes para o domínio a que o framework proposto se destina;
- Definir e delimitar o domínio para o framework a ser construído;
- Definir uma arquitetura, que formará a base para o projeto do framework orientado a objetos;

- Definir o projeto do framework orientado a objetos conforme os conceitos e domínios estudados e definidos;
- Implementar o projeto do framework orientado a objetos em uma linguagem de programação orientada a objetos;
- Fazer uso do framework desenvolvido buscando identificar ajustes necessários frente ao domínio e reuso.

1.4. Justificativa

Legislação e regulamentos sobre privacidade foram decretados em muitos paises no mundo antes mesmo da era digital. O direito à privacidade está na constituição de países em todos os continentes sendo considerado um direito humano fundamental (ORACLE, 2003). A Constituição Brasileira trata sobre o direito à privacidade em seu artigo 5º, inciso X, onde diz: "são invioláveis a intimidade, a vida privada, a honra e a imagem das pessoas, assegurado o direito à indenização pelo dano material ou moral decorrente de sua violação" (CONSTITUIÇÃO, 1988). Também legislações e regulamentos decretados recentemente designam responsabilidade corporativa para a segurança de dados financeiros e pessoais.

O regulamento *Sarbanes-Oxley Act* foi criado em 2002 nos Estados Unidos após problemas com relatórios contábeis em grandes empresas (SARBANES-OXLEY, 2002). Apesar de ser Norte Americano, o regulamento afeta todas as empresas que possuam ações nas bolsas de valores dos Estados Unidos. A fim de restaurar a confiança nos relatórios contábeis, o regulamento responsabiliza pessoalmente os funcionários da corporação pelo fornecimento de informações contábeis públicas precisas aos investidores. Desta forma, os controles internos são destacados como fundamentais para a integridade e precisão das informações e, cada vez mais, a segurança é vista como a parte central de qualquer sistema eficiente de controle interno.

A Diretiva de Proteção de Dados da União Européia (*European Union Data Protection Directive*) exige que cada país membro da União Européia aprove a legislação que exige controles de sigilo e integridade para redes, sistemas e dados que contêm informações pessoais. Esta diretiva especifica que todas as informações pessoais

coletadas, como de funcionários ou clientes, devem ser protegidas contra destruição acidental ou ilegal, perda, alterações e divulgação ou acesso não autorizado (EU, 2002).

Privacidade e segurança são conceitos que se relacionam de forma muito próxima, apesar de não serem sinônimos. Privacidade envolve o direito de uma pessoa escolher e controlar sobre o uso de seus dados pessoais. A pessoa deve estar ciente sobre os dados pessoais que estão sendo utilizados e também ter a garantia de que estarão protegidos adequadamente. A segurança é a proteção dos dados de um acesso não autorizado, ou seja, é um conjunto de meios para se alcançar a privacidade de determinados dados. As principais máximas da segurança são o sigilo, a integridade e a disponibilidade dos dados (ORACLE, 2003).

Preocupações sobre proteção de dados vêm crescendo com o passar do tempo. Segundo BURNETT & PAINE (2002), o CSI (*Computer Security Institute*) constatou que: "55% dos entrevistados da nossa pesquisa informaram atividades maliciosas relacionadas com o pessoal interno".

BURNETT & PAINE (2002) ainda observam a necessidade de proteção em dados não-autorizados à revelação:

> *"Uma revelação dos dados não-autorizados ocorre quando uma pessoa acessa e lê informações [...] Uma causa comum de acesso não-autorizado é a falha em não cifrar informações que deveriam permanecer em sigilo. Os dados podem ser comprometidos explorando os seguintes tipos de vulnerabilidades:*
>
> - *Armazenar os dados em texto claro (isto é, decifrados) quando forem considerados suficientemente sigilosos.*
> - *Falhar em implementar, monitorar e impor mecanismos de controle de acesso e autorização apropriados no local em que os dados sigilosos são armazenados."*

Também estudos da IDC (*International Data Corporation*) (COMPUTERWORLD, 2004) que apontam para o crescimento dos mercados de armazenamento (*storage*) de dados, revelam que o tema segurança em aspectos relacionados ao armazenamento de dados foi a principal preocupação na maioria dos segmentos pesquisados.

Várias evidências sobre roubo de informações pessoais são relatadas na mídia em geral. SYBASE (2005a) cita alguns casos de destaque que foram publicados:

- Uma instituição acadêmica relatou o roubo de um laptop que tinha os dados pessoais de mais de 98.000 graduados (COMPUTERWORLD, 2005b).
- Um grupo médico perdeu os dados pessoais de 185.000 pacientes quando dois computadores foram roubados (COMPUTERWORLD, 2005a).
- Uma empreiteira federal relatou que 35.000 acionistas podem estar em risco depois que um computador *desktop* foi roubado.

Normalmente, bancos de dados possuem mecanismos de controle de acesso eficientes. Porém somente este controle de acesso não é suficiente para garantir o sigilo e integridade total dos dados, pois os dados armazenados sobre mídias como discos ou fitas, podem ser lidos e escritos através de aplicativos utilitários que lêem o bloco físico destes dados. Uma poderosa solução para garantir o sigilo e integridade destes dados é utilizar a criptografia (DB4O, 2005a) (ORACLE, 2005) (SYBASE, 2005a) (SQL, 2005).

Quanto ao uso da criptografia, observa-se tanto na literatura MENEZES (1997), SCHNEIER (1996), STALLINGS (2003), TANENBAUM (2003) quanto nas entidades do setor privado e público, uma grande preocupação em utilizar a criptografia para os dados em trânsito, dando ênfase na transmissão de dados. Também empresas que fornecem bens e serviços de segurança fazem um bom trabalho em proteger os dados em trânsito de seus clientes. Entretanto, muitos não conseguem perceber que os dados requerem uma maior proteção quando estão em repouso – armazenados (BURNETT & PAINE, 2002).

Para se desenvolver com facilidade aplicações que implementem segurança de dados em repouso através da criptografia, deve-se fazer uso de soluções reusáveis. Uma solução reusável é fazer uso de bibliotecas de classes – APIs de criptografia (CAPIs) – que implementam os vários mecanismos de criptografia (MICROSOFT, 2003) (SUN, 2002) (RSA, 2002). Ao utilizar bibliotecas de classes, o programa principal é responsável pelo controle de fluxo e o desenvolvedor faz chamadas às bibliotecas de classes. Outra solução reusável poderosa é fazer uso de frameworks. Utilizando um framework, há inversão de controle de fluxo e as implementações (especializações) do

desenvolvedor são chamadas pelo framework (GAMMA, 1995). Neste caso, o desenvolvedor não se preocupa com o controle do fluxo e nem com as chamadas à biblioteca de classes.

Segundo JOHNSON (1997), "um framework orientado a objetos é um projeto reusável como um todo ou em parte, sendo representado por um conjunto de classes abstratas e pela maneira que suas instâncias interagem". Assim, quando se utiliza um framework, está se reutilizando um projeto que trata de um domínio específico de problema e a nova aplicação projetada será iniciada a partir do projeto do framework. Isto traz uma maior facilidade no desenvolvimento de aplicações, pois o desenvolvedor deverá se preocupar somente com as características específicas – quando houverem – não contempladas no framework.

A metodologia de desenvolvimento de frameworks proposta pela empresa TALIGENT [1] (1995), busca construir um conjunto de frameworks estruturalmente menores e mais simples. Quanto à frameworks muito grandes, a TALIGENT (1995) propõe: "procure formas para quebrá-los em frameworks menores e focalizados. Se estiverem projetados para se interoperarem, frameworks menores podem ser reusados com mais freqüência e são mais flexíveis".

Dentro deste contexto, entende-se como justificável a construção de um framework orientado a objetos que forneça funcionalidades de segurança de dados em repouso, em aplicações novas e existentes. Este framework será utilizado como parte de uma aplicação – ou de outro framework – atendendo o domínio específico de segurança para dados em repouso.

1.5. Apresentação do Trabalho

No primeiro capítulo é apresentada uma introdução objetivando dar uma visão geral deste trabalho. A motivação mostra a importância de segurança através de um exemplo e a necessidade de facilitar a implementação de segurança para os dados em repouso. Os objetivos definem de forma clara quais os resultados almejados. Na

[1] A empresa Taligent hoje não existe mais, pois foi absorvida pela IBM.

justificativa são encontradas citações da literatura que justificam tanto a motivação quanto os objetivos.

O segundo capítulo apresenta as definições sobre frameworks orientados a objetos e a contribuição que os padrões de projetos trazem na construção do framework. Também são apresentadas as classificações de frameworks, fases de desenvolvimento e uso dos frameworks, benefícios de frameworks e os pontos fracos de frameworks.

No terceiro capítulo são explorados os conceitos fundamentais da criptografia. São apresentados os serviços de segurança, alguns problemas relacionados à segurança no armazenamento de dados e os fundamentos da criptografia de chave simétrica, criptografia de chave assimétrica, resumo de dados, assinatura digital, certificado digital e infra-estrutura de chaves públicas.

No quarto capítulo são apresentadas algumas APIs de criptografia mais conhecidas, que fornecem a implementação das funcionalidades de vários algoritmos de criptografia e também os bancos de dados mais conhecidos e como se utilizam da criptografia para reforçar a segurança de dados.

O framework orientado a objetos Frasedare é apresentado no quinto capítulo, no qual são descritas as fases de desenvolvimento, de uso e de evolução e manutenção. Na fase de desenvolvimento são descritas todas as atividades desde a análise de domínio, passando pelo projeto arquitetural, projeto do framework até a implementação e teste. Na fase de uso, é descrito como o framework Frasedare deve ser utilizado e também apresenta a constatação de sua aplicabilidade baseada em uma aplicação teste e em um sistema corporativo consolidado que fizeram uso de subsistemas construídos, estendendo o framework Frasedare. Por fim é descrita a fase de evolução e manutenção.

O sexto capítulo apresenta a conclusão, considerações e limitações sobre o trabalho desenvolvido, e também lança algumas questões para trabalhos futuros.

E o sétimo capítulo relaciona as referências bibliográficas utilizadas no desenvolvimento deste trabalho.

2. Frameworks Orientados a Objetos

O reuso de software tem sido um objetivo na engenharia de software desde o seu início. Em 1968, na Conferência de Trabalho sobre Engenharia de Software do Comitê de Ciência da OTAN[1], foram publicadas as primeiras idéias sobre o reuso de software. Segundo McIlroy, a indústria de software deveria produzir famílias de componentes reusáveis e o desenvolvedor de software poderia escolher os componentes que melhor se adaptassem às suas necessidades (MCILROY, 1968).

Durante a década de 1970, a programação modular foi definida e engenheiros de software entendiam que os módulos podiam ser utilizados como componentes reusáveis para o desenvolvimento de novas aplicações. Módulos, no entanto, somente proviam o seu reuso como um todo, e eram feitas as adaptações pela edição de código ou pela importação, alterando todos os aspectos inadequados para a nova aplicação. Na década de 1980, as linguagens orientadas a objeto cresciam em popularidade, já que seus proponentes afirmavam aumentar o reuso do código orientado a objetos através da herança. Diferentemente da importação ou edição do código, a herança fornece um meio mais poderoso de adaptar código.

O reuso em um nível individual, de pequena escala, é possível quando se utilizam componentes como blocos já construídos para desenvolvimento de novas aplicações. Todavia, o maior problema é o reuso de componentes em um nível mais abrangente, de larga escala, que podem compor a maior parte de uma aplicação, uma vez que neste caso, os aspectos que deveriam ser adaptados não foram tratados no paradigma orientado a objetos. Este entendimento conduz para o desenvolvimento de frameworks orientados a objetos, já que, de forma geral, um framework é uma grande estrutura, consistindo de uma aplicação abstrata em um domínio[2] particular que pode ser reusado e adaptado para a construção de novas aplicações específicas (MATTSSON, 2000).

---

[1] Organização do Tratado Atlântico Norte.

[2] Domínio refere-se a uma área de conhecimento ou atividade, caracterizada por um conjunto de conceitos e terminologias que são compreendidos pelos participantes desta área. (BOOCH, 1999)

Neste capítulo serão apresentados: definição de frameworks, utilização de padrões de projetos na construção, desenvolvimento e uso de frameworks, benefícios e pontos fracos na utilização de frameworks.

2.1. Definição

A definição de framework mais usada é: “um framework orientado a objetos é um projeto reusável como um todo ou em parte, sendo representado por um conjunto de classes abstratas e pela maneira que suas instâncias interagem” (JOHNSON, 1997). As definições de frameworks variam segundo JOHNSON (1997). Outra definição também comum, e porque não dizer, complementar à primeira é: “um framework é o esqueleto de uma aplicação que pode ser customizada por um desenvolvedor de aplicação”. A primeira descreve a estrutura de um framework, enquanto a segunda descreve seus propósitos.

Detalhando um pouco mais a definição de um framework, em termos de propósitos e estrutura, FAYAD (1999a) define que um projeto de framework proporciona as seguintes características:

- **Característica de Domínio** – São características relevantes de um domínio, úteis nas aplicações. A TALIGENT (1995) classifica os problemas de domínio em frameworks de aplicação, que encapsulam competências aplicáveis para uma grande variedade de programas; frameworks de domínio, que encapsulam competências em um problema particular de domínio; e frameworks de suporte, que fornecem serviços em nível de sistemas, como acesso a arquivo, suporte a computação distribuída ou *drivers* de dispositivos.
- **Característica Estrutural** – São características que facilitam a adaptação (uso) e evolução do framework. Estas características são especialmente importantes porque uma implementação e projeto de um framework são as interfaces reais usadas pelos usuários do framework. O uso (ou adaptação) pode ser dado por meio do reuso de caixa-branca, caixa-cinza e caixa-preta. O reuso de framework será tratado em uma próxima seção.

Frameworks orientados a objetos constituem uma tecnologia promissora para concretização de projetos de software testados e implementados, com objetivo de reduzir custos e melhorar a qualidade do software (JOHNSON & FOOTE, 1988).

Técnicas anteriores de reuso do paradigma OO[1], como biblioteca de classes, são construídas para serem utilizadas de forma bastante genérica, em quaisquer tipos de aplicação. Frameworks são construídos para atender um domínio específico de aplicação – como interfaces gráficas de usuário ou segurança – ou para determinada área de negócio – como comércio eletrônico ou aplicações corporativas.

A figura 2.1 ilustra a reutilização de um conjunto de classes inter-relacionadas de um framework em uma aplicação.

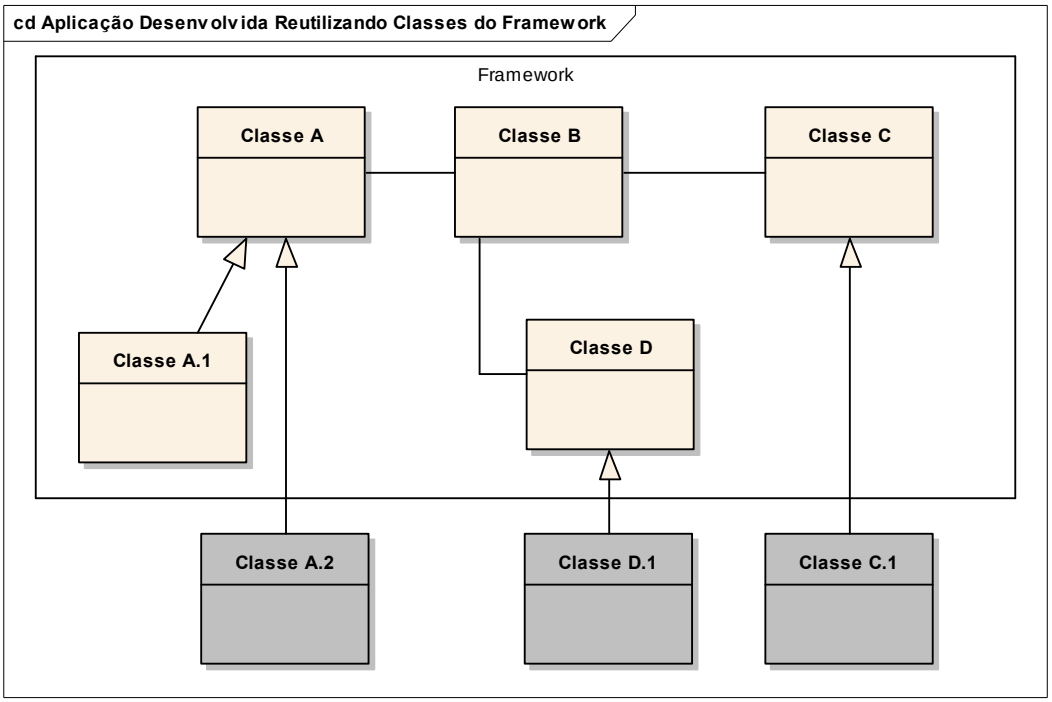

Figura 2.1 – Aplicação desenvolvida reutilizando classes do framework. Fonte Adaptada: (SILVA, 2000)

[1] Orientado a Objetos

Na figura 2.1, as classes que estão fora do quadro “Framework”, são classes estendidas das classes inter-relacionadas do framework para uma aplicação desenvolvida a partir do framework.

2.2. Frameworks e Padrões de Projeto

Padrões de projeto desempenham um papel importante na construção de frameworks por tratarem de pontos críticos como a flexibilização e documentação.

Sobre os pontos críticos, GAMMA (1995) afirma que:

> *“Um framework que os trata através do uso de padrões de projeto tem maior probabilidade de atingir altos níveis de reusabilidade de projeto e código, comparado com um que não usa padrões de projeto. Frameworks maduros comumente incorporam vários padrões de projeto. Os padrões ajudam a tornar a arquitetura do framework adequada a muitas aplicações diferentes, sem necessidade de reformulação”.*

Uma definição para padrões de projeto é: “cada padrão descreve um problema no nosso ambiente e o núcleo da sua solução, de tal forma que você pode utilizar esta solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira” (ALEXANDER, 1977 *apud* GAMMA, 1995) embora ALEXANDER (1977), um arquiteto, se referisse à construção civil, pode-se fazer uma analogia aos padrões de projeto orientados a objeto.

Em um sentido mais amplo, FOWLER (1997) diz: “Um padrão é uma idéia que foi útil em um contexto prático e provavelmente será útil em outros”.

De acordo com GAMMA (1995), em geral um padrão tem quatro elementos essenciais:

- **Nome do padrão** – Nome pelo qual o padrão é chamado. Com um nome bem definido, é mais fácil pensar sobre projetos e a comunicá-los.
- **Problema** – Descreve quando aplicar o padrão. Detalha o problema e o contexto onde o padrão poderá fornecer uma solução.

- **Solução** – Descreve os elementos que compõem o padrão, como a estrutura, seus relacionamentos, responsabilidades e colaborações. A solução não descreve um projeto concreto ou uma implementação específica porque um padrão é como um gabarito que pode ser aplicado em muitas situações diferentes. Assim, o padrão fornece uma descrição abstrata de um problema de projeto e como resolver este problema.
- **Conseqüências** – São os resultados e análise das vantagens e desvantagens da aplicação do padrão.

Padrões normalmente são documentados em um catálogo contendo os elementos acima descritos (GAMMA, 1995) (BUSHMANN, 1996) (ALUR, 2001). Com um catálogo, o desenvolvedor pode facilmente encontrar algum padrão que possa solucionar o seu problema dentro de um contexto específico.

Um exemplo de padrão de projeto muito utilizado na construção de frameworks é o padrão de projeto Método *Template (Template Method).* Segundo GAMMA (1995), a intenção deste padrão de projeto é "definir o esqueleto de um algoritmo em uma operação, postergando alguns passos para subclasses. Método *Template* permite que subclasses redefinam certos passos de um algoritmo sem mudar a estrutura do mesmo".

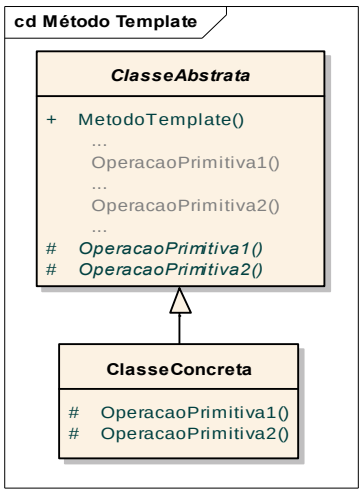

Figura 2.2 – Padrão de projeto Método *Template.*

A figura 2.2 apresenta um diagrama de classes com a estrutura do padrão de projeto Método *Template.* Este padrão consiste basicamente de uma classe abstrata que possui pelo menos uma operação implementada por um método que invoca operações abstratas. Estas operações abstratas, conhecidas como *Hook,* deverão ser implementadas por métodos nas subclasses.

Sobre padrões de projeto e frameworks, FAYAD & SCHMIDT (1997) afirmam que:

> *"Padrões e frameworks juntos facilitam o reuso através da captura das estratégias de sucesso no desenvolvimento de software. A principal diferença entre eles é que frameworks focam o reuso de um projeto concreto, algoritmos e implementações em uma linguagem de programação específica, enquanto os padrões estão focados no reuso de projetos abstratos e micro-arquiteturas de software."*

GAMMA (1995) afirma que padrões de projeto e frameworks são diferentes em três aspectos:

- Padrões de projeto são mais abstratos que frameworks – os frameworks são concretizados através de sua implementação (código fonte), já com os padrões de projeto, somente os exemplos podem ser concretizados através do código;
- Padrões de projeto são elementos de arquitetura menores que frameworks – um framework pode conter vários padrões de projeto, mas nunca um padrão conterá um framework;
- Padrões de projeto são menos especializados que frameworks – frameworks sempre têm um domínio específico de aplicação. Já os padrões de projeto podem ser utilizados em qualquer tipo de aplicação.

Segundo FAYAD (1999a), se além de padrões de projeto e frameworks, também forem utilizadas bibliotecas de classes e componentes, o software terá um ganho de qualidade e redução do esforço de desenvolvimento.

2.3. Classificação de Frameworks

Os frameworks podem ser classificados de acordo com seu escopo e pelas técnicas utilizadas para estendê-los.

Segundo FAYAD (1999a), embora os benefícios e princípios de projeto sobre frameworks sejam em grande parte, independentes do domínio para o qual eles são aplicados, é importante classificar frameworks por seu escopo. A TALIGENT (1995) classifica frameworks nos seguintes escopos:

- **Frameworks de Aplicação** – Frameworks de aplicação fornecem funcionalidades em grande escala, tipicamente necessárias em uma aplicação. Estas funcionalidades normalmente envolvem questões como uma interface gráfica com o usuário, documentos, banco de dados, etc. Um exemplo de um framework de aplicação é o MFC (Microsoft *Foundation Classes*). O MFC é usado para construir aplicações compatíveis com o sistema operacional MS Windows. Um outro exemplo de framework de aplicação é o JFC (*Java Foundation Classes*). Este último é interessante do ponto de vista de projeto orientado a objetos, pois incorpora muitas idéias sobre como um framework orientado a objetos deve ser construído. Muitos padrões de projeto de GAMMA (1995) são utilizados neste framework (GURP & BOSCH, 2001).

- **Frameworks de Subsistema ou Frameworks de Domínio** – Frameworks de subsistema ou de domínio são úteis para implementar programas de um determinado domínio específico. Eles encapsulam a solução de um problema de domínio, sendo considerado como uma parte vertical do domínio do cliente. Exemplos de frameworks de subsistema ou de domínio pode ser um framework de segurança para controle de acesso de aplicações de manufatura ou um framework de para acesso a dados. Frameworks de subsistema ou de domínio ajudam a reduzir o tempo e custos necessários para implementar estas aplicações.

- **Frameworks de Suporte** – Frameworks de suporte tipicamente destinam-se a domínios muito específicos relacionados ao computador, como

gerenciamento de memória ou sistema de arquivos. Dar suporte a estes tipos de recursos simplifica o desenvolvimento de programas. Frameworks de suporte são tipicamente usados em conjunto com frameworks de domínio ou aplicação.

FAYAD (1999a) ressalta que, independente do escopo, frameworks também podem ser classificados pelas técnicas utilizadas para estendê-los (uso do framework), que variam em uma série contínua, de frameworks caixa-branca para frameworks caixa-cinza e para frameworks caixa-preta.

- **Caixa-branca** – As funcionalidades existentes são reutilizadas e estendidas por herança de classes bases do framework e sobrescrevendo ou implementando métodos para operações *Hook* usando padrões de projeto igual a métodos *Template*.
- **Caixa-preta** – Estes frameworks proporcionam a extensibilidade pela definição de interfaces para componentes que podem ser conectados em um framework através de composição de objeto. As funcionalidades existentes são reutilizadas pela definição dos componentes que irão se adequar a uma determinada interface e integrar estes componentes dentro de frameworks.
- **Caixa-cinza** – Utiliza uma combinação das técnicas de caixa-branca e caixa-preta. Assim, como certas funcionalidades de um framework caixa-cinza podem ser estendidas por herança e outras por composição, o framework tem suficiente flexibilidade e extensibilidade, e também possui a capacidade de esconder informações desnecessárias de um desenvolvedor de aplicação.

A figura 2.3 ilustra a classificação das técnicas utilizadas para estender frameworks.

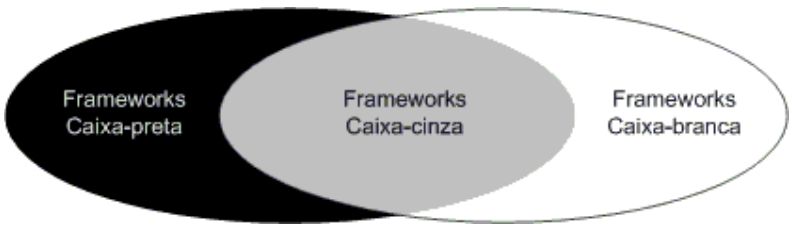

Figura 2.3 – Classificação de frameworks de acordo com a técnica de extensão. Fonte: (FAYAD, 1999b)

2.4. Desenvolvimento e Uso de Frameworks

De acordo com FAYAD (1999a), o desenvolvimento de software centrado em frameworks possui as seguintes fases: desenvolvimento do framework; uso do framework e, por último, a evolução e manutenção do framework. Além das fases, papéis também são definidos para especificar as funções dentro do desenvolvimento e uso.

2.4.1. Atores Participantes

Em um desenvolvimento tradicional existem dois atores: o desenvolvedor de aplicação e o usuário da aplicação. O desenvolvedor faz levantamento de requisitos de uma aplicação, desenvolve e a entrega ao usuário. Usuários interagem com uma aplicação através da interface da aplicação (FAYAD, 1999a) (SILVA, 2000).

A figura 2.4 apresenta os elementos do desenvolvimento tradicional de aplicação.

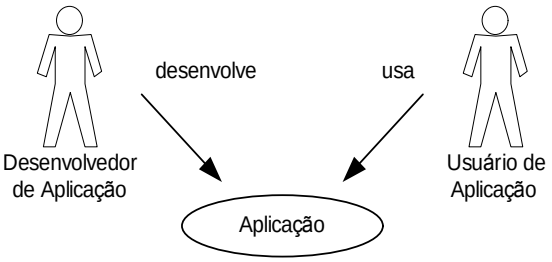

Figura 2.4 – Elementos do desenvolvimento tradicional de aplicação.
Fonte: (SILVA, 2000).

No desenvolvimento com a utilização do framework, além dos atores, desenvolvedor e usuário de aplicação, existe mais um ator, o desenvolvedor de framework. Neste contexto, o papel do usuário de aplicação é o mesmo, como descrito anteriormente. Já o papel do desenvolvedor de aplicações é diferente do caso anterior, justamente pela inserção do framework. Assim, o desenvolvedor de aplicações é um usuário de um framework, devendo estender e adaptar a estrutura deste framework para

o desenvolvimento de aplicações. Este papel possui as mesmas funções anteriormente citadas, ou seja, obter os requisitos para a aplicação, desenvolvê-la usando o framework e elaborar a sua documentação. O novo papel criado, o desenvolvedor de framework, tem a responsabilidade de produzir frameworks e encontrar algum modo de ensinar como utilizar o framework para desenvolver aplicações (SILVA, 2000).

A figura 2.5 apresenta os elementos do desenvolvimento de aplicações baseado em frameworks, incluindo um novo papel, o desenvolvedor de framework.

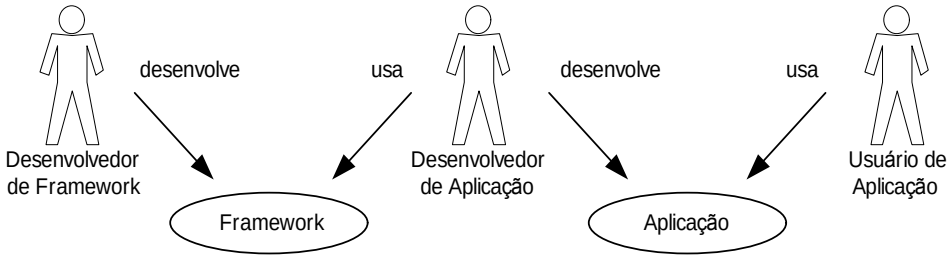

Figura 2.5 – Elementos do desenvolvimento de aplicações baseado em frameworks.
Fonte: (SILVA, 2000)

2.4.2. Fase de Desenvolvimento do Framework

Esta fase tem o objetivo de produzir um projeto reusável em um domínio específico. Freqüentemente, é a fase que mais esforço requer. Segundo FAYAD (1999a), o desenvolvimento de um framework é um tanto diferente do desenvolvimento de uma aplicação padrão. Uma importante diferença no desenvolvimento de framework está em abranger todos os conceitos relevantes em um domínio, enquanto uma aplicação está interessada somente em todos os conceitos mencionados nos requisitos da aplicação. De modo geral, um modelo de desenvolvimento de framework possui as seguintes atividades:

- **Análise do Domínio** – Tem como objetivo descrever o domínio que deverá ser atendido pelo framework. O desenvolvedor deve levantar os requisitos e identificar conceitos em aplicações desenvolvidas no domínio, com o especialista no domínio e padrões existentes no domínio. O resultado desta atividade é um modelo de análise do domínio, contendo requisitos do

domínio, conceitos do domínio e a relação entre estes conceitos (SCHÄFER, 1994 *apud* FAYAD 1999a).

- **Projeto Arquitetural** – Através do modelo de análise do domínio, o projetista precisa definir um estilo de arquitetura adequado para formar a base do framework. Segundo SHAW (1996), estilo de arquitetura define um vocabulário de tipos de elementos e conectores juntamente com as restrições de como eles podem ser combinados. Exemplos de elementos são servidores, camadas e banco de dados; já exemplos de conectores são protocolos de banco de dados, chamadas a procedimento e eventos de *broadcast*. Definido o estilo da arquitetura, entre outras considerações, o primeiro nível do framework é projetado.
- **Projeto do Framework** – Nesta atividade, o primeiro nível do projeto do framework é melhorado e classes adicionais são projetadas. Os resultados são as funcionalidades fornecidas pelo escopo do projeto do framework, a interface reusável do framework, a obediência das regras do projeto baseado nas decisões arquiteturais, e um documento do histórico do projeto descrevendo os problemas encontrados no projeto e as soluções utilizadas, com um argumento.

  A abordagem de frameworks insere um conjunto de requisitos de modelagem em geral não atendido por metodologias de Análise e Projeto Orientado a Objetos. Assim, SILVA (2000) sugere a classificação de redefinibilidade e essencialidade das classes do framework conforme descrito abaixo:

  - **Redefinibilidade de Classe** – Indica se a classe no projeto de um framework pode ou não originar subclasses no desenvolvimento de uma aplicação que utiliza o framework.
  - **Essencialidade da Classe** - Identifica se a classe é essencial ou não no framework. Quando uma classe de um framework for essencial, determinará que toda a aplicação desenvolvida utilizando o framework deverá utilizar esta classe, ou uma subclasse dela.

Além das classificações de redefinibilidade e essencialidade, as classes e suas operações também possuem a seguinte classificação:

- **Classe** – Além das propriedades de redefinibilidade e essencialidade, as classes também são classificadas como concretas ou abstratas. Classes abstratas são identificadas com o nome em itálico nos diagramas da UML;
- **Operações** – As operações podem ser classificadas como regulares, ou abstratas. As operações abstratas são identificadas com o nome em itálico nos diagramas da UML.

- **Implementação do Framework** – Efetua a codificação das classes abstratas e concretas do framework em uma linguagem de programação.
- **Teste do Framework** – Avalia a usabilidade do framework e determina se o framework proporciona as pretendidas funcionalidades. Para avaliar a usabilidade do framework, são desenvolvidas aplicações baseadas no framework. Baseando-se nas aplicações desenvolvidas decide-se se o framework precisa ser re-projetado ou se está suficientemente maduro.
- **Documentação** – No desenvolvimento de framework, a documentação é uma das mais importantes atividades. Ela precisa ser clara, completa e correta descrevendo como usar o framework como um manual de usuário, e como o projeto do framework trabalha.

SILVA (2000) descreve três metodologias voltadas ao desenvolvimento de frameworks. Estas metodologias buscam dar uma visão geral do processo de desenvolvimento:

- **Projeto Dirigido por Exemplo** – JOHNSON (1993) propõe que o desenvolvimento de um framework seja projetado observando-se aplicações de um domínio específico. Os aspectos semelhantes de diferentes aplicações podem dar origem a classes abstratas, sendo que as particularidades podem ser implementadas em classes concretas no desenvolvimento de novas aplicações utilizando o framework.

- **Projeto Dirigido por *Hot Spot*** – PREE (1995) propõe esta metodologia, que consiste em construir frameworks identificando as partes flexíveis na estrutura de classes de um domínio. *Hot spots* são as partes mantidas flexíveis do framework, ou seja, as partes que são diferentes entre as aplicações. Segundo (FAYAD, 1999a), as etapas básicas desta metodologia são: Definição de um modelo específico de objetos, identificação de *hot spots*, (re)projeto do framework e a adaptação através do uso do framework. Caso se constatar a necessidade de uma maior flexibilidade, deve-se retornar à etapa de identificação de *hot spots*. A flexibilidade do framework é obtida através da aplicação de padrões de projeto na etapa de projeto. O framework é considerado concluído quando se constata que chegou a um alto grau de flexibilidade.
- **Metodologia de Projeto da Empresa Taligent** – Busca construir um conjunto de frameworks estruturalmente menores e mais simples. Quanto a frameworks muito grandes, segundo TALIGENT (1995), "procure formas para quebrá-los em frameworks menores e focalizados. Se estiverem projetados para se interoperarem, frameworks menores podem ser reusados com mais freqüência e são mais flexíveis". Tornar o uso do framework o mais simples possível é um objetivo alcançado através da minimização da quantidade de código produzida pelo desenvolvedor da aplicação por meio da disponibilidade de implementações de classes concretas que possam ser usadas diretamente, minimizando o número de classes que devem ser criadas e o número de métodos que devem ser sobrepostos. Quanto ao processo de desenvolvimento, este é dividido em quatro etapas: identificação e caracterização do domínio do problema, definição da arquitetura e o projeto, implementação do framework e desdobramento do framework.

Frameworks são complexos por natureza, conseqüentemente, um dos maiores problemas está em como aprender a utilizá-los. Desenvolvedores de frameworks precisam documentar bem seus frameworks e desenvolver um bom material de treinamento para eles. Na seção seguinte, será detalhada a necessidade de documentação para frameworks.

2.4.3. Fase de Uso do Framework

O principal resultado desta fase é uma aplicação (ou subsistema) desenvolvida reutilizando-se um ou mais frameworks. O desenvolvedor deve incluir o projeto do framework ou parte dele, dependendo dos requisitos da aplicação. Assim, o projeto da aplicação começa a partir do projeto do framework, pois ele força a aplicação a reusar seu projeto. Depois que o projeto da aplicação é definido, o projetista precisa decidir quais incrementos internos deve incluir. Para todas as partes do projeto da aplicação não atendidas por classes reusáveis, novas classes devem ser desenvolvidas para atender os requisitos atuais.

A forma de se usar um framework é definida dentro da classificação de frameworks caixa-branca, caixa-preta e caixa-cinza, como definido na Seção 2.3. A reutilização do framework de caixa-preta é feita através da conexão de componentes sem precisar ter a visão da implementação. Já a reutilização do framework caixa-branca é feita por herança e normalmente requer mais conhecimento por parte dos desenvolvedores. Frameworks caixa-preta são mais fáceis de se aprender a usar, mas frameworks caixa-branca são freqüentemente mais poderosos nas mãos de especialistas. Frameworks que mesclam características caixa-preta e caixa-branca são normalmente chamados de frameworks caixa-cinza.

Conforme mencionado na seção anterior, para aprender a usar um framework, existe a necessidade de uma boa documentação. Os frameworks simples são mais fáceis de serem usados se um bom material de treinamento estiver disponível para o desenvolvedor, porém os frameworks complexos requerem treinamentos por alguém que conheça o framework. Segundo FAYAD (1999a), o melhor caminho para se começar a aprender um framework é através de exemplos. De forma ideal, um framework deveria vir com um conjunto de exemplos – de básicos até avançados – que implementariam a maioria das características do framework. FAYAD (1999a) ainda ressalta que idealmente um framework deve ter um conjunto completo de documentação que explore: o propósito do framework, como usar o framework e como o framework trabalha.

Uma boa documentação pode ser um tipo de *cookbook*[1]. Desenvolvedores iniciantes podem usar o *cookbook* para fazer suas primeiras aplicações e desenvolvedores mais avançados podem utilizá-lo na procura de soluções para problemas particulares.

Usuários de frameworks precisam planejar no orçamento o tempo necessário dedicado à aprendizagem do framework.

2.4.4. Fase de Evolução e Manutenção do Framework

Como todo software está sujeito a mudanças, frameworks também não fogem à regra. As mudanças podem vir de erros lançados pelas aplicações, de identificação de novas abstrações devido à alterações no domínio do problema, de alterações no domínio do negócio, e assim por diante (FAYAD, 1999a).

No desenvolvimento de frameworks, desde o início são freqüentemente necessárias várias iterações no projeto. Uma importante razão para iteração é que um framework é supostamente reusável, e o único caminho para fornecer este reuso, é reusar o framework e identificar suas deficiências (JOHNSON & RUSSO, 1991 *apud* FAYAD, 1999a). Assim, esta fase de evolução e manutenção do framework incorpora as duas primeiras fases, de desenvolvimento e de uso do framework.

As mudanças em um framework não são muito simples. Se estas mudanças ocorrem enquanto a aplicação está sendo desenvolvida, não há grandes problemas, pois poderão ser tratadas normalmente durante o desenvolvimento da aplicação. Porém, mudanças como alteração de interfaces, novas classes e novas funcionalidades gerarão problemas que deverão ser tratados para aplicações já desenvolvidas com o framework.

À medida que mudanças ocorrem no desenvolvimento de frameworks, também a evolução acontece. A evolução de um framework começa com o framework de caixa-branca, onde o reuso é feito através de subclasses, e conforme a evolução ocorre, ele tende a ser um framework de caixa-preta, onde o reuso é feito através de composição e parametrização (FAYAD, 1999a). Finalmente, a evolução leva à solidificação das

[1] *Cookbook* traduzindo do inglês significa livro de receitas.

diferenças entre a parte adaptável (*hot spots*) da estável (*frozen spots*), e entre o genérico do específico.

2.5. Benefícios de Frameworks

De acordo com FAYAD (1999a), os principais benefícios dos frameworks de aplicação orientados a objetos decorrem da sua modularidade, reusabilidade, extensibilidade e inversão de controle que provêem para os desenvolvedores, como descritos a seguir:

- **Modularidade** – Frameworks aumentam a modularidade por encapsular os detalhes de implementação volátil dentro de interfaces estáveis, melhorando a qualidade do software por identificar o impacto das mudanças do projeto e implementação.
- **Reusabilidade** – As interfaces estáveis fornecidas pelo framework facilitam a reusabilidade por definir componentes genéricos que podem ser utilizados em novas aplicações.
- **Capacidade de Extensão** – Um framework aumenta a capacidade de extensão pelo fornecimento explícito de operações *Hook* (PREE, 1995 *apud* FAYAD, 1999a), que permitem às aplicações estender suas interfaces. Frameworks com capacidade de extensão são essenciais para garantir customizações em uma nova aplicação.
- **Inversão de Controle** – A inversão de controle em tempo de execução também é referenciada como Princípio de *Hollywood*: "Não nos chame, nós chamaremos você". No desenvolvimento tradicional de aplicações, o desenvolvedor faz chamadas à biblioteca de classes e controla o fluxo de execução. Já no desenvolvimento de aplicações utilizando framework, o controle do fluxo de execução e as chamadas à biblioteca de classes, são providos pelo framework. Dessa forma, o desenvolvedor pode se concentrar somente em implementar as funcionalidades específicas, sem se preocupar com a invocação das operações.

2.6. Pontos Fracos de Frameworks

Segundo FAYAD (1999a), mesmo frameworks proporcionando vários benefícios, é importante salientar alguns pontos fracos:

- **Esforço de desenvolvimento** – Enquanto que desenvolver software complexo já é bastante difícil, desenvolver frameworks de alta qualidade, extensíveis e reusáveis para domínios de aplicações complexas é ainda mais difícil. O conhecimento requerido para o sucesso na construção de frameworks geralmente fica restrito aos desenvolvedores experientes.
- **Curva de aprendizado** – Aprender a usar frameworks de aplicação orientados a objetos requer um considerável esforço de investimento. O usuário do framework pode aprender a usar o framework através da documentação do framework e de exemplos de aplicações desenvolvidas utilizando o framework. Porém, em frameworks mais complexos, mentores e cursos de treinamentos são necessários para se ensinar aos usuários do framework como utilizá-lo de forma eficiente.
- **Integração** – O desenvolvimento de aplicação pode estar baseado na integração de múltiplos frameworks, junto com bibliotecas de classes, sistemas legados e componentes existentes. A integração pode trazer alguns problemas como o de concorrência. Por exemplo, na inversão de controle, que é uma característica essencial do framework, será difícil integrar frameworks cujos eventos de *loops* não foram projetados para interoperar com outros frameworks.
- **Manutenção** – Os requisitos de aplicações mudam freqüentemente. Portanto, os requisitos de frameworks também mudam com freqüência. Como frameworks invariavelmente mudam, as aplicações que os usam devem mudar também. As atividades de manutenção de frameworks incluem a modificação e a adaptação, ambas podendo ser tanto a nível funcional como não funcional. Um profundo entendimento dos componentes do frameworks e seus inter-relacionamentos são fundamentais para fazer a execução com sucesso desta tarefa.

- **Validação e remoção de defeitos** – Embora frameworks bem projetados e modularizados possam minimizar o impacto dos defeitos das aplicações, a validação e depuração das aplicações construídas usando-se o framework podem ser complicadas. Componentes genéricos são tipicamente abstratos, afastando os detalhes de uma aplicação específica. Os detalhes específicos são providos por meio de herança, composição de objetos ou parametrização. Ainda que isto melhore a flexibilidade e extensibilidade do framework, dificulta muito o teste do módulo, visto não ser possível a validação isolada de suas instâncias específicas. A inversão de controle provida pelos frameworks também pode dificultar a depuração, oscilando entre a infra-estrutura do framework e as invocações feitas por ele às operações da aplicação específica.
- **Falta de padronização** – Atualmente, não existe nenhuma padronização amplamente aceita para projeto, implementação, documentação e adaptação de frameworks.

2.7. Considerações

Apesar de a construção de um bom framework ser dispendiosa em relação a tempo e dinheiro, pode-se reduzir o custo no desenvolvimento de várias aplicações ao se reutilizar o projeto e implementação fornecidos pelo framework. Um framework deve incorporar abstrações e funcionalidades de um domínio de problema específico, objetivando tornar o desenvolvimento de aplicações mais rápidas por reutilizar o projeto e a implementação e, flexível por fornecer pontos adaptáveis que podem ser configurados ou estendidos.

3. Fundamentos da Criptografia

O objetivo principal da criptografia é tornar os dados sigilosos a um adversário que possa vir a interceptá-los. A palavra criptografia é formada por duas palavras gregas: *kryptós* (oculto, secreto) e *grapho* (escrita, escrever) (CARVALHO, 2001).

Cifragem é o processo pelo qual a criptografia torna dados incompreensíveis, isto é, codificados ou secretos, normalmente chamados de *texto cifrado*. Já o processo inverso é a decifragem, em que dados são decodificados para possibilitar sua leitura e normalmente chamados de *texto claro* (CARVALHO, 2001).

Alguns processos de criptografia geram textos cifrados em que não há um processo inverso. Estes processos, denominados de *mão única* são conhecidos como *função para resumo de dados*[1] ou *função hash*. Em outros processos, para cifrar e decifrar os dados se utiliza uma chave, da qual a segurança depende em grande parte. O número de bits de uma chave é um parâmetro fundamental no sistema de segurança, pois quanto maior for o número de bits da chave, maior é o número de possibilidades de chaves que podem ser utilizadas, assim dificultando que um ataque de força bruta[2] descubra a chave.

Há aproximadamente 4000 anos os egípcios já utilizavam a criptografia, o que mostra que ela já era conhecida desde os primórdios da escrita. Contudo, até alguns séculos atrás, aqueles envolvidos no seu uso eram predominantemente profissionais das áreas militares, serviços diplomáticos e governos, sendo utilizada como ferramenta para proteção de segredos e estratégias nacionais.

Com a proliferação dos computadores e sistemas de comunicação nos anos de 1960, surgiu no setor privado uma forte demanda por meios de proteção dos dados na forma digital e por serviços de segurança. Com o trabalho de Feistel, na IBM, no início da década de 1970, surge o mecanismo de criptografia mais conhecido da história, o DES (*Data Encryption Stantard*).

---

[1] Também conhecido como resumo de mensagem.

[2] Ataque de força bruta é a tentativa de todas as possíveis chaves até que a correta seja identificada.

Uma grande evolução da criptografia ocorreu em 1976 com a introdução do conceito de criptografia de chave pública por Diffe e Helman, proporcionando um novo e engenhoso método para troca de chaves (DIFFE & HELLMAN, 1976). Baseados nestas idéias, em 1978 Rivest, Shamir e Adleman inventaram o primeiro esquema prático para criptografia de chave pública e assinatura digital (RIVEST, 1978), que ficou conhecido como RSA (MENEZES, 1997).

Deste então, a criptografia através de chave vem sendo classificada em duas categorias: criptografia de chave simétrica e criptografia de chave assimétrica. A primeira caracterizada por utilizar a mesma chave para cifrar e decifrar, e a segunda por utilizar o conceito de chave pública, onde existe um par de chaves, a privada e a pública, sendo que enquanto uma cifra a outra decifra.

Neste capítulo serão apresentados os serviços de segurança de dados, considerações sobre segurança no armazenamento dos dados, criptografia de chave simétrica, criptografia de chave assimétrica, função para resumo de dados e assinatura digital e uma relação entre os mecanismos de criptografia e os serviços de segurança de dados.

3.1. Serviços de Segurança de Dados

Serviços de segurança são usados para garantir objetivos da segurança de dados (MAIWALD, 2001). Um serviço de segurança é caracterizado por um conjunto de mecanismos, procedimentos e outros controles, atendendo a um determinado objetivo de segurança. MENEZES (1997) e BURNETT & PAINE (2002), enfatizam os seguintes serviços de segurança:

- **Sigilo** – Utilizado para manter o conteúdo dos dados conhecidos somente por aqueles que são autorizados. É sinônimo de confidencialidade e privacidade.
- **Integridade** – É um mecanismo que identifica a ocorrência de uma manipulação de dados não autorizada. A manipulação de dados inclui formas como inclusão, exclusão e alteração.
- **Autenticação** – Está relacionada com a identificação e muitas vezes dividida em categorias distintas: autenticação de entidade, ou seja, se duas entidades

estão se comunicando, cada uma pode ser identificada; e a autenticação da origem dos dados, que proporciona implicitamente a integridade dos dados, por exemplo, através de um resumo de dados.

- **Não-repúdio** – Permite que uma determinada entidade não negue um compromisso ou ação já efetuada por ela. Em outras palavras, é uma imposição legal que orienta e impele as pessoas a honrar suas palavras.

Além dos serviços acima citados, a recomendação X.800 da ITU-T [1] (*International Telecommunications Union*) acrescenta o serviço de controle de acesso. Este serviço define a prevenção do uso desautorizado de algum recurso, controlando quem pode acessar o recurso, sob que condições o acesso pode ocorrer e o que aqueles que acessam o recurso têm permissão para fazer (STALLINGS, 2003).

O objetivo fundamental da criptografia é tratar suficientemente, tanto na teoria quanto na prática, dos quatro tópicos acima citados. De uma forma geral, a criptografia trata da prevenção, da detecção de roubo e de outras atividades maliciosas relacionadas com os dados, isto é, trata da segurança de dados.

3.2. Segurança no Armazenamento de Dados

Tanto na literatura, MENEZES (1997), SCHNEIER (1996), STALLINGS (2003), TANENBAUM (2003), bem como em entidades do setor privado e público, existe uma grande preocupação em utilizar a criptografia para os dados em trânsito, com ênfase na comunicação de dados. Apesar das empresas que fornecem bens e serviços de segurança fazerem um bom trabalho em proteger os dados em trânsito de seus clientes, também os dados em repouso (armazenados) requerem uma atenção para proteção (BURNETT & PAINE, 2002).

Dados em repouso podem estar armazenados em arquivos, tabelas ou objetos de banco de dados, enfim, dados registrados de alguma forma em um determinado lugar por algum tempo. Muitas vezes, estes dados necessitam de proteção, e esta proteção pode ser atendida com um ou mais dos serviços de segurança, como citado na seção 3.1.

[1] ITU-T é o setor para padronização de telecomunicações do ITU.

3.2.1. Problemas no Armazenamento de Dados

Segundo SCHNEIER (1996), existem alguns problemas ao se armazenar dados com segurança:

- Os dados podem também existir na forma de texto claro, em algum outro disco, em outro computador ou em papel. Existem assim, muito mais oportunidades para um criptoanalista[1] fazer um ataque de texto claro.
- Em aplicações de banco de dados, o tamanho dos blocos de dados, como coluna de tabelas em um banco de dados relacional, pode ser menor que o tamanho do bloco de dados gerados por muitos algoritmos. Como resultado, pode-se ter um texto cifrado consideravelmente maior que o texto claro.
- A velocidade dos dispositivos de E/S (Entrada e Saída) demanda rapidez ao cifrar e decifrar, e provavelmente requer cifragem através de hardware. Em algumas aplicações, algoritmos especiais de alta velocidade podem ser requeridos.
- Segurança no armazenamento da chave em longo prazo é requerida.
- Gerenciamento de chave é mais complicado, visto que diferentes pessoas precisam acessar diferentes arquivos, diferentes porções do mesmo arquivo e assim por diante.

3.3. Criptografia de Chave Simétrica

A criptografia de chave simétrica utiliza a mesma chave para cifrar e decifrar dados. Isso significa que a chave deve ser de conhecimento tanto de quem cifra os dados como de quem necessita conhecer estes dados.

A criptografia de chave simétrica é utilizada para manter o sigilo ou a confidencialidade dos dados. Em um cifrador simétrico, normalmente o algoritmo para cifrar e decifrar os dados é basicamente o mesmo, mudando apenas a forma de como a chave é utilizada (STALLINGS, 2003).

---

[1] Pessoa dedicada a decifrar mensagens ou quebrar a criptografia de mensagens cifradas.

Existem dois tipos de algoritmos de chave simétrica, como descrito abaixo (BURNETT & PAINE, 2002), (STALLINGS, 2003):

- **Cifragem de bloco** – Opera sobre blocos de dados. Quando dados são enviados como entrada para o algoritmo cifrar ou decifrar, ele divide estes dados em blocos e opera sobre cada bloco de modo independente. Exemplos de cifradores de bloco são o Blowfish, DES, 3DES, IDEA, RC5 e AES (*Advanced Encryption Standard*).

- **Cifragem de fluxo** – Para cifrar os dados, o algoritmo gera um enchimento[1] com base na chave. Este enchimento pode ser tão grande quanto necessário. Depois, o algoritmo faz um XOR em um bit ou byte de cada vez entre o enchimento e o texto claro para cifrar, ou entre o enchimento e o texto cifrado, para decifrar. Exemplos de cifradores de fluxo são o Vigenère, Vernam, *On-time pad*[2] e RC4.

O principal problema relacionado à criptografia de chave simétrica está no fato de que as partes devem ter acesso à mesma chave. Desta forma, há a necessidade da adoção de uma política de segurança para a troca e guarda de chave. Porém, esta política acarreta dois problemas quanto à segurança, decorrentes do gerenciamento de chaves. O primeiro diz respeito à conservação do segredo de uma chave que é de conhecimento de várias pessoas, uma vez que bastaria uma delas agir de forma mal intencionada para que todos sofressem as eventuais conseqüências. O segundo problema refere-se à própria distribuição da chave, pois sempre que novas pessoas fossem admitidas no grupo, mais pessoas compartilhariam esta chave.

A figura 3.1 mostra o processo de cifragem dos dados e o processo inverso, a decifragem dos dados cifrados, utilizando para ambos os casos a mesma chave.

---

[1] Enchimento também é conhecido como fluxo de chave.

[2] O termo *On-time pad*, segundo CARVALHO (2001), é utilizado por já estar estabelecido. BURNETT & PAINE (2002), traduzem este termo como “enchimento de uma única vez”.

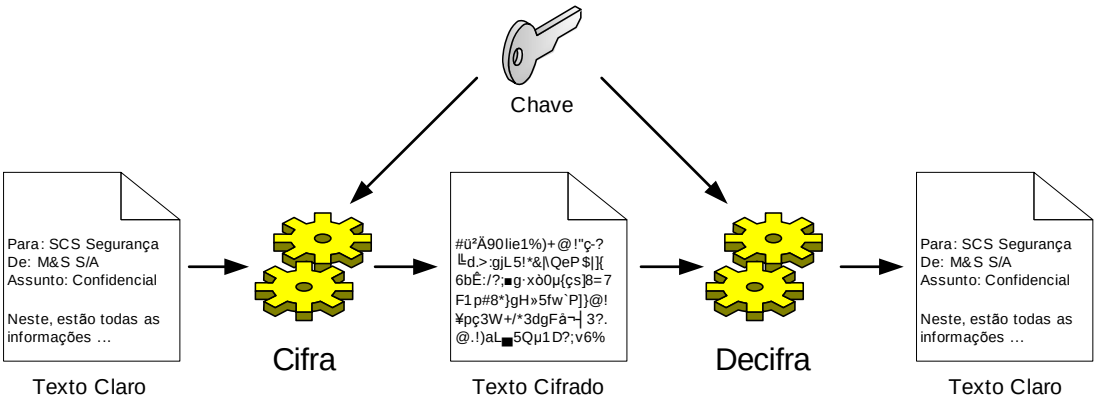

Figura 3.1 – Processo de cifragem e decifragem na criptografia de chave simétrica.

3.4. Criptografia de Chave Assimétrica

O mecanismo de criptografia de chave assimétrica, também chamado de criptografia de chave pública, é aquele em que cada usuário possui um par de chaves: uma chave pública e uma chave privada. Qualquer uma das chaves pode ser usada para cifrar ou decifrar. Se a chave privada é usada para cifrar, então se deve utilizar a chave pública para decifrar. Se a chave pública é usada para cifrar, deve-se utilizar a chave privada para decifrar. A chave privada deve ser mantida em segredo, enquanto que a chave pública deve ser tornada, de alguma forma, pública (STALLINGS, 2003).

Alguns dos mais conhecidos algoritmos de chave assimétrica são o DH (*Diffe-Hellman*), ECC (*Elliptic Curve Cryptography*) e o RSA. O DH não é utilizado para criptografia, e neste algoritmo cada parte possui um valor secreto e outro valor público. Se for combinado o valor privado com o outro valor público, cada parte gerará o mesmo valor secreto. A segurança do algoritmo ECC está baseada na dificuldade de resolução do problema do logaritmo discreto, já a segurança do RSA está baseada na dificuldade de se resolver o problema da fatoração. Esta seção está centrada nas propriedades do algoritmo RSA.

As chaves públicas e privadas do RSA têm as seguintes propriedades (STALLINGS, 2003):

- Diferentemente da criptografia de chave simétrica, na qual a chave é única, existem aqui, duas chaves[1];
- Cada chave pode ser utilizada para cifrar ou decifrar;
- Dados cifrados com uma das chaves somente podem ser decifrados com a outra chave;
- O conhecimento da chave pública não permite a descoberta da chave privada correspondente.

A figura 3.2 apresenta o processo de cifragem e decifragem utilizando a criptografia de chave assimétrica com base no algoritmo RSA. Quando uma chave é utilizada para cifrar, somente a outra decifra.

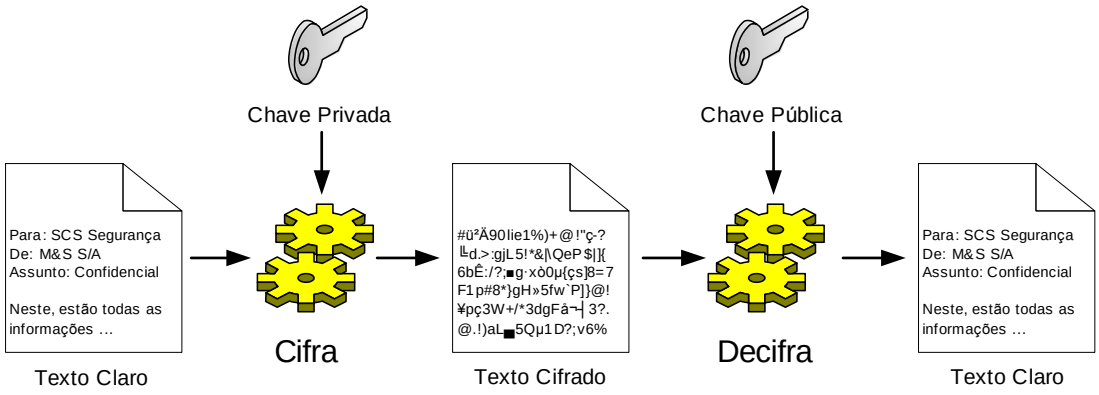

Figura 3.2 – Processo de cifragem e decifragem utilizando a criptografia de chave assimétrica com base no algoritmo RSA.

O desempenho é um problema ao se utilizar criptografia de chave assimétrica para cifrar os dados. Algoritmos de chave pública são lentos, ao passo que a criptografia de chave simétrica pode cifrar os dados em grande quantidade de forma bem rápida. Por este motivo, em um processo chamado de envelope digital, se utiliza a criptografia de chave simétrica para cifrar os dados através de uma chave gerada, chamada de chave de

[1] Atualmente, com as assinaturas múltiplas, podem existir uma ou mais chaves privadas e uma chave pública (SCHNEIER, 1996)

sessão. Depois, a chave de sessão é cifrada com a chave pública, assim fechando o envelope digital.

Para decifrar os dados cifrados com a criptografia de chave simétrica e a chave de sessão gerada, primeiramente é decifrada a chave de sessão, utilizando a chave privada que corresponde à chave pública, utilizada antes para cifrar a chave de sessão. E por fim, utiliza-se a chave de sessão para decifrar os dados com a criptografia de chave simétrica.

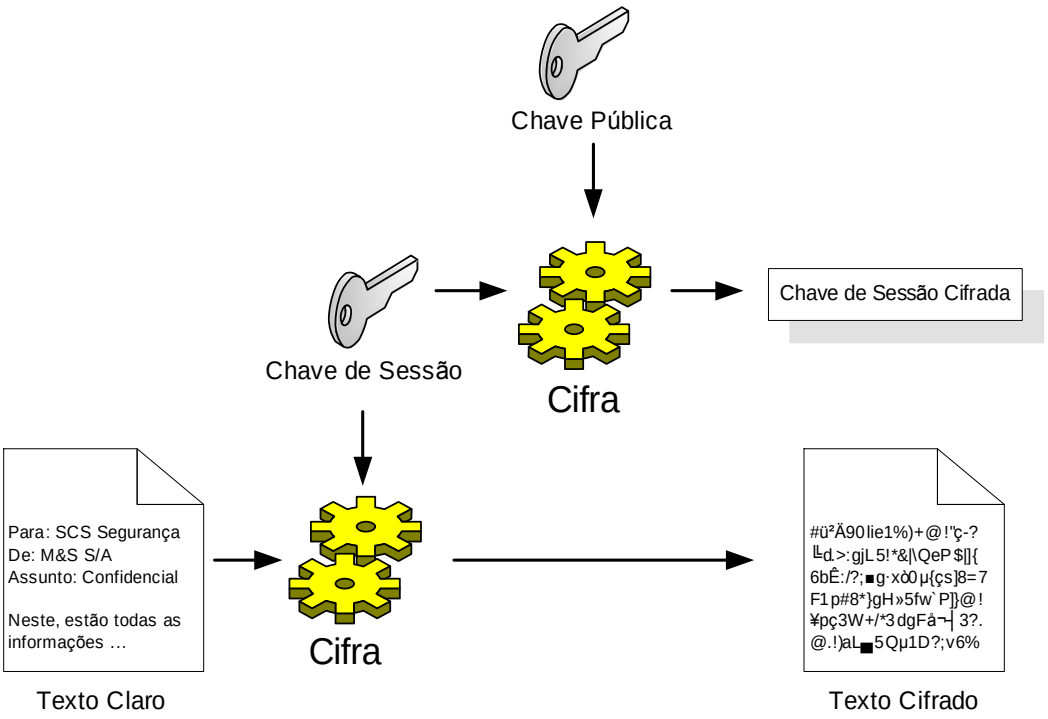

Figura 3.3 – Envelope digital. Processo para cifrar um texto claro utilizando criptografia de chave simétrica e assimétrica.

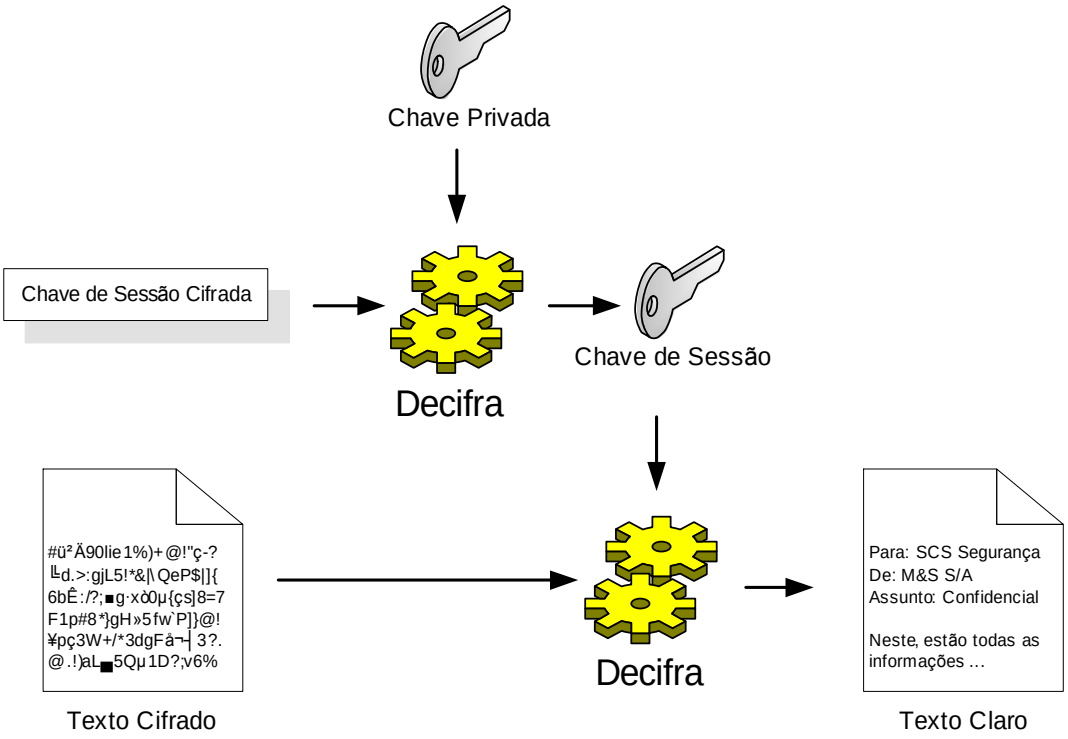

Figura 3.4 – Envelope digital. Processo para decifrar um texto cifrado utilizando criptografia de chave simétrica e assimétrica.

Este mecanismo é muito utilizado para melhorar a performance da criptografia de chave assimétrica, pois o tamanho dos dados provavelmente deve ser maior do que a chave de sessão. A figura 3.3 ilustra o processo para cifrar um texto claro e a figura 3.4 ilustra o processo para decifrar um texto cifrado.

Com a criptografia de chave assimétrica, pode-se prover a autenticação e a confidencialidade. A autenticação é a garantia de identificação das entidades envolvidas em um processo e a garantia de confidencialidade é dada onde somente as entidades envolvidas na comunicação podem ler e utilizar os dados na forma eletrônica.

3.5. Função para Resumo de Dados ou Função *Hash*

A função para resumo de dados retorna um valor resumo, provendo uma imagem representativa compacta da cadeia de bits de entrada e pode ser utilizada como se fosse

unicamente identificável com aquela entrada (STALLINGS, 2003). Desta forma, as funções resumo de dados são utilizadas para garantir a integridade dos dados.

Como exemplo, as funções resumo de dados funcionam de forma semelhante ao dígito verificador do CPF. Se um número qualquer do CPF for modificado, o dígito verificador também será alterado.

Há vários algoritmos de resumo, mas três dominaram o mercado: MD2, MD5 e SHA-1 (BURNETT & PAINE, 2002). O MD2, criado por Ron Rivest, produz um resumo de 128 bits (16 bytes) e foi amplamente utilizado, contudo, com o passar dos anos, defeitos foram encontrados. O MD5, mais rápido e forte, foi a reposta de Rivest aos problemas encontrados nas versões anteriores. O SHA-1 produz um resumo de 160 bits, e variantes deste algoritmo produzem resumo de 192 bits e 256 bits. As partes internas do SHA-1 são mais fortes do que as do MD5 e, por produzir um resumo maior, o uso deste algoritmo é altamente recomendado pela comunidade de criptografia.

Algumas propriedades da função para resumo de dados são destacadas abaixo (STALLINGS, 2003):

- Deve ser computacionalmente inviável fazer a operação inversa, ou seja, dado um resumo deve ser inviável obter os dados originais;
- Dois conjuntos de dados semelhantes devem produzir um resumo completamente diferente;
- Deve ser fácil e rápido produzir um resumo de um conjunto de dados.

A figura 3.5 mostra o resumo de um conjunto de dados utilizando a função para resumo de dados SHA-1. Em (a) e (b) os dados informados na entrada são iguais, com exceção de que em (a) existe um hífen (-) e em (b) o hífen é substituído por um sinal de mais (+). Pode-se observar que o resumo obtido é diferente, apesar da pequena alteração nos dados.

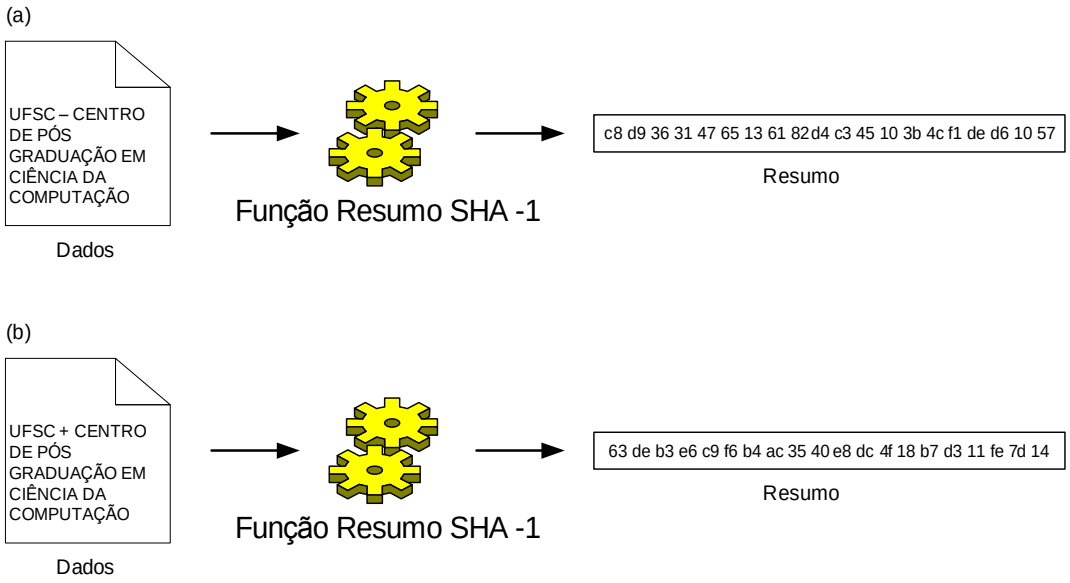

Figura 3.5 – Resumos de dados produzidos através da função para resumo de dados utilizando o algoritmo SHA-1.

Segundo CARVALHO (2001), uma boa função resumo de dados tem uma característica chamada de efeito avalanche. Isto significa que uma pequena mudança no arquivo de entrada acarreta uma grande e imprevisível mudança na saída (resumo).

3.6. Assinatura Digital

Quando se assina um documento no papel, o que se assina efetivamente é o próprio papel. O papel é um meio físico que faz a ligação entre a assinatura propriamente dita e a informação impressa no mesmo papel. A assinatura manuscrita é considerada uma forma de medida biométrica indireta, pois imprime no papel uma escrita que tem certa dependência das bio-características de uma pessoa. Deste modo, existe uma ligação entre a pessoa que assina e o documento papel. Este mesmo caso não ocorre com os documentos eletrônicos, pois não há um meio físico que permita o estabelecimento de uma ligação entre a informação e a assinatura (STALLINGS, 2003).

Uma assinatura digital é um código binário determinado com base no documento e mais alguma outra informação que associe este a uma determinada pessoa ou conjunto de pessoas. Utilizando técnicas de criptografia, a assinatura digital fornece de forma única e exclusiva a comprovação de autoria de um determinado conjunto de dados.

Vários algoritmos de assinatura foram propostos com o passar dos anos, mas apenas o DSA (*Digital Signature Algorithm*), o ECDSA (*Elliptic Curve Digital Signature Algorithm*) e o RSA têm sido adotados por pessoas que utilizam a assinatura digital. Provavelmente o DSA é o segundo algoritmo em utilização, sendo o algoritmo RSA o primeiro. O algoritmo ECDSA faz essencialmente a mesma coisa que o DSA, porém com curvas elípticas (BURNETT & PAINE).

Uma assinatura digital é análoga à assinatura em papel. Ela deve ter as seguintes propriedades (STALLINGS, 2003):

- Deve verificar o autor e a data/hora da assinatura;
- Deve autenticar o conteúdo na época da assinatura;
- Deve ser verificável por terceiros, para o caso da necessidade de se resolver disputas.

A figura 3.6 mostra o processo de assinatura digital utilizando o algoritmo RSA. Neste processo, um resumo é gerado sobre os dados que serão assinados. Ele é cifrado através do algoritmo RSA, em que a chave privada deve ser informada. Assim, com o resumo cifrado é gerada a assinatura digital.

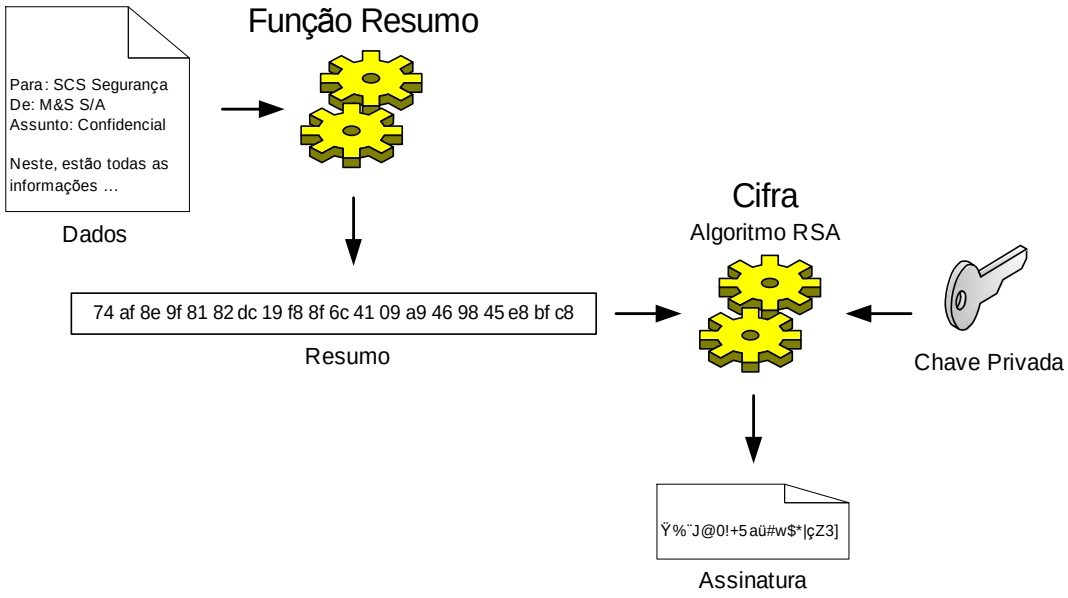

Figura 3.6 – Processo de assinatura digital utilizando o algoritmo RSA.

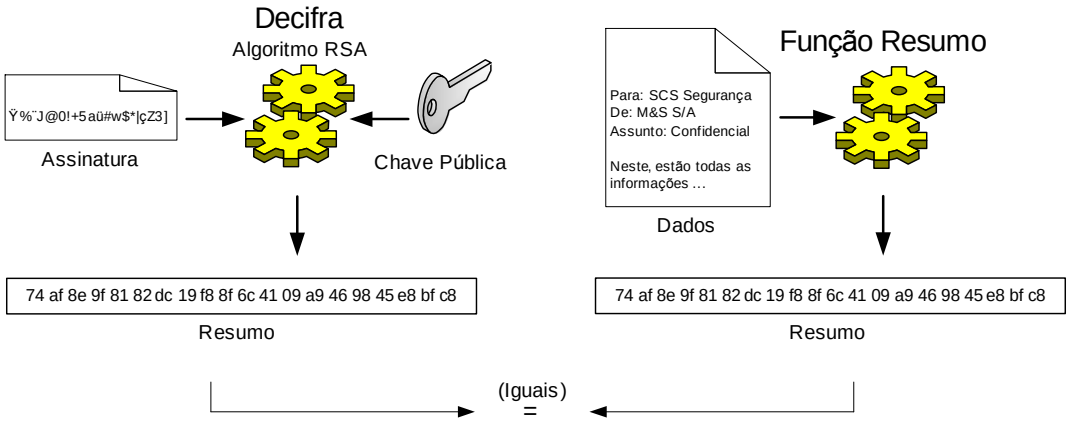

Figura 3.7 – Processo para verificar a assinatura digital utilizando o algoritmo RSA.

A figura 3.7 está mostrando o processo para se verificar a assinatura quando utilizado o algoritmo RSA. Primeiro, decifra-se com a chave pública o resumo cifrado, ou seja, a assinatura. Dessa forma, se obtém um resumo. Depois é gerado um resumo dos dados originais. Então, compara-se o resumo gerado com o resumo decifrado, e se forem iguais, então a assinatura está correta.

Na assinatura digital é utilizada uma combinação de vários mecanismos de criptografia. A tecnologia de chaves públicas resolve o problema da distribuição de chaves. Um resumo de dados assegura a integridade de dados no sentido de que possa detectar-se qualquer alteração nos dados originais. A assinatura digital também oferece autenticação, a qual permite que alguém no mundo eletrônico confirme dados e identidades, e o não-repúdio que impede pessoas de retificarem o que foi acordado em um documento eletrônico.

3.7. Certificado Digital

Certificado digital é um arquivo assinado de forma digital por uma entidade confiável com o objetivo de associar a chave pública a uma determinada entidade. O certificado digital serve, então, como um mecanismo para a divulgação da chave pública.

Uma Autoridade Certificadora (AC) – que é uma entidade confiável – assina o certificado com sua chave privada. Para alguém confirmar a autenticidade do certificado basta ter a chave pública da AC e verificar a assinatura do certificado.

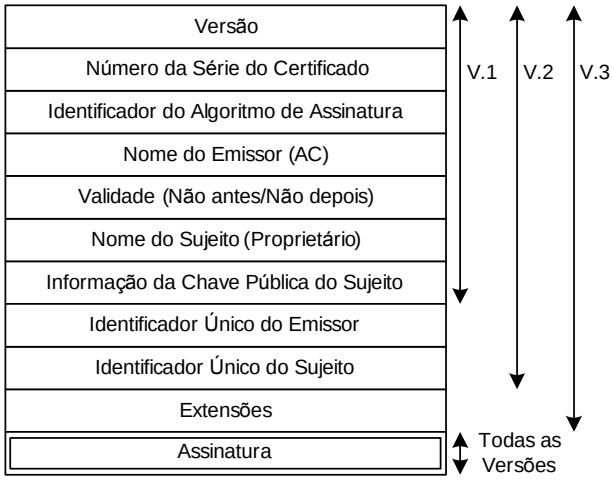

Figura 3.8 – Estrutura do certificado X.509

A estrutura de um certificado contém no mínimo as seguintes informações: chave pública, nome do proprietário, número de série do certificado, nome da AC que emitiu o certificado e assinatura digital da AC. O certificado mais amplamente aceito é o X.509 versão 3, da ITU (BURNETT & PAINE, 2002). A figura 3.8 mostra uma estrutura simplificada de um certificado.

3.8. ICP – Infra-estrutura de Chaves Públicas

Uma Infra-estrutura de Chaves Públicas é um conjunto de ferramentas e processos para a implementação e a operação de um sistema de emissão de certificados digitais, baseada na criptografia de chaves públicas. Engloba também os detalhes do sistema de credenciamento e as práticas e políticas que fundamentam a emissão de certificados e outros serviços relacionados.

De forma genérica, conforme é mostrado na figura 3.9, uma infra-estrutura de chaves públicas envolve um processo colaborativo entre várias entidades: o usuário

final (A), uma AR (Autoridade de Registro) (B), a AC (Autoridade Certificadora) (C) e um repositório de certificados (D). O registro é um dos processos mais importantes em uma ICP, pois é onde o usuário final e a AC estabelecem a confiança. Para tanto, o usuário final pode revisar as políticas e instruções das práticas de certificação do certificado publicado pela AC, e para a AC estabelecer confiança com o usuário final, talvez queira uma comprovação dos rendimentos e a prova de identidade por meio de contato pessoal (BURNETT & PAINE, 2002).

Depois que o registro estiver completo (1) e for estabelecido um relacionamento de confiança entre a AC e o usuário final (2), uma solicitação de certificado pode começar. Normalmente o usuário final gera um par de chaves e envia a chave pública na forma de um padrão de "solicitação de assinatura de certificado", que é composta de dados + chave pública. A AC gera um certificado digital e o assina com a sua chave privada (3). Enfim o certificado digital de chave pública é enviado para um repositório de certificados (e/ou diretório público) (4), e depois normalmente é enviado para a AR (5), que o encaminha para o usuário final (6) ou, então, a AC pode enviar diretamente para o usuário final.

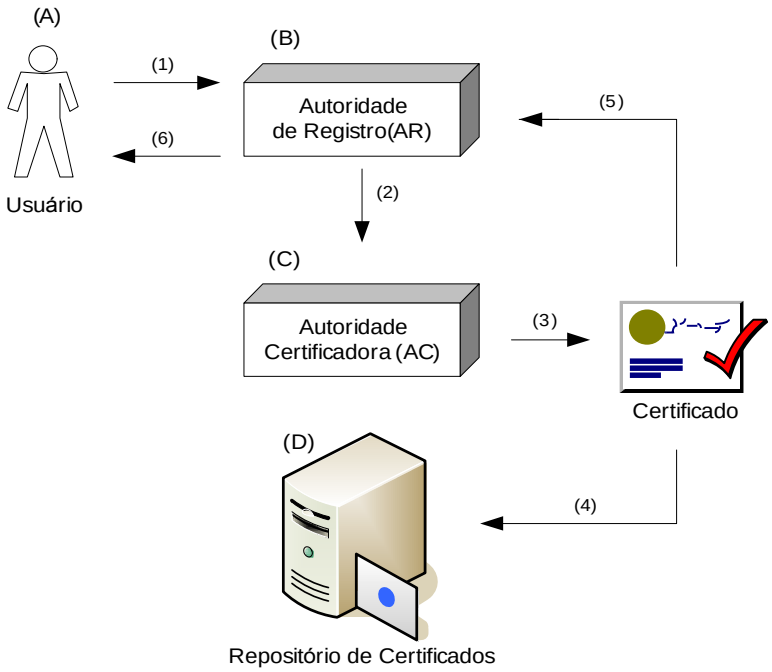

Figura 3.9 – A integração entre os vários componentes da ICP

3.9. Serviços de Segurança e Mecanismos de Criptografia

Serviços de segurança, descrito na seção 3.1, podem ser implementados através de mecanismos de criptografia. A tabela 3.1 apresenta a relação entre os serviços de segurança e os mecanismos de criptografia, conforme descrito nas seções anteriores.

Serviços de Segurança	**Mecanismos de Criptografia**
Sigilo	Criptografia de Chave Simétrica Criptografia de Chave Assimétrica
Integridade	Função para Resumo de Dados (Função *Hash*) Assinatura Digital
Autenticação	Criptografia de Chave Assimétrica Assinatura Digital
Não-repúdio	Assinatura Digital

Tabela 3.1 – Relação entre os serviços de segurança e mecanismos de criptografia.

Nesta relação entre os serviços de segurança e mecanismos de criptografia, a criptografia de chave simétrica implementa o serviço de segurança de sigilo. A criptografia de chave assimétrica implementa os serviços de segurança de sigilo e autenticação. O mecanismo de criptografia de função para resumo de dados implementa o serviço de segurança de integridade. Por fim, a assinatura digital implementa os serviços de segurança de integridade, autenticação e não-repúdio.

3.10. Considerações

A criptografia é a ferramenta mais importante de segurança de dados (BURNETT & PAINE, 2002). Com a criptografia pode-se converter dados legíveis em algo sem sentido – sigilosos – sem se perder a capacidade de recuperar os dados originais a partir destes mesmos dados sem sentido. Neste caso, normalmente é utilizado um mecanismo de criptografia de chave simétrica. Já o mecanismo de criptografia de chave assimétrica resolveu o problema de troca de chaves, por manter uma chave pública e a outra privada, o que também proporcionou atender os serviços de segurança como não-repúdio e autenticação. Com o mecanismo de função para resumo de dados pode-se obter a integridade dos dados.

Desta forma, os mecanismos de criptografia implementam um ou mais serviços de segurança de sigilo, integridade, autenticação e não-repúdio.

4. Criptografia com APIs e Banco de Dados

A indústria de software fornece algumas formas de se alcançar à segurança de dados através da criptografia. Uma forma normalmente utilizada são as APIs de criptografia que são invocadas pelas aplicações que necessitam de alguma segurança.

Gerenciadores de banco de dados possuem formas eficientes de segurança dos dados em repouso e dentre estas formas, alguns ainda utilizam a criptografia como uma opção de aumentar a segurança sobre os dados.

Este capítulo, de forma geral, relaciona algumas APIs de criptografia mais citadas e utilizadas no desenvolvimento de aplicações com segurança. Também apresenta alguns bancos de dados conhecidos, com um enfoque em suas funcionalidades de segurança, fornecidas através da criptografia.

4.1. API de Criptografia

Uma API de criptografia, comumente chamada de CAPI (*Cryptographic Application Programming Interface*) é uma biblioteca de classes com interfaces que os desenvolvedores de software podem invocar para adicionarem os serviços de segurança através da criptografia. O objetivo de uma CAPI é tornar prático para os desenvolvedores a integração da criptografia dentro das aplicações. Nas próximas seções, são listados alguns exemplos de CAPIs (RSA, 2002).

4.1.1. Microsoft CryptoAPI

Proporciona serviços que habilitam os desenvolvedores a adicionar segurança baseada em criptografia em suas aplicações. CryptoAPI inclui funcionalidades para codificar e decodificar ASN.1, cifrar e decifrar dados, autenticação usando certificados digitais e gerenciamento de certificado digital utilizando depósito de certificado. Um subconjunto de interfaces da CryptoAPI baseados no paradigma de componentes é a

CAPICOM que fornece interfaces COM[1], permitindo que os desenvolvedores de aplicação possam facilmente incorporar assinatura digital e funcionalidades de criptografia em suas aplicações baseadas em Windows (MICROSOFT, 2003).

4.1.2. JCA e JCE

JCA (*Java Cryptography Architecture*) ou arquitetura de criptografia Java fornece APIs de criptografia na linguagem Java para assinatura digital, resumo de dados, infra-estrutura para gerenciamento de certificados. Também proporciona um controle de acesso detalhado, altamente configurável, flexível e extensível. O JCE (*Java Cryptography Extension*) estende a API JCA incluindo APIs para cifragem, troca de chaves e MAC (*Message Authentication Code*) ou código de autenticação de informação. Juntos, JCA e JCE proporcionam um completo conjunto de APIs para criptografia, independente de plataforma (SUN, 2002).

4.1.3. RSA BSAFE

Fornece um kit de ferramentas e APIs de criptografia para que os desenvolvedores possam adicionar características de privacidade e autenticação para suas aplicações. É um produto da RSA e possui duas versões. O RSA BSAFE CRYPTO-C, que suporta vários padrões de segurança mundial, e o RSA BSAFE CRYPTO-J que é compatível com alguns padrões da indústria; ambos fornecem para os desenvolvedores o estado da arte em implementação em Java de todas as rotinas mais importantes de privacidade, autenticação e integridade de dados. O RSA BSAFE CRYPTO-J também utiliza algumas APIs de criptografia da linguagem Java (RSA, 2002).

---

[1] COM (*Component Object Model*) é um sistema de plataforma independente, distribuído, orientado a objeto para criar componentes de software binário que podem se interagir (MICROSOFT, 2003).

4.1.4. GSS-API

GSS (*Generic Security Service*) é uma CAPI para serviços de segurança distribuídos. Tem a capacidade de controlar uma sessão de comunicação seguramente, incluindo autenticação, integridade de dados e confidencialidade de dados. A GSS-API foi projetada para isolar seus usuários de especificações dos mecanismos de criptografia e tem sido construído sobre uma gama de mecanismos de chave simétrica e de chave assimétrica (RSA, 2002). A definição da versão atual (versão 2, atualização 1) está na RFC 2743 (GSS, 2000a). Nesta RFC, o GSS-API é definido em uma linguagem independente. A RFC 2853 define GSS-API para linguagem Java, sendo também implementada pela linguagem Java (GSS, 2000b).

4.1.5. GCS-API

GCS (*Generic Crypto Service*) é uma CAPI desenvolvida pelo *The Open Group*. O modelo de programação GCS-API é um conjunto de interfaces para um CSF (*Cryptographic Support Facility*) que pode suportar inúmeros diferentes algoritmos de criptografia. Também fornece suporte para gerenciamento de chaves em aplicações individuais e compartilhadas.

Proporciona serviços de segurança de autenticação de identidades, autenticação de origem dos dados, não repúdio, confidencialidade e proteção da integridade através dos mecanismos de criptografia. Uma preocupação da especificação foi fornecer um padrão de API para serviços de criptografia. (OPENGROUP, 1996).

4.1.6. Security Builder Crypto

Security Builder Crypto é uma CAPI multi-plataforma da Certicom que integra cifragem, chaves públicas e outros mecanismos de criptografia, incluindo muitos algoritmos aprovados pela FIPS[1] dentro de aplicações e dispositivos. É construído com

[1] FIPS (*Federal Information Processing Standard*) – mais especificamente FIPS 140-1 – especifica os requisitos de segurança que devem ser atendidos por um módulo criptográfico, sendo coordenado pelo NIST (*National Institute of Standards and Technology*) dos Estados Unidos.

tamanho de código pequeno e inclui uma gama de algoritmos atuais e legados que fornecem comprovada segurança para ambientes restritos (CERTICOM, 2004).

Esta CAPI possui versões para as linguagens C, C# e Java, sendo que a versão Java possui um maior número de algoritmos de criptografia implementados.

4.2. Criptografia em Banco de Dados

Os bancos de dados de uma forma geral possuem um padrão para segurança de dados. Dentre estes padrões, encontram-se os seguintes:

- **Controle de acesso:** O controle de acesso normalmente é feito através de contas que são compostas de um par de usuário e senha.
- **Sigilo de dados:** O sigilo pode ser alcançado de duas formas. A primeira forma é o sigilo sobre todos os dados armazenados. Nesta forma, normalmente bancos de dados utilizam um mecanismo de criptografia de chave simétrica para cifrar os dados quando são armazenados e decifrar quando são recuperados. A segunda forma é o sigilo em uma unidade menor de dados e normalmente também utiliza o mecanismo de criptografia de chave simétrica. Em banco de dados relacionais, esta segunda forma é conhecida como segurança de dados em nível de colunas das tabelas (ORACLE, 2005).
- **Integridade de dados:** A integridade de dados é fornecida basicamente através do relacionamento entre os dados, restrições e transações atômicas bem como também através de mecanismos de criptografia de resumo de dados. Considerando um banco de dados relacional, os mecanismos de criptografia de resumo de dados basicamente são utilizados como uma forma de manter a integridade de dados em nível de colunas das tabelas.

A seguir, são apresentadas algumas características de segurança em termos de sigilo e integridade de alguns bancos de dados conhecidos.

4.2.1. Db4o

É um banco de objetos de código aberto (*open source*) para desenvolvimento de software em Java e .Net. Este banco de objetos busca melhorar o desenvolvimento e performance evitando o mapeamento objeto-relacional[1], permitindo o armazenamento do próprio objeto de uma aplicação, armazenando até as mais complexas estruturas de objetos com facilidade enquanto atinge altos níveis de performance (DB4O, 2005a).

O Db4o fornece funcionalidades de criptografia. Uma forma de se habilitar a criptografia é dada quando se cria o banco de dados configurando a criptografia e a senha que deverá ser utilizada. Outra forma é permitindo que qualquer usuário possa escolher seus próprios mecanismos de criptografia. O mecanismo de entrada e saída de arquivo do Db4o é configurável e qualquer mecanismo de criptografia de chave simétrica pode ser adicionado. Para tanto, precisa-se implementar a classe *IoAdapter* que irá delegar o acesso ao arquivo para outro *Adapter* usando o padrão de projeto *Decorator*, e também deve-se implementar os métodos para cifrar e decifrar nas operações *read()* e *write()* (DB4O, 2005b).

Não havia até a versão 5.0 do Db4o, funcionalidades embutidas para segurança de dados em uma unidade menor, como por exemplo, sigilo somente em determinados atributos de uma classe.

4.2.2. Oracle

O banco de dados Oracle é um dos bancos de dados mais utilizados pelas corporações. Desde sua versão 8i, este banco de dados possui opções de segurança de dados em nível de colunas das tabelas. Esta segurança pode ser obtida cifrando os dados de determinadas colunas quando eles são armazenados e decifrando estes dados quando eles são recuperados. Normalmente são colunas que contém informações de identificação pessoal que devem ser mantidas integras e em sigilo. Os dados cifrados para as colunas podem utilizar os algoritmos de criptografia 3DES ou AES e a integridade de dados é alcançada através dos algoritmos de criptografia MD5 e SHA-1

[1] Mapeamento necessário quando aplicações são desenvolvidas baseadas no paradigma orientado a objetos e os bancos de dados são relacionais.

(ORACLE, 2005). Também para a transmissão de dados, a segurança pode ser obtida cifrando estes dados utilizando os algoritmos RC4, DES, 3DES, AES.

Para que se possam utilizar estas opções de segurança, o banco de dados Oracle fornece interfaces flexíveis que implementam estes algoritmos de criptografia.

Ele também fornece uma infra-estrutura de gerenciamento de chaves, necessária para implementar a criptografia. Funciona com um mecanismo de chave mestre que cifra as demais chaves utilizadas para cifrar os dados que permanecem em uma tabela, possibilitando assim que os dados cifrados armazenados em backup possam ser carregados novamente sem problemas (ORACLE, 2005).

4.2.3. SQL Server

O banco de dados SQL Server é fornecido pela Microsoft. Ele possui muitas características relacionadas à segurança que ajudam a proteger os dados como, por exemplo, a aplicação de políticas para senhas, uma funcionalidade forte para autenticação, e um modelo granular de hierarquia de permissões. Também inclui capacidade para cifrar dados em nível de coluna das tabelas. Funções e APIs de criptografia embutidos no banco de dados proporcionam facilidades para uma organização criar alternativas de segurança (SQL, 2005).

Suporta três tipos de criptografia. Cada tipo de criptografia utiliza diferentes tipos de chaves e cada tipo tem vários algoritmos de criptografia e tamanho de chaves, conforme segue:

- **Criptografia de chave simétrica:** Suporta os algoritmos de criptografia RC4, RC2, DES e AES.
- **Criptografia de chave assimétrica:** Suporta o algoritmo de criptografia RSA juntamente com o tamanho de chaves de 512 bit, 1024 bit e 2048 bit.
- **Certificados:** O uso de certificado é uma outra forma de criptografia de chave assimétrica. Suporta a especificação X.509v3. Certificados podem ser externos ou o banco de dados poderá gerá-los.

4.2.4. Sybase

O *Sybase Adaptive Server Enterprise*, é um banco de dados bem conhecido. Quanto a sigilo dos dados, ele possui recurso para cifrar os dados em nível de coluna. Podem ser utilizadas instruções SQL para criar chaves para criptografia e especificar colunas para serem cifradas. O Sybase controla o armazenamento e geração de chaves..

Para cifrar os dados, é utilizado o algoritmo simétrico AES. Geração de chaves e funcionalidades de criptografia são fornecidas pelas APIs de criptografia da *Security Builder Crypto* e as chaves são armazenadas de forma cifrada. O processo para cifrar e decifrar dados é semelhante aos demais bancos de dados. Quando os dados são inseridos ou alterados em uma coluna que cifra os dados, transparentemente e imediatamente o banco de dados cifra os dados antes de armazená-los. Quando os dados são selecionados de uma coluna que cifra os dados, o banco de dados decifra os dados logo após os recuperar (SYBASE, 2005b).

4.3. Considerações

Em sua maioria, a indústria de software fornece as implementações de conceitos e algoritmos de criptografia através das CAPIs. Neste capítulo, foram descritos de forma sintetizada algumas APIs de criptografia disponíveis atualmente no mercado para o desenvolvimento de aplicações com segurança. Também foi apresentado como alguns bancos de dados utilizam a criptografia para fornecer maior segurança para dados em repouso.

O objetivo principal deste capítulo foi apresentar de forma geral algumas soluções para segurança, especialmente quanto à criptografia, que existem atualmente e como elas são aplicadas, buscando fornecer subsídios ao desenvolvimento do framework orientado a objetos para segurança de dados em repouso.

5. O Framework Orientado a Objetos Frasedare

Frasedare é um framework orientado a objetos para segurança de dados em repouso, classificado como framework de subsistema (ou sub-aplicação). Tem por objetivo facilitar a inclusão de funcionalidades que garantam a segurança de dados em repouso de aplicações ou de frameworks de aplicações. Ele fornece um conjunto de classes com interfaces bem definidas, que proporcionam o controle da segurança para os dados persistidos e recuperados em um dispositivo de armazenamento. A segurança é alcançada através de mecanismos de criptografia, implementados nas subclasses estendidas do framework Frasedare.

O projeto do framework Frasedare foi implementado na linguagem *Object Pascal* do ambiente Delphi. Este projeto também pode ser implementado em outras linguagens orientadas a objetos, desde que sejam observadas as particularidades de cada linguagem e adequadas conforme a necessidade.

A organização deste capítulo consiste nas fases de desenvolvimento, de uso e de evolução e manutenção do framework Frasedare.

5.1. Fase Desenvolvimento do Framework Frasedare

De acordo com o que foi descrito na seção 2.4.2, metodologias buscam apresentar uma visão geral do processo de desenvolvimento de um framework. Neste sentido, elas definem atividades que devem ser seguidas desde a concepção até o uso de um framework.

Nas próximas seções, serão apresentadas as metodologias aplicadas e as atividades de análise de domínio, de projeto arquitetural, de projeto e de implementação do framework Frasedare. Também serão apresentadas as ferramentas e recursos utilizados.

5.1.1. Ferramentas e Recursos Utilizados

No desenvolvimento do framework Frasedare foram utilizadas as seguintes ferramentas e recursos:

- **Ferramenta de Modelagem UML** – Enterprise Architect 5.0: é uma ferramenta de modelagem UML (*Unified Modeling Language*) da empresa australiana Spax Systems. Ela foi utilizada para modelar a estrutura e comportamento do framework Frasedare com os diagramas da UML. Foram utilizados os diagramas de classes e de seqüência; o primeiro para apresentar a estrutura (modelo estático), e o segundo o comportamento (modelo dinâmico) (UML, 2005).
- **Linguagem de Programação** – Borland Delphi: é uma ferramenta de desenvolvimento de aplicativos produzida pela empresa Borland. Esta ferramenta foi utilizada para implementação do framework Frasedare por utilizar a linguagem *Object Pascal*, assim atendendo ao requisito de orientação a objetos do projeto do framework Frasedare. Outro motivo pelo qual foi utilizada esta linguagem está no fato de que um sistema que utilizaria o framework Frasedare já estar implementado nesta linguagem. A versão do Delphi utilizada para a implementação do framework foi a versão 5, pois o sistema que utilizaria o framework foi desenvolvido nesta versão. Entretanto foram feitos testes na versão Delphi 2005, nos quais foi identificada simplesmente a necessidade de alteração de nomes das bibliotecas requeridas pelo pacote do framework, devendo ser assim estes nomes substituídos pelos da versão Delphi 2005.

5.1.2. Metodologia de Desenvolvimento

Observando-se aplicações que necessitavam de características de segurança para os dados em repouso, e também identificando aplicações que poderiam se beneficiar da utilização destas características, pode-se delimitar um domínio específico para o framework Frasedare.

Analisando-se o domínio, foram identificadas funcionalidades focadas e específicas, não se caracterizando uma aplicação como um todo, mas somente parte dela ou um subsistema.

Assim, o desenvolvimento do framework Frasedare segue em parte a metodologia de projeto da empresa Taligent, que propõe a construção de frameworks estruturalmente menores e focalizados em funcionalidades específicas (TALIGENT, 1995). O benefício obtido pelo framework com esta característica é que ele pode ser facilmente reutilizado em outras aplicações ou ainda, em outros frameworks.

Conforme ocorria o desenvolvimento do framework Frasedare, utilizou-se também a metodologia de projeto Dirigido por *Hot Spots* de PREE (1995), buscando-se encontrar os pontos flexíveis na estrutura de classes do framework Frasedare.

5.1.3. Análise do Domínio

A análise do domínio tem como objetivo descrever o domínio que deverá ser atendido pelo framework, levantando os requisitos e conceitos.

O domínio do projeto framework Frasedare compreende a segurança de dados em repouso, e o resultado da implementação do projeto é um framework que fornece funcionalidades de segurança para os dados em repouso, às aplicações ou outros frameworks que fizerem uso dele.

As fontes para a identificação e o levantamento dos requisitos e conceitos do domínio vieram principalmente da análise de aplicações que precisavam de recursos de segurança de dados em repouso. Também foram exploradas reuniões com especialistas do domínio e aplicações que poderiam utilizar estes recursos.

Dentre as fontes do domínio analisado, as aplicações que precisavam do recurso de segurança de dados em repouso, foram:

- **Autorização de Procedimentos Médicos** – Esta funcionalidade pertence ao sistema Blendus (EXTERKOETTER, 2003). O sistema Blendus, da empresa Extersoft Tecnologia, é um sistema corporativo desenvolvido para gerenciar benefícios assistenciais e financeiros de forma integrada. A funcionalidade de autorização de procedimentos médicos, do módulo de Saúde, tem como

objetivo controlar as solicitações de autorização de procedimentos médicos e/ou hospitalares por beneficiários inscritos em um plano de saúde. As solicitações são analisadas, e autorizadas ou negadas por profissionais da operadora dos planos de saúde, sendo que determinados procedimentos médicos, dependendo de critérios de valor e complexidade, necessitam de perícia médica, efetuada por um perito da área de saúde. Para tanto, este perito deve relatar o seu parecer, dentro da aplicação, quanto aos procedimentos a serem executados. Esta aplicação precisa garantir a integridade, sigilo, autenticação e não-repúdio do parecer relatado. Em termos de arquitetura, esta aplicação é cliente/servidor.

- **Aplicação de Controle de Consulta Psicossocial** – O objetivo desta aplicação é controlar os registros de consultas psicossociais. Ela é utilizada por profissionais de psicologia que entrevistam os pacientes, buscando identificar problemas psicológicos ou sociais envolvendo tanto o próprio paciente quanto sua família. Estas informações devem ser mantidas em sigilo, e a integridade, autenticação e o não-repúdio devem ser garantidos.

Reuniões com especialistas do domínio foram importantes para se esclarecer certos aspectos sobre segurança, assim possibilitando definir e delimitar melhor o domínio para o framework Frasedare.

Quanto às aplicações que poderiam utilizar recursos de segurança de dados em repouso, o objetivo principal foi buscar compreender uma gama de aplicações que poderiam utilizar este recurso; e, dentre várias, foram consideradas as seguintes aplicações:

- **Aplicações que Precisam Armazenar Número de Cartão de Crédito** – Aplicações que armazenam o número do cartão de crédito precisam manter este número a todo sigilo possível, não permitindo o acesso a esta informação sem autorização.
- **Aplicações de Controle de Prontuário Médico** – Este tipo de aplicação normalmente armazena informações confidenciais dos pacientes em prontuários. Portanto, devem ser utilizados recursos de segurança que

permitam a integridade, autenticação e não-repúdio do prontuário escrito por um especialista da área de saúde.

- **Aplicações de Gerenciamento de Informações Empresariais Estratégicas** – Muitas empresas possuem aplicações que gerenciam as informações estratégicas das empresas. Informações deste tipo, na maioria das vezes, devem ser mantidas em sigilo, apenas tendo acesso as mesmas, um grupo seleto e pequeno de pessoas, evitando-se ao máximo que esta informação caia em mãos erradas.

A análise de aplicações, reuniões com especialista no domínio e a análise de aplicações que poderiam utilizar os recursos de segurança, forneceram um conjunto de informações que provocaram uma delimitação e entendimento para o domínio do framework Frasedare.

Assim, foram identificadas algumas características comuns sobre a segurança de dados em repouso, o que proporcionou a identificação dos requisitos e conceitos de domínio para o framework Frasedare, como segue:

a) Os dados informados nos campos da interface com o usuário devem ser armazenados com segurança em algum dispositivo de armazenamento.

b) Os dados seguros devem ser recuperados de algum dispositivo de armazenamento e apresentados na forma original (sem segurança) nos correspondentes campos da interface com o usuário, verificando-se a integridade, autenticação e não-repúdio.

c) Proporcionar segurança na forma de sigilo, integridade, autenticação e não-repúdio nos dados em repouso, possibilitando a combinação entre estas formas de segurança.

De forma geral, estes foram os requisitos para o framework Frasedare, identificados na análise de domínio.

Com os requisitos identificados e outras informações obtidas da análise de domínio, também os principais conceitos foram identificados.

Pode se observar que foram identificados alguns conceitos relacionados à segurança, como sigilo, integridade, autenticação e não-repúdio de dados. Estes conceitos de acordo com a seção 3.1, são tratados como serviços de segurança de dados.

Os serviços de segurança de dados podem ser implementados através de mecanismos de criptografia. Dependendo destes mecanismos, mais de um serviço de segurança pode ser implementado conforme descrito na seção 3.9. Os mecanismos de criptografia considerados neste trabalho foram o de criptografia de chave simétrica, criptografia de chave assimétrica, resumo de dados e assinatura digital.

Abaixo seguem os conceitos identificados como pertinentes para o projeto do framework Frasedare:

a) Campo de Dados da Interface Persistente e Seguro;

b) Criptografia Simétrica;

c) Criptografia Assimétrica;

d) Resumo de Dados;

e) Assinatura Digital;

f) Segurança de Dados.

Baseado nos requisitos e conceitos identificados foi definido o modelo de domínio. Segundo, LARMAN (2004), um modelo de domínio é uma representação de classes conceituais do mundo real. Um modelo de domínio é amplamente utilizado como fonte de inspiração para projetar objetos de software. Assim, identificar um conjunto rico de objetos ou de classes conceituais é um esforço que vale a pena pelo retorno que traz na fase de projeto e implementação.

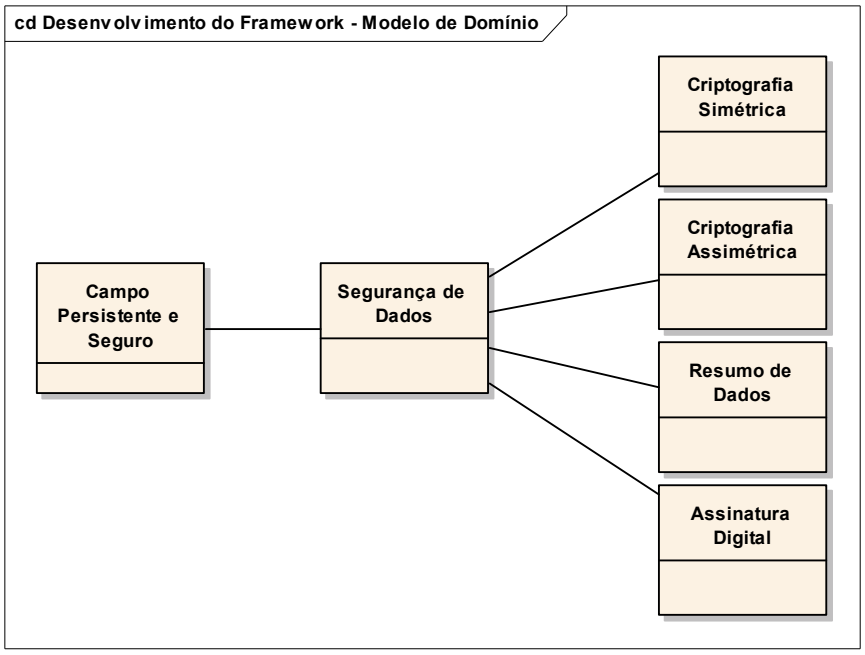

Figura 5.1 – Modelo de domínio do framework Frasedare

A figura 5.1 ilustra o modelo de domínio do framework Frasedare com as classes conceituais e relacionamentos identificados nos requisitos levantados na análise do domínio.

Abaixo são descritos os objetivos de cada classe conceitual:

- **Criptografia Simétrica** – É uma classe conceitual que representa o mecanismo de criptografia de chave simétrica. A criptografia de chave simétrica implementa o serviço de segurança de sigilo.
- **Criptografia Assimétrica** – Esta classe conceitual representa o mecanismo de criptografia de chave assimétrica. Com a criptografia de chave assimétrica se implementam os serviços de segurança de sigilo e autenticação.
- **Resumo de Dados** – É a classe conceitual que foi identificada para representar o mecanismo de criptografia de resumo de dados. Este mecanismo de criptografia implementa o serviço de segurança de integridade.
- **Assinatura Digital** – A assinatura digital é a classe conceitual identificada para representar o mecanismo de criptografia de assinatura digital. Este

mecanismo de criptografia implementa os serviços de integridade, autenticação e não-repúdio.

- **Segurança de Dados** – Esta classe conceitual possui a associação das classes conceituais de criptografia de chave simétrica, criptografia de chave assimétrica, resumo de dados e assinatura digital. Ela deve controlar a combinação entre estas classes conceituais de criptografia de chave simétrica ou criptografia de chave assimétrica e resumo de dados ou assinatura digital, dependendo da configuração de segurança desejada.
- **Campo Persistente e Seguro** – Esta classe conceitual possui uma associação com a classe conceitual de Segurança de Dados. Esta classe conceitual representa um campo de dados persistente que implementa a segurança de dados. Este campo de dados persistente fica associado a um conjunto de campo de dados que é armazenado e recuperado através de um mecanismo de persistência. A classe conceitual Segurança de Dados associada a esta classe conceitual atua aplicando a segurança nos dados originais quando são persistidos, e recuperando os dados originais ao retirar a segurança de dados armazenados com segurança em um dispositivo de armazenamento.

O modelo de domínio apresenta os principais conceitos e relacionamentos do framework Frasedare, possibilitando assim um entendimento inicial da estrutura, além de servir como subsídio para identificar as classes de software na atividade de projeto.

5.1.4. Projeto Arquitetural

O projeto arquitetural define um estilo de arquitetura adequado para formar a base para um framework.

Partindo-se da análise do domínio, o projeto arquitetural é definido. O framework Frasedare foi desenvolvido para aplicações que possuam uma arquitetura cliente/servidor. Ele trabalha entre a camada de aplicação e os mecanismos de persistência. A figura 5.2 ilustra a arquitetura utilizada em uma aplicação que usa o framework Frasedare.

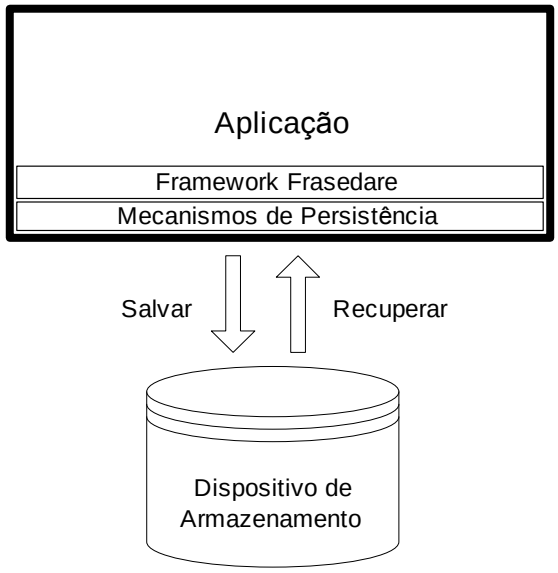

Figura 5.2 – Arquitetura de uma aplicação que usa o framework Frasedare.

Normalmente em uma aplicação com arquitetura cliente/servidor, os dados são representados através de campos em uma interface com o usuário na aplicação cliente. Cada campo possui uma conexão com algum mecanismo de persistência. Este mecanismo de persistência faz a interface entre o campo da aplicação e um equivalente no dispositivo de armazenamento. Considerando um banco de dados relacional, este equivalente pode ser uma coluna em uma tabela de banco de dados, conforme apresentado na figura 5.3. Assim, ao se salvar os dados que estão nos campos da interface de usuário, a aplicação armazena os dados que estão nos campos, nas respectivas colunas da tabela. Para recuperar estes dados, uma consulta é efetuada pela aplicação no banco de dados sobre as linhas da tabela para novamente popular os campos da interface de usuário.

Para uma aplicação que faz uso do framework Frasedare, uma camada adicional deve ser considerada, isto é, a camada do framework Frasedare, conforme apresentado na figura 5.2.

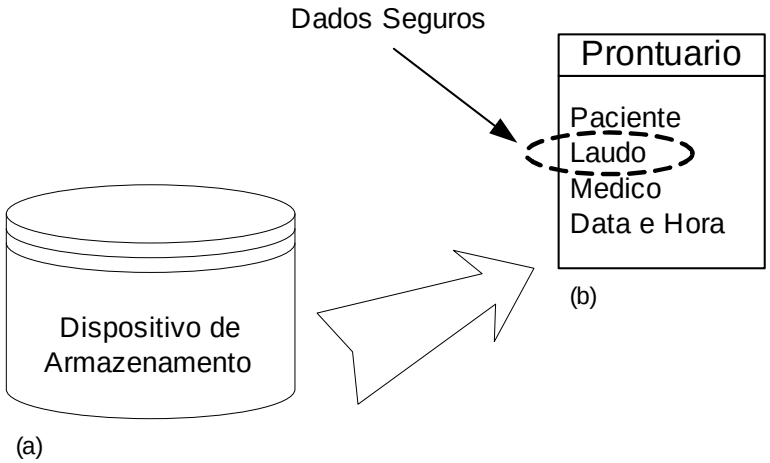

Figura 5.3 – Estrutura de armazenamento. (a) pode ser um arquivo ou gerenciador de banco de dados; (b) a estrutura de um arquivo ou tabela, indicando a coluna que recebe dados seguros.

Nesta arquitetura, o framework Frasedare atua sobre a camada de mecanismos de persistência, assegurando que a segurança seja adicionada quando os dados são persistidos, e que a segurança seja retirada quando os dados forem recuperados.

5.1.5. Projeto do Framework Frasedare

O projeto do framework Frasedare partiu do produto resultante da análise de domínio e projeto arquitetural. No projeto do framework Frasedare foram definidas as classes de software, e o resultado foi uma estrutura com interfaces bem definidas.

Inicialmente, as classes de software foram identificadas a partir do modelo de domínio. Depois, outras foram adicionadas buscando-se a generalização das classes para possibilitar a flexibilidade do framework Frasedare e também pela exigência de algumas funcionalidades necessárias em certos mecanismos de criptografia.

Como uma decisão de projeto no desenvolvimento do framework Frasedare, os mecanismos de criptografia foram agrupados em duas categorias: sigilo e integridade. A primeira categoria, *sigilo*, agrupa os mecanismos de criptografia de chave simétrica e criptografia de chave assimétrica, já a segunda, *integridade*, agrupa os mecanismos de resumo de dados e assinatura digital. Assim, a categoria de sigilo agrupa os mecanismos de criptografia que implementam pelo menos o serviço de segurança de sigilo, e a

categoria de integridade agrupa os mecanismos de criptografia que implementam o serviço de segurança de integridade. Nesse sentido, ocorreu toda a evolução do framework Frasedare, sempre considerando estas duas categorias.

Desta forma, quando se fizer uso do framework Frasedare, poderá se implementar somente uma das categorias ou ambas, dependendo do objetivo de segurança requerido pela aplicação. Porém, no mínimo uma categoria deverá ser implementada.

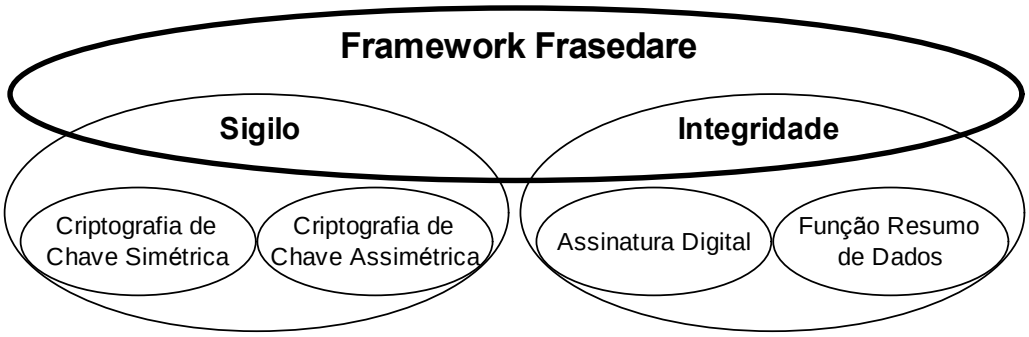

Figura 5.4 – Agrupamento dos mecanismos de criptografia em duas categorias, sigilo e integridade.

A figura 5.4 ilustra o agrupamento dos mecanismos de criptografia do framework Frasedare nas categorias de sigilo e integridade.

Categoria	Mecanismos de Criptografia	Serviços de Segurança
Sigilo	Criptografia de Chave Simétrica	Sigilo
	Criptografia de Chave Assimétrica	Sigilo e Autenticação
Integridade	Resumo de Mensagem	Integridade
	Assinatura Digital	Integridade, Autenticação e Não-Repúdio
Sigilo e Integridade	Criptografia de Chave Simétrica e Resumo de Mensagem	Sigilo e Integridade
	Criptografia de Chave Simétrica e Assinatura Digital	Sigilo, Integridade, Autenticação e Não-Repúdio
	Criptografia de Chave Assimétrica e Resumo de Mensagem	Sigilo, Autenticação e Integridade
	Criptografia de Chave Assimétrica e Assinatura Digital	Sigilo, Autenticação, Integridade e Não-Repúdio

Tabela 5.1 – Categorias de segurança e suas combinações.

Com a categorização dos mecanismos de criptografia em sigilo e integridade, o framework Frasedare passou a ter as possibilidades de segurança apresentadas na tabela 5.1.

Para ampliar as opções de dispositivos de armazenamento, foi definido no projeto do framework Frasedare que os dados devem ser armazenados em formato de texto ASCII. Para tanto, na implementação das subclasses concretas dos mecanismos de criptografia do framework Frasedare, deve-se considerar a função de codificação Base64. Base64 é um esquema utilizado para transformar dados binários em caracteres ASCII, com três bytes de dados binários para quatro caracteres de seis bits.

5.1.5.1. Convenções de Nomes e Notação no Projeto

Por convenção, as classes de software definidas no ambiente Delphi recebem antes do nome a letra *T*. Assim, as classes definidas no framework Frasedare também se basearam nesta convenção.

Para melhor identificar as classes, foi adotado um conjunto de quatro letras *Fsdr*. Este conjunto de letras foi obtido das primeiras letras da frase Framework para Segurança de Dados em Repouso. Elas são concatenadas logo após a letra *T* da convenção do ambiente Delphi.

Os nomes de classes, atributos e operações que estão em português foram incluídos na construção do framework Frasedare. Os nomes em inglês são das APIs de criptografia e classes que foram reusadas do ambiente Delphi.

Para identificar as classes quanto à classificação de redefinibilidade e essencialidade definida por SILVA (2000), foram utilizados os estereótipos de «R» e «E», respectivamente, ou «RE» quando forem ambas as classificações, também utilizados por EXTERKOETTER (2003).

As classes mais escuras apresentadas nos digramas são externas ao framework Frasedare e as demais são classes pertencentes a ele.

5.1.5.2. Modelo Estático – Estrutura

O modelo estático é representado pelo diagrama de classes da UML e captura a estrutura lógica de um software, ilustrando as classes e seus relacionamentos, e os atributos e operações que as classes de software possuem.

A figura 5.5 apresenta as classes de software e seus relacionamentos no projeto do framework Frasedare, excluindo os atributos e as operações.

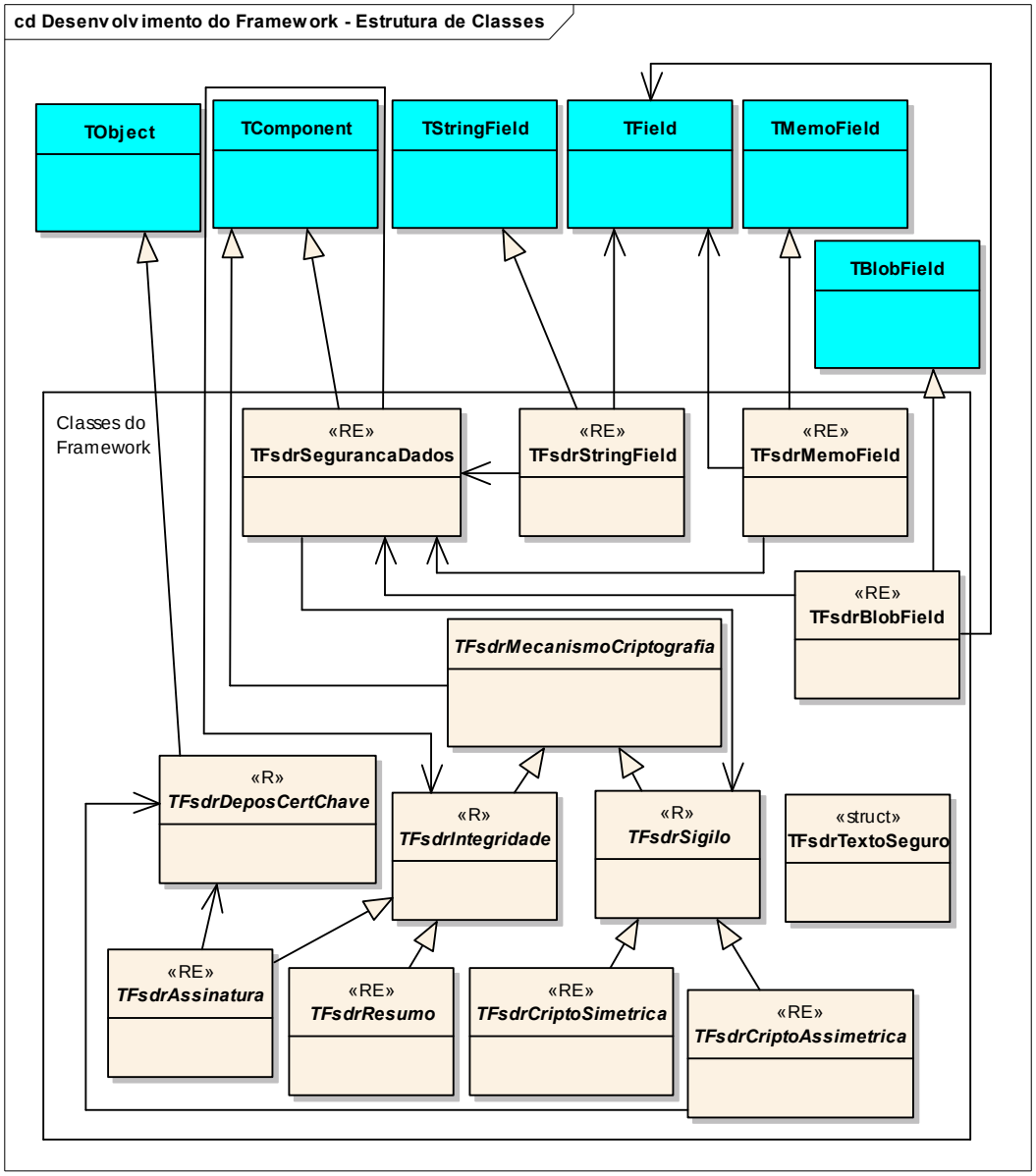

Figura 5.5 – Estrutura de classes do framework Frasedare.

Buscando dar um melhor entendimento do modelo estático, a seguir são descritas as responsabilidades das classes no projeto do framework Frasedare apresentadas na figura 5.5.

- ***TObject*** – É uma classe externa ao framework Frasedare. Ela é a classe base para todas as outras classes da linguagem *Object Pascal* do ambiente Delphi. No framework Frasedare, ela é a superclasse da classe *TFsdrDeposCertChave*.
- ***TComponent*** – É uma classe externa ao framework Frasedare. Ela é a classe base para as classes componentes e possui propriedades específicas que proporcionam o gerenciamento de componentes no ambiente Delphi. É a superclasse das classes *TFsdrSegurancaDados* e *TFsdrMecanismoCriptografia* do framework Frasedare.
- ***TField*** – É uma classe externa ao framework Frasedare. É a superclasse de campos persistentes do ambiente Delphi.
- ***TBlobField*** – Externa ao framework Frasedare. É uma classe de campo persistente do ambiente Delphi que encapsula um comportamento comum para dados *binários* e mecanismo de persistência.
- ***TStringField*** – Externa ao framework Frasedare. É uma classe de campo persistente do ambiente Delphi que encapsula um comportamento comum para dados *string* e mecanismo de persistência.
- ***TMemoField*** – É externa ao framework Frasedare. É uma classe de campo persistente do ambiente Delphi que encapsula um comportamento comum para dados *memo* e mecanismo de persistência.
- ***TFsdrTextoSeguro*** – É uma estrutura de dados do tipo registro definida para o framework Frasedare. Possui dois atributos de dados, um que é utilizado para sigilo e outro para integridade. Esta estrutura é utilizada para passagem de parâmetro e retorno das operações.
- ***TFsdrStringField*** – Esta é uma subclasse concreta de campo persistente de *TStringField*. Implementa a segurança nos campos persistentes através de

uma associação com a classe *TFsdrSegurancaDados*. Também possui uma associação com a classe externa *TField* com a responsabilidade de indicar o campo de integridade para os mecanismos de criptografia da categoria de integridade. É classificada como redefinível e essencial.

- ***TFsdrMemoField*** – Esta é uma subclasse concreta de campo persistente de *TMemoField*. Implementa a segurança nos campos persistentes através de uma associação com a classe *TFsdrSegurancaDados*. Também possui uma associação com a classe externa *TField* com a responsabilidade de indicar o campo de integridade para os mecanismos de criptografia da categoria de integridade. É classificada como redefinível e essencial.
- ***TFsdrBlobField*** – É uma subclasse concreta de campo persistente de *TBlobField*. Implementa a segurança nos campos persistentes através de uma associação com a classe *TFsdrSegurancaDados*. Também possui uma associação com a classe externa *TField* com a responsabilidade de indicar o campo de integridade para os mecanismos de criptografia da categoria de integridade. É classificada como redefinível e essencial.
- ***TFsdrMecanismoCriptografia*** – É uma subclasse abstrata de *TComponent*. É superclasse para todas as outras classes que representam mecanismos de criptografia utilizados no framework Frasedare, define interfaces comuns para todas as subclasses.
- ***TFsdrSigilo*** **–** É uma subclasse abstrata de *TFsdrMecanismoCriptografia.* Define interfaces comuns para a categoria de mecanismos de criptografia que implementam pelo menos os serviços de segurança de sigilo. É classificada como redefinível.
- ***TFsdrIntegridade*** – É uma subclasse abstrata de *TFsdrMecanismoCriptografia*. Define interfaces comuns para a categoria de mecanismos de criptografia que implementam pelo menos os serviços de segurança de integridade. É classificada como redefinível.
- ***TFsdrCriptoSimetrica*** – É subclasse de *TFdsrSigilo*. Foi definida baseando-se na classe conceitual *Criptografia Simétrica* do modelo de domínio e

especifica as interfaces para implementação de mecanismo de criptografia de chave simétrica. É redefinível e essencial.

- ***TFsdrCriptoAssimetrica*** – É subclasse de *TFdsrSigilo*. Foi definida baseada na classe conceitual *Criptografia Assimétrica* do modelo de domínio e especifica as interfaces para implementação de mecanismo de criptografia de chave assimétrica. Possui uma associação com a classe *TFsdrDeposCertChave*. É redefinível e essencial.
- ***TFsdrAssinatura*** – É subclasse de *TFsdrIntegridade.* Foi definida baseada na classe conceitual *Assinatura Digital* do modelo de domínio e especifica as interfaces para implementação de assinatura digital. Possui uma associação com a classe *TFsdrDeposCertChave*. É redefinível e essencial.
- ***TFsdrResumo*** – É subclasse de *TFsdrIntegridade.* Foi definida baseada na classe conceitual *Resumo de Dados* do modelo de domínio e especifica as interfaces para implementação de resumo de dados. É redefinível e essencial.
- ***TFsdrDeposCertChave*** – É uma subclasse abstrata de *TObject* do ambiente Delphi. Esta classe foi incluída no projeto do framework Frasedare para definir as interfaces que manipulam os certificados digitais e chaves. Está associada às classes *TFsdrSegurancaDados, TFsdrCriptoAssimetrica* e *TFsdrAssinatura* as quais necessitam conhecer certificados digitais para implementar seus comportamentos. É definida como um padrão de projeto *Singleton.* É classificada como redefinível e essencial.
- ***TFsdrSegurancaDados*** – É subclasse de *TComponent*. Esta classe foi definida no projeto do framework Frasedare para controlar e gerenciar a segurança. Corresponde à classe conceitual *Segurança de Dados* do modelo de domínio. Possui uma associação com as classes *TFsdrIntegridade* e *TFsdrSigilo*. Esta classe especifica operações que aplicam a segurança de dados em repouso e retiram a segurança para obter os dados originais quando os dados são recuperados novamente do dispositivo de armazenamento. É redefinível e essencial.

As classes *TFsdrAssinatura, TFsdrResumo, TFsdrCriptoSimetrica* e *TFsdrCriptoAssimetrica* possuem classificação de redefinibilidade e essencialidade pois

no mínimo uma delas deve ser estendida para se implementar um mecanismo de criptografia específico, ao se utilizar o framework Frasedare. Assim, mesmo sendo todas estas classes classificadas como redefiníveis e essenciais, no mínimo uma deverá ser estendida para que o framework Frasedare possa ser utilizado. O mesmo acontece com as classes *TFsdrStringField*, *TFsdrMemoField* e *TFsdrBlobField*, que possuem a classificação de redefinibilidade e essencialidade, pois no mínimo uma delas deverá ser utilizada na aplicação que usar o framework Frasedare, não sendo necessário que todas sejam utilizadas.

Uma descrição das responsabilidades das classes foi fornecida até agora. A seguir, serão apresentadas as classes com seus respectivos atributos e operações.

Para possibilitar uma melhor visualização das classes e seus respectivos atributos e operações, o conjunto de classes do framework Frasedare foi dividido em dois diagramas de classes, o primeiro apresentando as classes de mecanismos de criptografia e o segundo as classes de segurança, visualização e persistência.

A figura 5.6 apresenta o diagrama de classes contendo as classes de mecanismos de criptografia do framework Frasedare, com seus respectivos atributos e operações.

Quanto a alguns termos utilizados na descrição do projeto do framework Frasedare, foram utilizadas algumas adaptações. Como o framework Frasedare considera duas categorias de mecanismos de criptografia, uma que implementa pelo menos o serviço de segurança de sigilo e outra que implementa pelo menos o serviço de segurança de integridade, por convenção e generalização para a descrição do projeto do framework Frasedare, o termo “texto sigiloso”, indicando a categoria de sigilo, será utilizado no lugar de “texto cifrado”, e o termo “texto integridade”, indicando a categoria de integridade, será utilizado no lugar de “assinatura digital” ou “resumo de dados”. Porém, os termos mais específicos poderão ser utilizados quando for necessário tratar diretamente de algum mecanismo de criptografia específico.

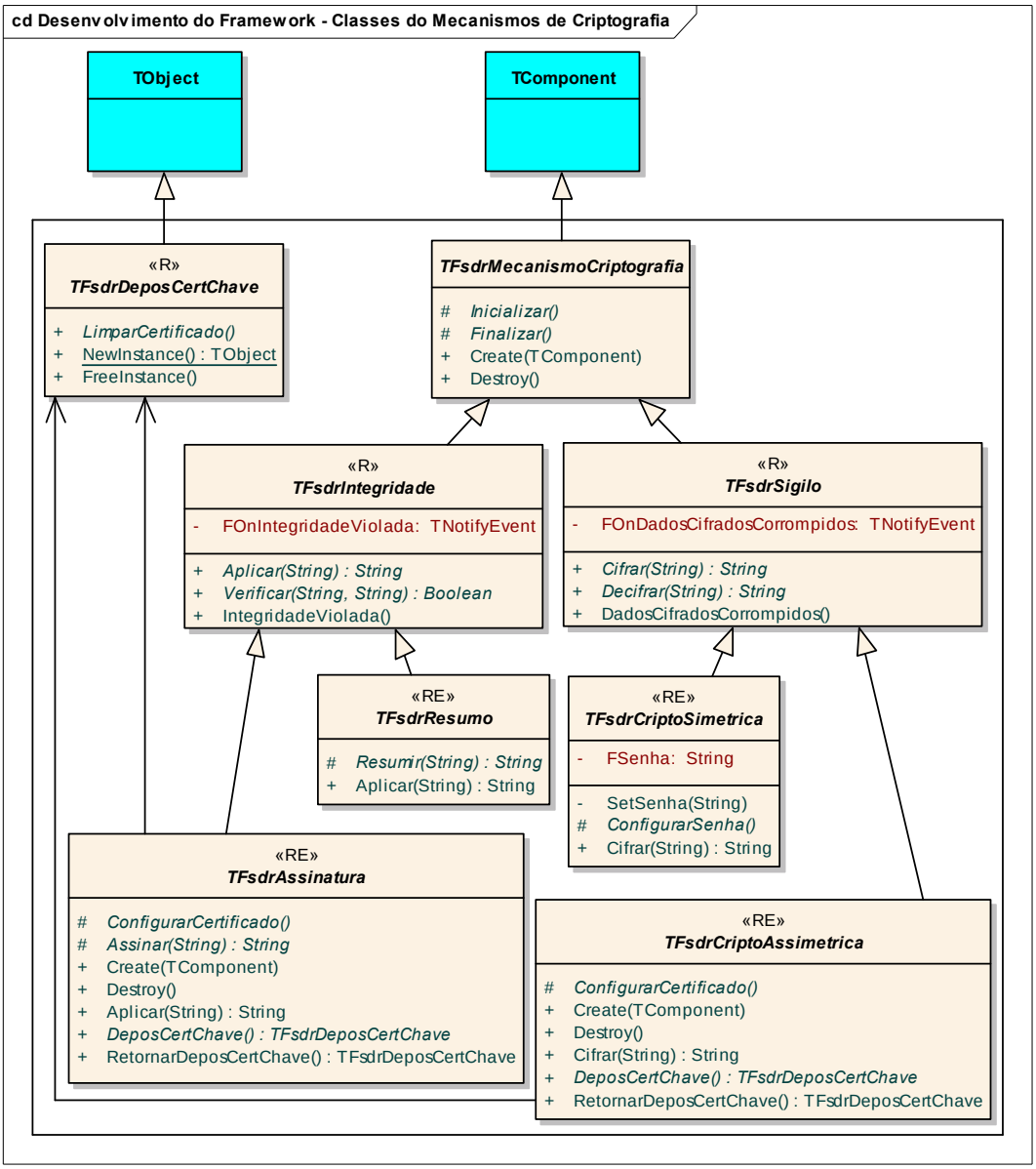

Figura 5.6 – Diagrama de classes contendo as classes de mecanismos de criptografia do framework Frasedare.

As descrições dos atributos e dos comportamentos dos métodos que implementam as operações das classes apresentadas na figura 5.6, se encontram no apêndice I.

A figura 5.7 demonstra o diagrama de classes das classes do framework Frasedare de segurança, visualização e persistência, com seus respectivos atributos e operações.

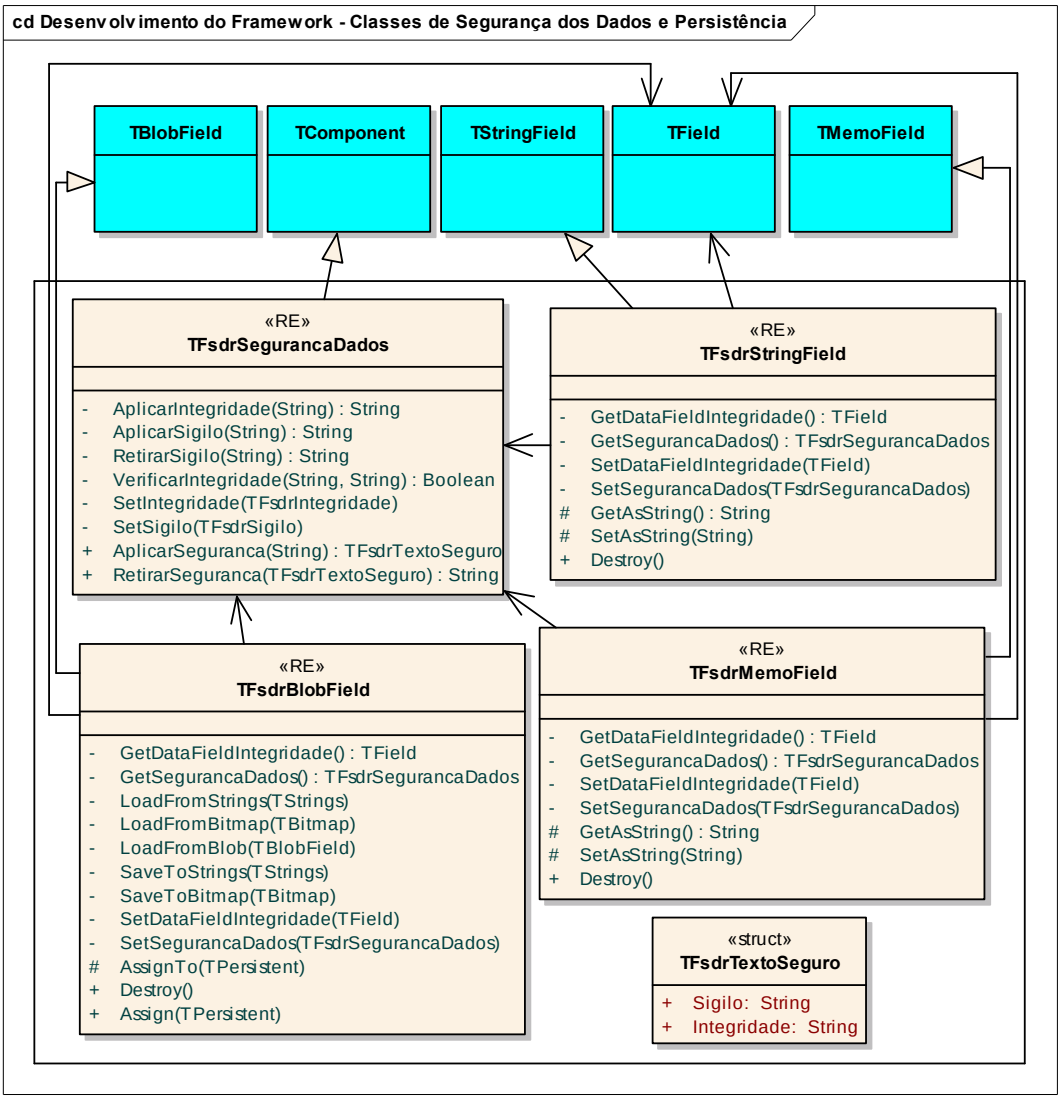

Figura 5.7 – Diagrama de classes contendo as classes concretas de segurança de dados e persistência.

As descrições dos atributos e dos comportamentos dos métodos que implementam as operações das classes apresentadas na figura 5.7, se encontram no apêndice II.

O modelo estático do framework Frasedare foi descrito em termos de classes, atributos, operações e associações.

Para uma maior compreensão da interação entre as instâncias de classes do framework Frasedare através do envio de mensagens, a próxima seção apresenta o modelo dinâmico.

5.1.5.3. Modelo Dinâmico – Comportamento

O modelo dinâmico é representado pelos diagramas de interação da UML. Um diagrama de interação exibe interações entre as instâncias das classes, através do envio de mensagens entre elas.

- **Interações entre a interface de usuário e as instâncias de campo persistente e seguro das classes *TFsdrMemoField* e *TFsdrStringField*.**

A figura 5.8 apresenta as interações através do envio de mensagens entre a interface de usuário e instâncias da classe de campo persistente e seguro *TFsdrMemoField*, e da classe de segurança de dados *TFsdrSegurancaDados*.

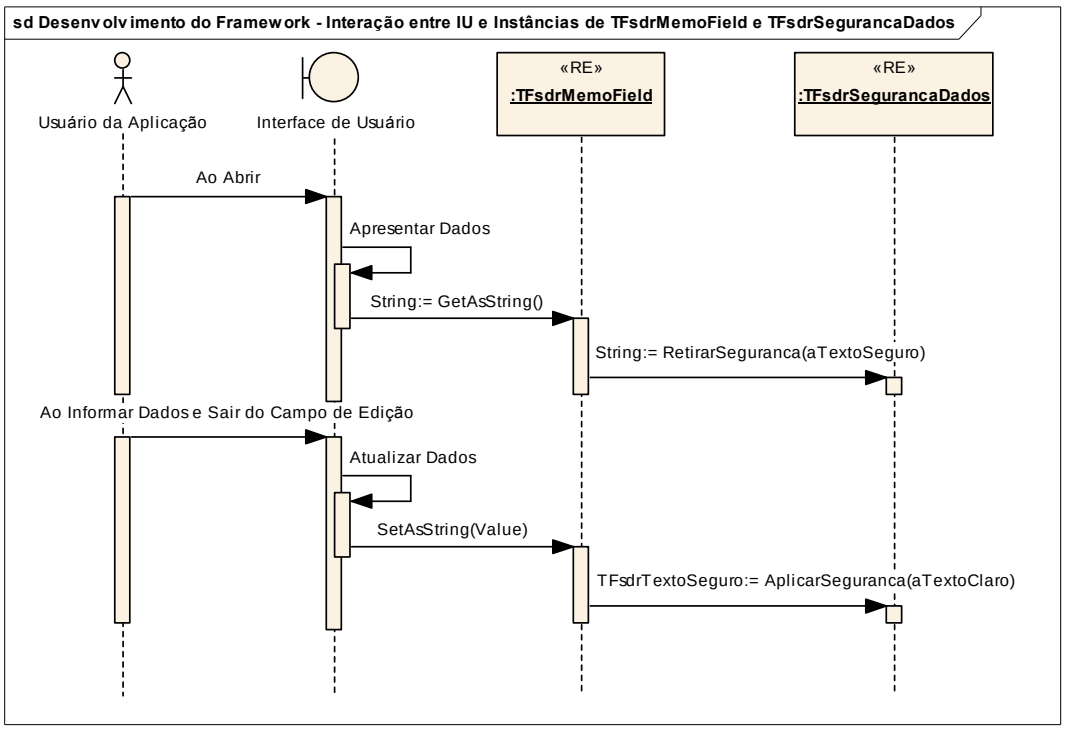

Figura 5.8 – Diagrama de seqüência apresentando a interação entre interface de usuário e instâncias das classes *TFsdrMemoField* e *TFsdrSegurancaDados*.

A figura 5.8 apresenta a interação entre uma instância da classe *TFsdrMemoField* de campo persistente e seguro, e uma instância da classe *TFsdrSegurancaDados*. Apesar de a figura estar apresentando a classe *TFsdrMemoField,* o mesmo comportamento acontece ao se substituir esta classe pela *TFsdrStringField*. O comportamento da classe de interface de usuário não será detalhado, nem seus parâmetros e retornos, pois foge do escopo deste trabalho.

A leitura do comportamento apresentado na figura 5.8 é feito da seguinte maneira: Se existir um campo persistente e seguro do tipo *TFsdrMemoField* ou *TFsdrStringField* sendo utilizado por uma interface de usuário, ao abrir a interface, os dados originais serão apresentados ao se recuperar a informação do dispositivo de armazenamento através do envio da mensagem *GetAsString()* para a instância da classe *TFsdrMemoField.* Em seguida, é retirada a segurança de dados através da mensagem *RetirarSeguranca(aTextoSeguro)* enviada para uma instância da classe *TFsdrSegurancaDados*.

A segurança nos dados é aplicada quando o usuário informa os dados no campo de edição e sai dele. Neste momento, a mensagem *SetAsString(Value)* é enviada a uma instância da classe *TFsdrMemoField* que delega os dados informados para receberem a segurança através do envio da mensagem *AplicarSeguranca(aTextoClaro)* para uma instância da classe *TFsdrSegurancaDados,* e por fim os dados já seguros são atualizados no dispositivo de armazenamento.

- **Interações entre a interface de usuário e a instância de campo persistente e seguro da classe *TFsdrBlobField*.**

A figura 5.9 apresenta o envio de mensagens entre uma interface de usuário e as instâncias da classe de campo persistente e seguro *TFsdrBlobField*, e da classe de segurança de dados *TFsdrSegurancaDados*.

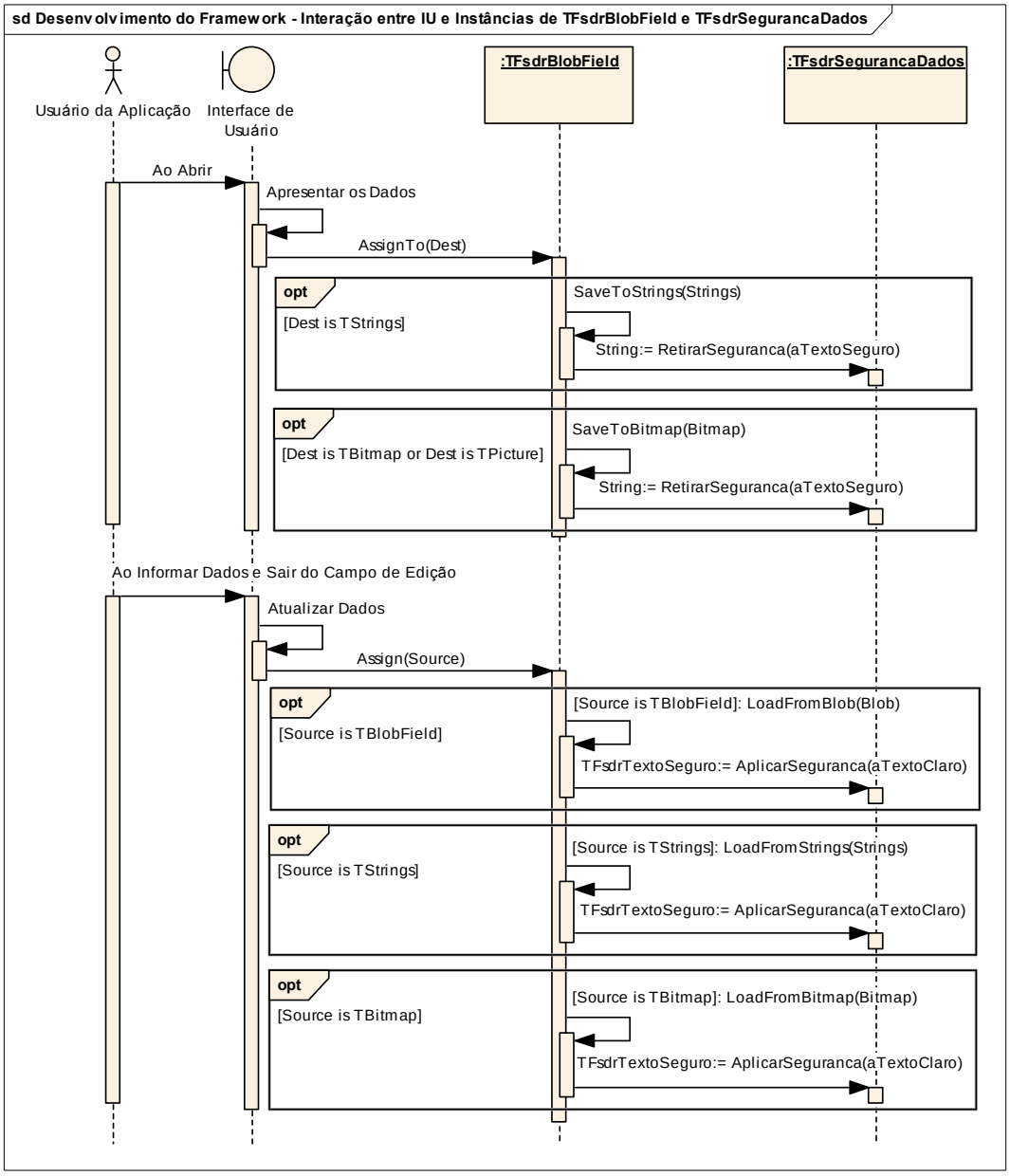

Figura 5.9 – Diagrama de seqüência apresentando a interação entre a interface de usuário e instâncias das classes *TFsdrBlobField* e *TFsdrSegurancaDados*.

A figura 5.9 apresenta a interação entre uma interface de usuário e as instâncias da classe *TFsdrBlobField* de campo persistente e seguro, e da classe *TFsdrSegurancaDados*. Como anteriormente já mencionado, aqui também o comportamento da classe de interface de usuário não será detalhado, nem seus parâmetros e retornos.

O comportamento da classe *TFsdrBlobField* se difere do comportamento apresentado nas classes de campo persistente e seguro *TFsdrMemoField* e *TFsdrStringField*, pois dados como gráficos e imagens são considerados tipo binário e as classes de interface de usuário que tratam deste tipo de dados, invocam operações diferentes de campos persistentes das classes de interface de usuário que tratam de dados do tipo *string*.

O comportamento se apresenta da seguinte maneira: Se existir um campo persistente e seguro do tipo *TFsdrBlobField* sendo utilizado pela interface de usuário, ao abrir a interface, os dados originais serão apresentados ao se recuperar a informação do dispositivo de armazenamento através do envio da mensagem *AssignTo(Dest)* para a instância da classe *TFsdrBlobField.* A operação *AssignTo(Dest)* verifica qual o tipo do parâmetro *Dest* recebido e então envia a mensagem *SaveToStrings(Strings)* se o parâmetro for do tipo *TStrings* ou envia a mensagem *SaveToBitmap(Bitmap)* se o parâmetro for do tipo *TBitmap* ou *TPicture.* Em seguida, é retirada a segurança de dados através da mensagem *RetirarSeguranca(aTextoSeguro)* enviada para uma instância da classe *TFsdrSegurancaDados* tanto por *SaveToStrings(Strings)* quanto por *SaveToBitmap(Bitmap)*, dependendo de quem o método da mensagem *AssignTo(Dest)* invocou.

A segurança nos dados é aplicada quando o usuário informa os dados no campo de edição e sai dele. Neste momento, a mensagem *Assign(Source)* é enviada a uma instância da classe *TFsdrBlobField.* A operação *Assign(Source)* verifica o tipo do parâmetro *Source* recebido e então envia a mensagem *LoadFromBlob(Blob)* para a própria instância se o parâmetro for do tipo *TBlobField*, ou envia a mensagem *LoadFromStrings(Strings)* se o parâmetro for do tipo *TStrings* ou então envia a mensagem *LoadFromBitmap(Bitmap)* se o parâmetro for do tipo *TBitMap*. Em seguida, a operação invocada delega os dados informados para receberem a segurança através do envio da mensagem *AplicarSeguranca(aTextoClaro)* para uma instância da classe *TFsdrSegurancaDados,* e por fim os dados já seguros são atualizados no dispositivo de armazenamento.

- **Interações entre as instâncias das classes *TFsdrSegurancaDados*, *TFsdrSigilo*, *TFsdrIntegridade* e *TFsdrDeposCertChave*.**

A figura 5.10 apresenta o diagrama de seqüência das instâncias de classes do framework Frasedare quando a mensagem *AplicarSeguranca(aTextoClaro)* é enviada para a instância de *TFsdrSegurancaDados*.

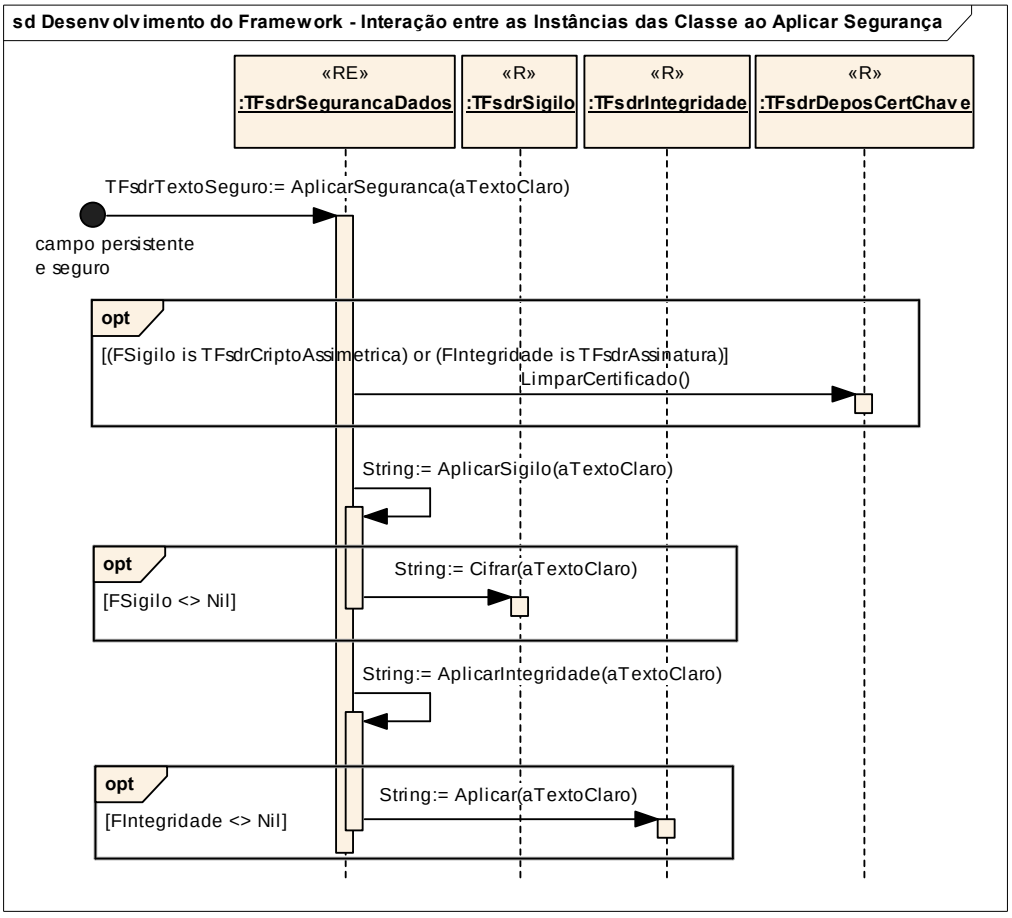

Figura 5.10 – Diagrama de seqüência das instâncias das classes do framework Frasedare ao aplicar segurança.

Uma leitura das interações da figura 5.10 pode ser feita da seguinte maneira: Quando uma instância de um campo persistente e seguro dos tipos *TFsdrMemoField, TFsdrStringField* ou *TFsdrBlobField* recebem dados, estas delegam a responsabilidade para uma instância da classe *TFsdrSegurancaDados*, enviando a mensagem *AplicarSeguranca(aTextoClaro).* A mensagem *LimparCertificado()* é enviada a uma instância do tipo *TFsdrDeposCertChave*, somente se for atendida a condição de que a propriedade *Sigilo* da classe *TFsdrSegurancaDados* esteja referenciando uma instância do tipo *TFsdrCriptoAssimetrica* ou quando a propriedade *Integridade* da classe *TFsdrSegurancaDados* esteja referenciando uma instância do tipo *TFsdrAssinatura.* Em seguida, é enviada a mensagem *AplicarSigilo(aTextoClaro)* para a instância da classe *TFsdrSegurancaDados*. Se a propriedade *Sigilo* estiver referenciando uma instância do tipo *TFsdrSigilo*, então é enviada a mensagem *Cifrar(aTextoClaro)* para a instância referenciada por *Sigilo*. Depois, a mensagem *AplicarIntegridade(aTextoClaro)* é enviada para a instância da classe *TFsdrSegurancaDados*. Se a propriedade *Integridade* estiver referenciando uma instância do tipo *TFsdrIntegridade*, a mensagem *Aplicar(aTextoClaro)* é enviada para a instância referenciada pela propriedade *Integridade*. Por fim, o resultado do texto sigilo e texto integridade é retornado através da estrutura de dados registro *TFsdrTextoSeguro*.

A figura 5.11 apresenta o diagrama de seqüência das instâncias de classes do framework Frasedare quando a mensagem *RetirarSeguranca(aTextoSeguro)* é enviada para a instância da classe *TFsdrSegurancaDados*.

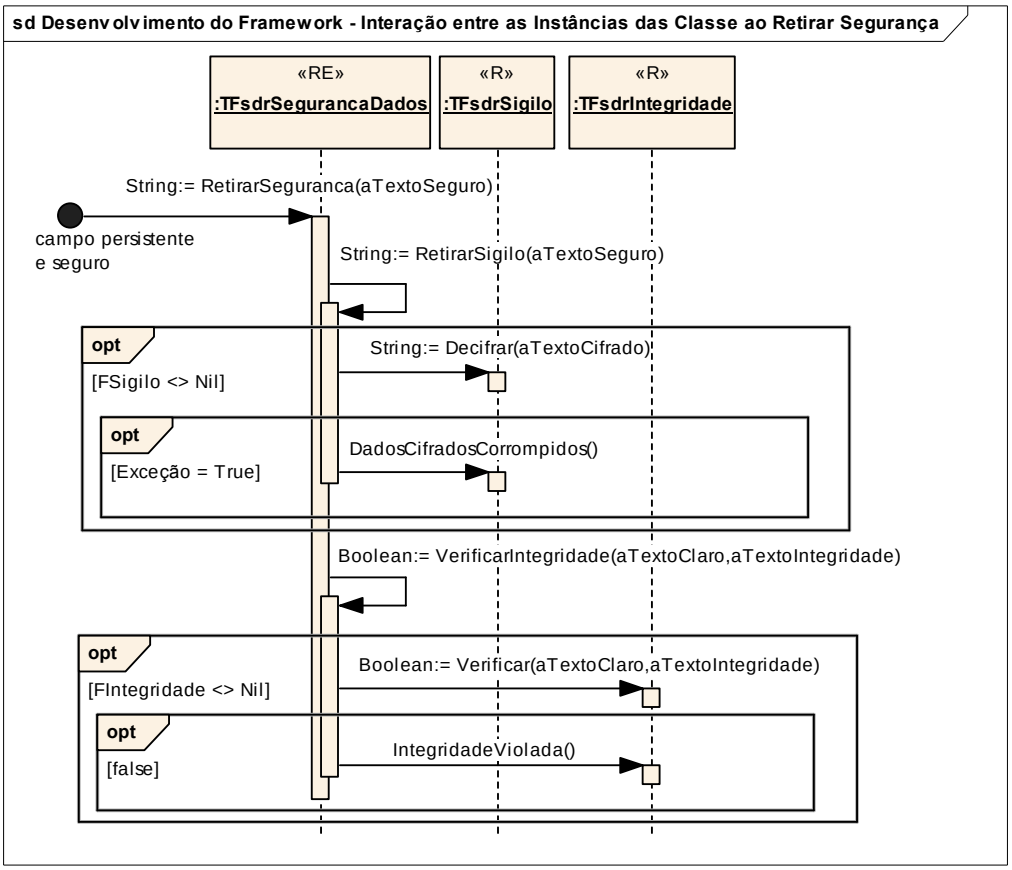

Figura 5.11 – Diagrama de seqüência das instâncias de classes do framework Frasedare ao retirar segurança.

Uma leitura das interações da figura 5.11 pode ser feita da seguinte maneira: Quando instâncias das classes de campo persistente e seguro recuperam os dados, estas delegam a responsabilidade para uma instância da classe *TFsdrSegurancaDados*, enviando a mensagem *RetirarSeguranca(aTextoSeguro).* Em seguida, é enviada a mensagem *RetirarSigilo(aTextoSeguro)*. Se a propriedade *Sigilo* estiver referenciando uma instância do tipo *TFsdrSigilo*, é enviada a mensagem *Decifrar(aTextoCifrado)* para a instância referenciada por *Sigilo*. Se ocorrer alguma exceção, é enviada a mensagem *DadosCifradosCorrompidos()* para a instância referenciada por *Sigilo.* Após, é enviada a mensagem *VerificarIntegridade(aTextoClaro, aTextoIntegridade)* para a instância de *TFsdrSegurancaDados*. Se a propriedade *Integridade* estiver referenciando uma instância do tipo *TFsdrIntegridade*, a mensagem *Verificar(aTextoClaro, aTextoIntegridade)* é enviada para esta instância. Caso retornar falso, isto indica que a

integridade dos dados foi violada, e com isto, é enviada a mensagem *IntegridadeViolada()* para a instância referenciada pela propriedade *Integridade.*

Os modelos dinâmicos apresentaram as interações entre as instâncias das classes do framework Frasedare ao aplicar a segurança e ao retirar a segurança.

5.1.5.4. Aplicação de Padrões de Projeto

Segundo GAMMA (1995), um benefício adicional é obtido quando o framework é documentado com os padrões de projeto. Quem conhece os padrões de projeto obtém rapidamente uma compreensão do framework. Mesmo quem não conhece pode se beneficiar da estrutura que os padrões de projeto emprestam à documentação do framework. A seção 2.2 apresenta padrões de projeto e suas contribuições na construção de frameworks.

A aplicação de padrões de projeto consiste em incluir uma estrutura de classes selecionada de um padrão de projeto na estrutura de classes do framework, ou ainda fazer com que classes do framework assumam responsabilidades correspondentes à estrutura de classes do padrão de projeto.

Abaixo serão apresentados os padrões de projetos aplicados no framework Frasedare.

- **Padrão de Projeto *Singleton*:**

O padrão de projeto *Singleton* garante que uma classe tenha somente uma instância e fornece um ponto global de acesso à mesma (GAMMA, 1995).

Este padrão de projeto foi utilizado na classe *TFsdrDeposCertChave*, pois havia a necessidade de somente uma única instância da classe para se conseguir utilizar um mesmo certificado digital selecionado, em dois mecanismos de criptografia que necessitassem de certificado digital. Isto ocorre quando o usuário do framework Frasedare combina os mecanismos de criptografia de chave assimétrica com assinatura digital. Neste caso, um mesmo certificado digital é utilizado para ambos os mecanismos

exatamente por referenciarem a mesma instância, fornecida pelo padrão de projeto *Singleton*.

Abaixo é apresentada a estrutura da classe *TFsdrDeposCertChave*.

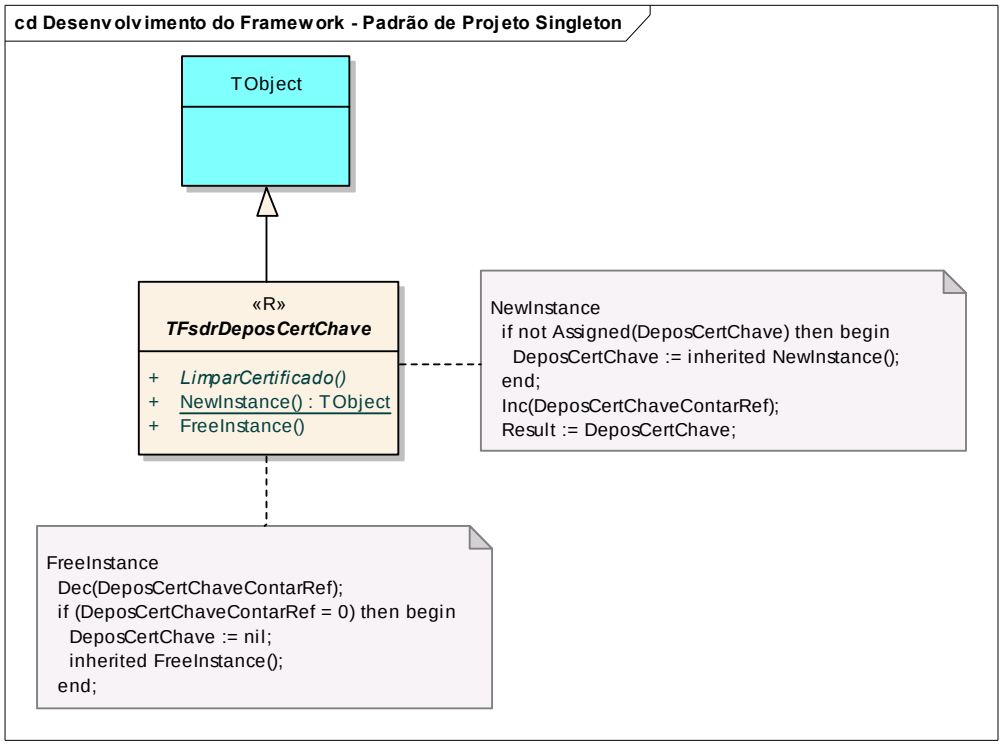

Figura 5.12 – Padrão de projeto *Singleton* aplicado no projeto do framework Frasedare.

A operação *NewInstance(): TObject,* é uma operação estática invocada pelo método construtor da classe. Seu método foi sobrescrito para verificar se uma instância da classe já existe quando for solicitada a criação de uma nova instância; caso exista, a instância anteriormente criada é devolvida, sem criar uma nova. Também acrescenta a um contador o número de referências que esta instância *Singleton* possui. A operação *FreeInstance()* é invocada pelo método destruidor da classe. O método verifica quantas referências a instância possui, destruindo a instância quando esta não possuir mais nenhuma referência. A linguagem *Object Pascal* do ambiente Delphi versão 5.0 não possui uma forma de definir atributos estáticos. Desta forma, para se implementar o padrão de projeto *Singleton*, utilizou-se variáveis locais de unidade.

- **Padrão de Projeto Método *Template*:**

A figura 5.13 apresenta todos os casos onde o padrão de projeto método *Template* foi utilizado no framework Frasedare.

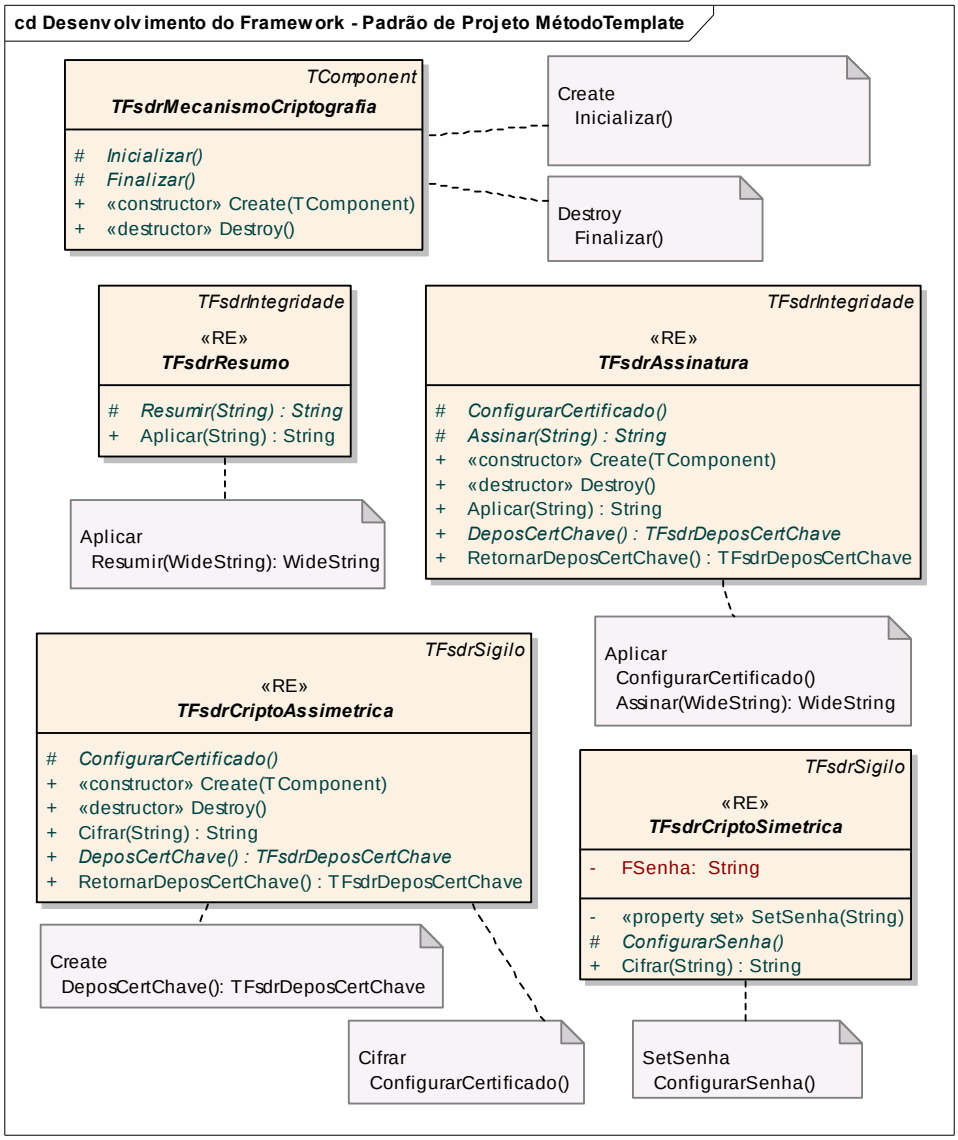

Figura 5.13 – Padrão de projeto Método *Template* aplicado no projeto do framework Frasedare.

Um método *Template* define o esqueleto de um algoritmo em um método, postergando alguns passos para subclasses sem mudar a estrutura do mesmo (GAMMA, 1995). Também segundo LARMAN (2004), um método *Template* define um método

gabarito em uma superclasse estabelecendo um esqueleto de algoritmo, com suas partes variantes e invariantes. O método gabarito chama outras operações, algumas das quais são operações que podem ter seus métodos sobrepostos em uma subclasse. Assim, as subclasses podem sobrepor os métodos variantes para adicionar seu próprio comportamento, exclusivo em pontos de variabilidade (flexibilidade). Um framework faz intenso uso de métodos *Template*, por incluir pontos de flexibilidade.

As notas ligadas às classes apresentadas na figura 5.13 mostram a operação que é implementada por um método *Template* contendo as operações que são invocadas pelo método *Template*, chamada de *Hook.*

- **Padrão de Projeto *Factory Method*:**

No projeto do framework Frasedare, foi utilizado este padrão de projeto em duas classes, conforme apresentado na figura 5.14.

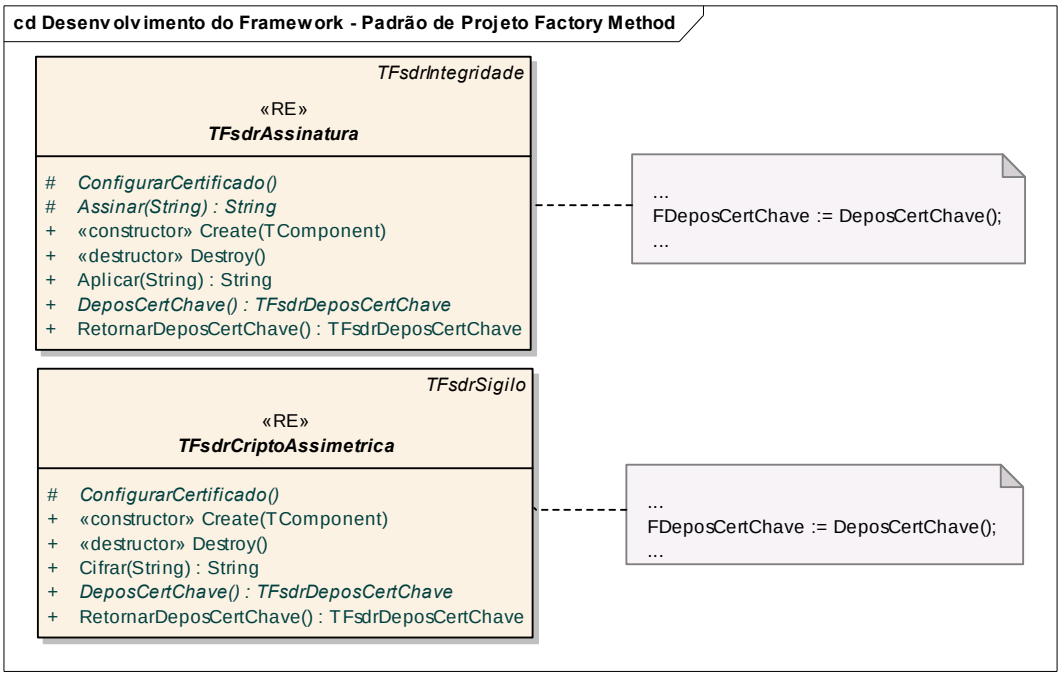

Figura 5.14 – Padrão de projeto *Factory Method* aplicado no projeto do framework Frasedare.

Segundo GAMMA (1995), a intenção do padrão de projeto *Factory Method* é "definir uma interface para criar um objeto, mas deixar as subclasses decidirem que classe instanciar".

O método da operação *Create(TComponent)* invoca a operação abstrata *DeposCertChave()* que deve ser implementada através de um método nas subclasses. Esta operação abstrata deverá ser implementada nas subclasses por um método que retorne uma instância do tipo *TFsdrDeposCertChave*. No framework Frasedare, fazia-se necessário existir uma referência para uma instância do tipo *TFsdrDeposCertChave*, porém não era conhecida a implementação da classe desta instância. Desta forma, foi incluído o padrão de projeto *Factory Method* para resolver este problema, sendo esta operação implementada por um método nas subclasses concretas.

A apresentação dos padrões de projeto utilizados no framework Frasedare serve como uma forma de documentação para o projeto.

5.1.6. Implementação do Framework Frasedare

A implementação do framework Frasedare foi feita através da codificação das classes abstratas e concretas do projeto do framework Frasedare em uma linguagem de programação. O framework Frasedare foi implementado na linguagem de programação *Object Pascal* contendo classes abstratas e concretas. Para utilizar o framework Frasedare, deve-se estender subclasses das classes abstratas definidas, nas quais as operações abstratas passarão à concretas com a implementação destas através dos seus devidos métodos.

O apêndice III contém a implementação do framework Frasedare.

5.1.7. Teste do Framework Frasedare

No teste de frameworks, é avaliada a usabilidade e é determinado se o framework proporciona as pretendidas funcionalidades. Para avaliar a usabilidade do framework Frasedare, por ele ser um framework de subsistema, foi desenvolvido um subsistema baseado no framework. A avaliação deste subsistema foi feita

desenvolvendo-se uma aplicação teste que utiliza este subsistema, podendo-se assim identificar onde o framework Frasedare precisava ser re-projetado.

5.2. Fase de Uso do Framework Frasedare

A fase de uso de frameworks tem como resultado principal uma aplicação ou subsistema desenvolvido reutilizando-se um ou mais frameworks.

Como o framework Frasedare é um framework de subsistema, o seu uso resulta em um subsistema. Uma vez construído o subsistema, este poderá ser utilizado por uma aplicação para acrescentar as funcionalidades de segurança de dados em repouso proporcionadas pelo framework Frasedare. A construção do subsistema ocorre estendendo as classes do framework Frasedare e também incluindo as classes concretas fornecidas.

5.2.1. Como Utilizar o Framework Frasedare

Para se utilizar o framework Frasedare, algumas etapas são necessárias. Primeiramente deve-se construir o subsistema, definindo-se quais classes concretas do framework Frasedare serão necessárias e quais subclasses concretas deverão ser implementadas.

Definido o subsistema, este poderá ser utilizado por aplicações. Para tanto, deve-se instalar as classes do subsistema no ambiente de desenvolvimento, e elas ficarão disponíveis para uso das aplicações. O uso pelas aplicações é feito através de composição das classes do subsistema.

Para se definir quais os serviços de segurança que o subsistema proverá deve-se identificar os requisitos desejados. A tabela 5.1 na seção 5.1.5 pode auxiliar na definição dos serviços de segurança, identificando assim quais mecanismos de criptografia deverão ser implementados no subsistema, e suas possíveis combinações baseadas nas categorias.

Uma vez identificados os mecanismos de criptografia que se deseja utilizar, deve-se também observar os pré-requisitos exigidos e necessários para cada mecanismo de criptografia específico, conforme apresentado na tabela 5.2.

Mecanismo de Criptografia	**Pré-Requisitos**	**Exemplos de Algoritmos**
Criptografia de Chave Simétrica	- Uma chave secreta deve ser informada ao se usar este mecanismo de criptografia. - O campo, coluna ou atributo do dispositivo de armazenamento deverá considerar um tamanho maior do que normalmente é utilizando para um texto claro.	Blowfish, DES, 3DES, IDEA, RC5 e AES.
Criptografia de Chave Assimétrica	- Recursos para o gerenciamento de certificados digitais. - Chave pública para cifrar a chave secreta da criptografia de chave simétrica e chave privada para decifrar, ou seja, quem necessitar da informação deverá possuir a chave privada correspondente à chave pública. - O campo, coluna ou atributo do dispositivo de armazenamento deverá considerar um tamanho maior do que normalmente é utilizando para um texto claro.	DH, ECC e o RSA.
Resumo de Dados	- Necessário criar no dispositivo de armazenamento um campo, coluna ou atributo a mais para armazenar o texto integridade. Este campo, coluna ou atributo sempre terá o mesmo tamanho, determinado pelo algoritmo selecionado.	MD2, MD5 e SHA-1.
Assinatura Digital	- Recursos para o gerenciamento de certificados digitais. - Chave privada para assinar e chave pública para verificar a assinatura (texto integridade). Assim, quem for verificar a assinatura, deverá possuir um certificado com a chave pública. - Deve-se criar no dispositivo de armazenamento um campo, coluna ou atributo a mais para armazenar o texto integridade. Este campo pode variar de tamanho.	DSA, ECDSA e o RSA.

Tabela 5.2 – Pré-requisitos ao se utilizar um mecanismo de criptografia e exemplos de algoritmos.

5.2.1.1. Construindo o Subsistema

Partindo-se da identificação dos requisitos e dos mecanismos de criptografia, são selecionadas as classes concretas que serão utilizadas e quais subclasses concretas deverão ser estendidas do framework Frasedare.

A figura 5.15 apresenta o diagrama de classes contendo as classes concretas do framework Frasedare.

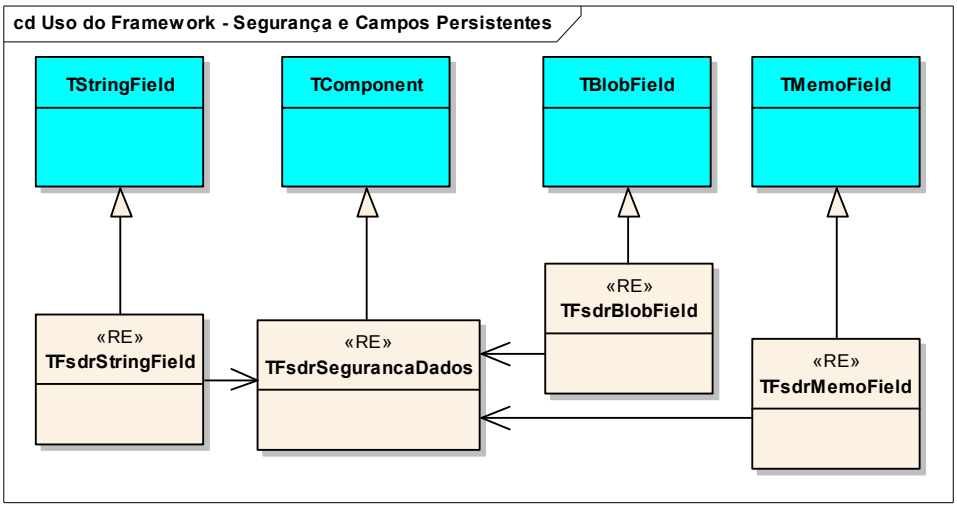

Figura 5.15 – Diagrama de classes com as classes concretas do framework Frasedare.

Ao se construir o subsistema estendendo o framework Frasedare, a classe *TFsdrSegurancaDados* sempre deverá ser utilizada. Esta classe é essencial e poderá ser redefinível conforme estabelecido pelo seu estereótipo.

As classes *TFsdrStringField, TFsdrMemoField* e *TFsdrBlobField* poderão ser utilizadas todas ao mesmo tempo ou somente uma, apesar de todas serem essenciais. No projeto, foram definidas como essenciais, pois pelo menos uma delas deverá sempre ser utilizada. Também poderão ser redefiníveis, conforme estabelecido pelos seus estereótipos.

A figura 5.16 apresenta um diagrama de classes com todas as classes e respectivas subclasses concretas (mais escuras) que implementam os mecanismos de criptografia. Porém, nem todas as classes são necessárias para se construir o subsistema. Somente são implementadas as subclasses de classes do framework Frasedare conforme

os mecanismos de criptografia identificados como necessários para o subsistema, como descrito anteriormente.

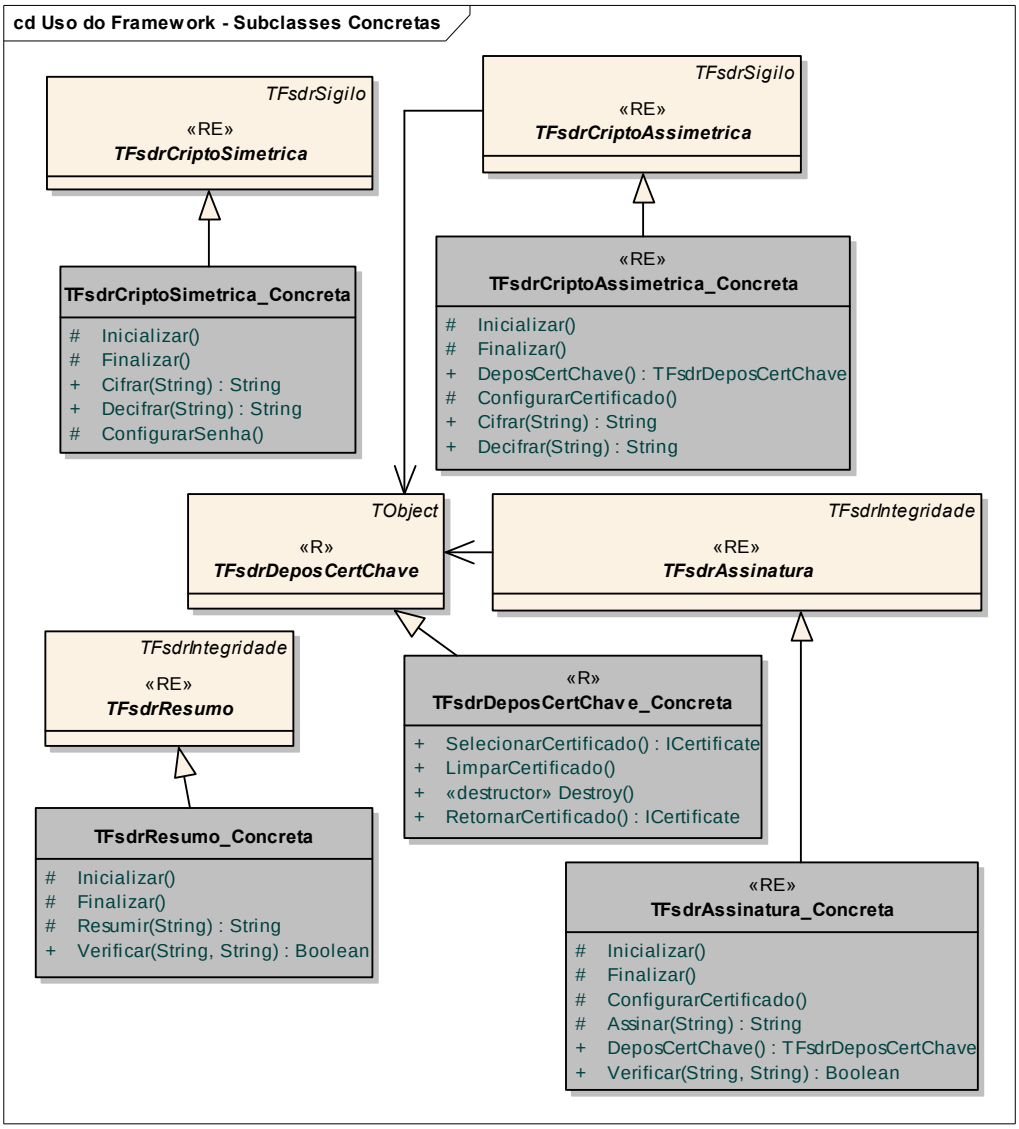

Figura 5.16 – Diagrama de classes contendo as classes e suas subclasses concretas definidas para o uso do framework Frasedare.

Para se implementar os mecanismos de criptografia nas subclasses, as classes do framework Frasedare fornecem um conjunto de operações que deverão ser implementadas por métodos específicos de algoritmos de criptografia ou invocar APIs de criptografia que implementem algoritmos de criptografia.

Dependendo das classes de mecanismos de criptografia do framework Frasedare identificadas e selecionadas para a construção do subsistema, as seguintes operações devem ser implementadas por seus respectivos métodos.

- Subclasse de *TFsdrCriptoSimetrica* deverá implementar:
  - *Inicializar()*
  - *Finalizar()*
  - *Cifrar(String): String*
  - *Decifrar(String): String*
  - *ConfigurarSenha()*
- Subclasse de *TFsdrCriptoAssimetrica* deverá implementar:
  - *Inicializar()*
  - *Finalizar()*
  - *DeposCertChave(): TFsdrDeposCertChave*
  - *ConfigurarCertificado()*
  - *Cifrar(String): String*
  - *Decifrar(String): String*
- Subclasse de *TFsdrAssinatura* deverá implementar:
  - *Inicializar()*
  - *Finalizar()*
  - *ConfigurarCertificado()*
  - *Assinar(String): String*
  - *DeposCertChave(): TFsdrDeposCertChave*
  - *Verificar(String, String): Boolean*
- Subclasse de *TFsdrResumo* deverá implementar:
  - *Inicializar()*
  - *Finalizar()*

- *Resumir(String): String*
- *Verificar(String, String): Boolean*

- Subclasse de *TFsdrDeposCertChave* deverá implementar:
  - *SelecionarCertificado(): ICertificate*
  - *LimparCertificado()*
  - *Destroy()*
  - *RetornarCertificado(): ICertificate*

A invocação das operações implementadas nas subclasses é feita através da interação entre as instâncias de classes do framework Frasedare, não sendo necessária a preocupação quanto à invocação das operações implementadas. Esta característica é chamada de princípio de *Hollywood*, conforme descrito na seção 2.5.

Assim, uma vez selecionadas as classes concretas, conforme o mecanismo de criptografia identificado e implementadas as subclasses do framework Frasedare, o subsistema está construído estendendo o framework Frasedare.

5.2.1.2. Utilizando o Subsistema

Como o projeto do framework Frasedare foi implementado utilizando o ambiente Delphi, as classes do framework Frasedare estenderam a superclasse de componente - *TComponent*. Estas classes componentes do framework Frasedare são classes concretas ou subclasses que estenderam o framework Frasedare na construção do subsistema. O ambiente Delphi proporciona todo um mecanismo para o gerenciamento de classes componentes, e para serem gerenciadas pelo ambiente, é necessária a instalação destas classes componentes no ambiente.

Desta forma, para se utilizar o subsistema construído estendo o framework Frasedare, são necessários alguns passos. Primeiramente, se deve instalar as classes do subsistema. Após a instalação, estas ficam disponíveis no ambiente, podendo ser utilizadas por qualquer aplicação.

Como já adotado no projeto e também por motivo de simplificação, as classes componentes do framework Frasedare serão apenas chamadas de classes.

a) Instalando as classes do subsistema

A instalação se procede da mesma forma quando se instala qualquer outra classe componente no ambiente Delphi. A figura 5.17 apresenta as classes do subsistema instaladas na paleta do ambiente Delphi.

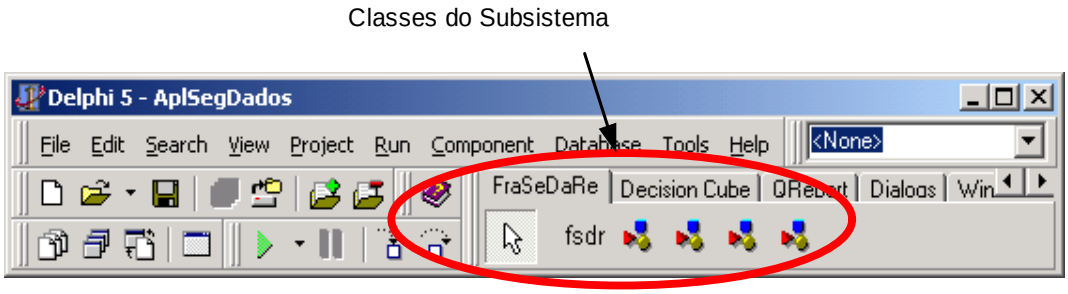

Figura 5.17 – Classes do subsistema no ambiente Delphi.

Após a instalação, a classe *TFsdrSegurancaDados* é apresentada na paleta e as classes de campo persistente e seguro *TFsdrStringField, TFsdrMemoField* e *TFsdrBlobField* permanecem instaladas como um tipo de componente especial, visualizado somente na tela de criação de campo persistente do ambiente Delphi.

As subclasses de *TFsdrCriptoSimetrica*, *TFsdrCriptoAssimetrica*, *TFsdrAssinatura* e *TFsdrResumo* que implementam os mecanismos de criptografia também ficam instaladas na paleta do ambiente Delphi.

A classe que aparece com o rótulo de *fsdr*, na figura 5.17 é a classe *TFsdrSegurancaDados*; as demais são classes que implementam mecanismos de criptografia.

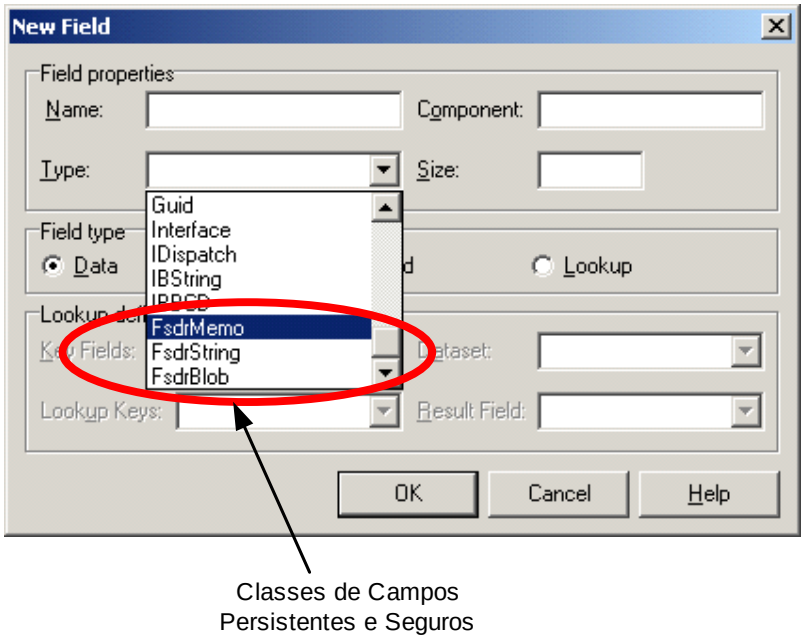

Figura 5.18 – Classes de campo persistente e seguro do subsistema no ambiente Delphi.

A figura 5.18 apresenta as classes de campo persistente. Estas classes, quando instaladas, são encontradas na opção *type* da tela de criação de campo persistente.

As classes que aparecem na figura como *FsdrMemo, FsdrString* e *FsdrBlob* são as classes *TFsdrMemoField, TFsdrStringField* e *TFsdrBlobField* do subsistema, respectivamente.

b) Utilizando as classes do subsistema em uma aplicação

Uma vez instaladas, as classes ficam disponíveis no ambiente Delphi para uso das aplicações. Alguns passos precisam ser executados para se utilizar o subsistema em uma aplicação.

O primeiro passo é inserir na aplicação uma referência para as classes *TFsdrSegurancaDados* e classes dos mecanismos de criptografia, fazendo a associação entre ambas. A figura 5.19 apresenta a tela de uma aplicação utilizando o subsistema e outra tela do *Object Inspector* apresentando as propriedades da classe *TFsdrSegurancaDados*.

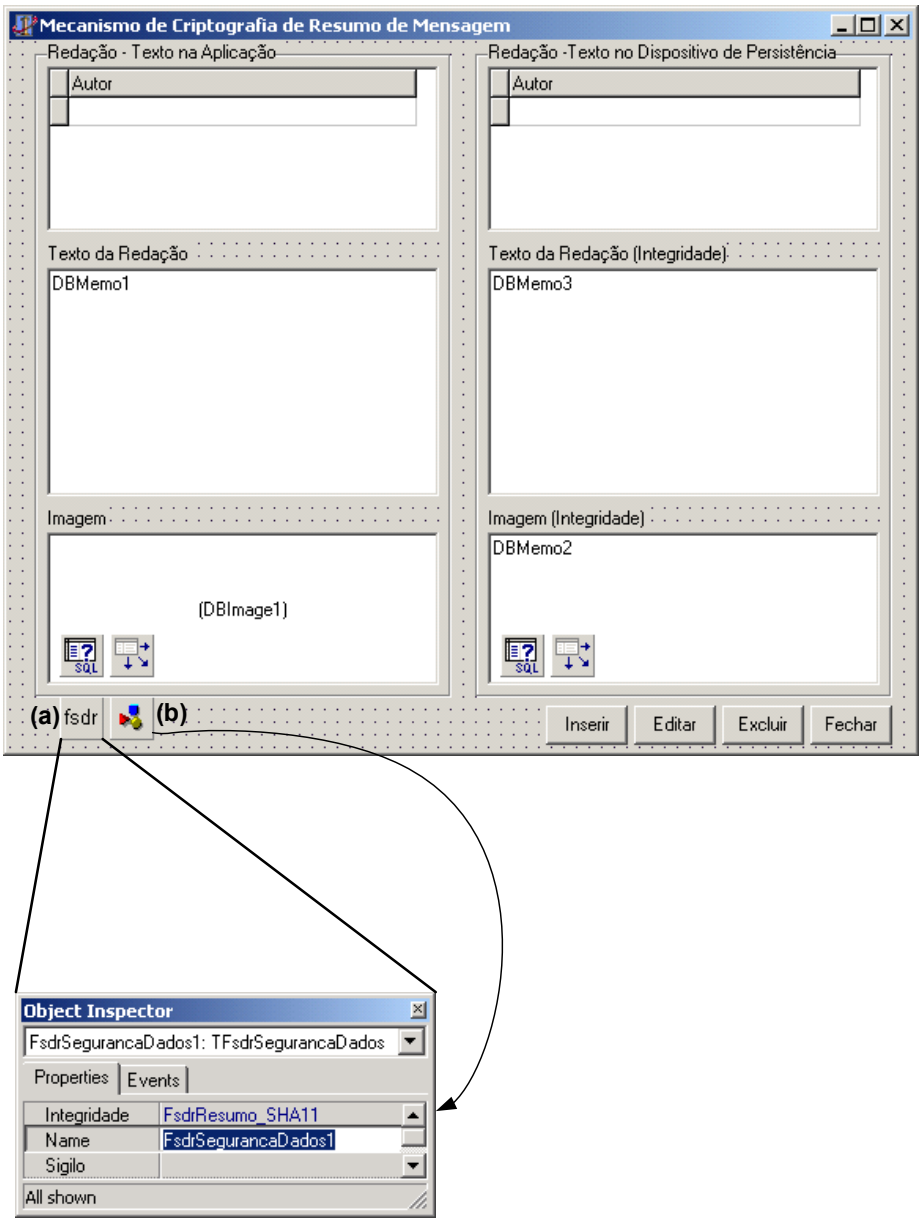

Figura 5.19 – Uso do subsistema em uma aplicação.

A figura 5.19, apresenta em (a) um componente do tipo *TFsdrSegurancaDados*. Em (b) apresenta um componente do tipo *TFsdrResumo*. Para (a), uma janela *Object Inspector* é demonstrada, apresentando as propriedades das categorias de segurança de

Integridade e Sigilo. Para este exemplo apresentado, existe uma referência através da categoria de Integridade para uma classe do tipo *TFsdrResumo*. Desta forma, o serviço de segurança obtido por esta aplicação que está utilizando o subsistema estendido do framework Frasedare, é a integridade, por estar utilizando um mecanismo de criptografia de resumo de dados, implementado em uma subclasse do tipo *TFsdrResumo.*

O próximo passo é criar uma referência para classe de campo persistente e seguro *TFsdrMemoField*, *TFsdrStringField* ou *TFsdrBlobField* que representará um campo ou coluna no dispositivo de armazenamento. As principais propriedades da classe de campo persistente e seguro são o *FieldName*, *SegurancaDados* e *DataFieldIntegridade*. *FieldName* referencia o nome de um campo ou coluna correspondente no dispositivo de armazenamento, *SegurancaDados* deverá possuir uma referência para uma instância da classe *TFsdrSegurancaDados*, onde considerando a figura 5.19, corresponde a (a). A propriedade *DataFieldIntegridade* deverá possuir uma referência a um campo ou coluna correspondente no dispositivo de armazenamento que possibilite armazenar o texto de integridade (resumo de mensagem ou assinatura digital) quando for utilizado um mecanismo de criptografia da categoria Integridade.

A figura 5.20 apresenta as principais propriedades quanto à segurança de um campo persistente e seguro. Em (a) é apresentada uma classe do tipo *TDataSet*, que representa um arquivo ou tabela de um dispositivo de armazenamento. É nesta classe que deve ser criado o campo persistente e seguro. (b) é uma referência para uma instância da classe do tipo *TFsdrSegurancaDados* que deve ser referenciada pela instância de campo persistente e seguro criado em (a).

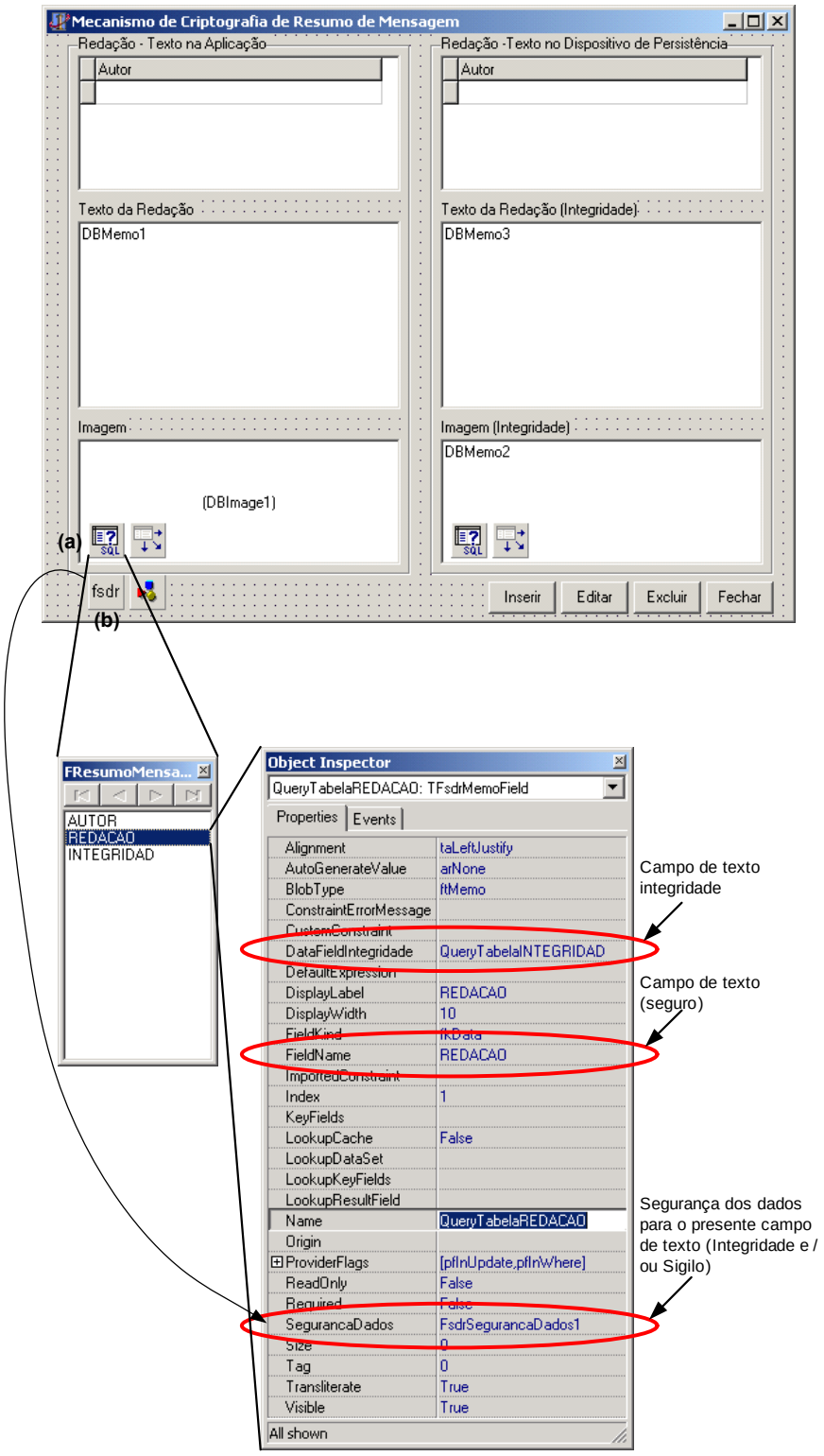

Figura 5.20 – Uso do subsistema em uma aplicação. Principais propriedades de um campo persistente e seguro.

Assim, a utilização do subsistema em uma aplicação é feita através do uso das classes componentes do subsistema instaladas no ambiente Delphi, conforme apresentado acima.

5.2.2. Utilizando o Framework Frasedare em uma Aplicação Teste

No decorrer da construção do framework Frasedare foi utilizada uma aplicação teste que explorou as possibilidades de segurança de dados em repouso proposto pelo framework Frasedare. Para tanto, conforme ocorria o desenvolvimento do framework Frasedare, também se construía um subsistema estendendo o framework Frasedare implementando-se os mecanismos de criptografia, e este subsistema era utilizado pela aplicação teste.

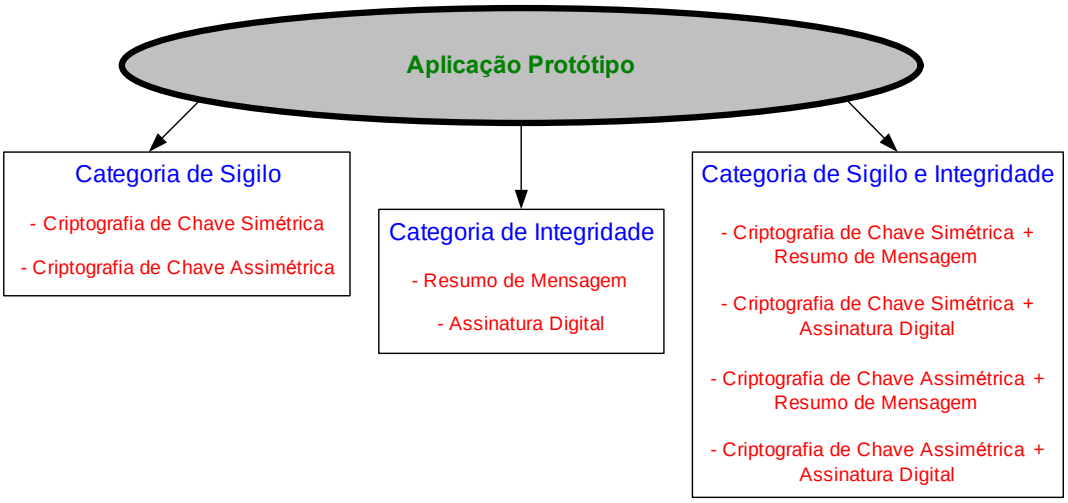

Figura 5.21 – Mecanismos de criptografia utilizados no subsistema para a aplicação teste.

A figura 5.21 apresenta os mecanismos de criptografia implementados pelo subsistema e utilizados na aplicação teste para o desenvolvimento do framework Frasedare, considerando as categorias de sigilo e integridade isoladamente e combinadas.

5.2.2.1. Construindo o Subsistema para Aplicação Teste

Conforme apresentado na seção 5.2.1.1, para fazer uso do framework Frasedare, deve-se primeiramente implementar as subclasses concretas estendidas de classes do framework Frasedare.

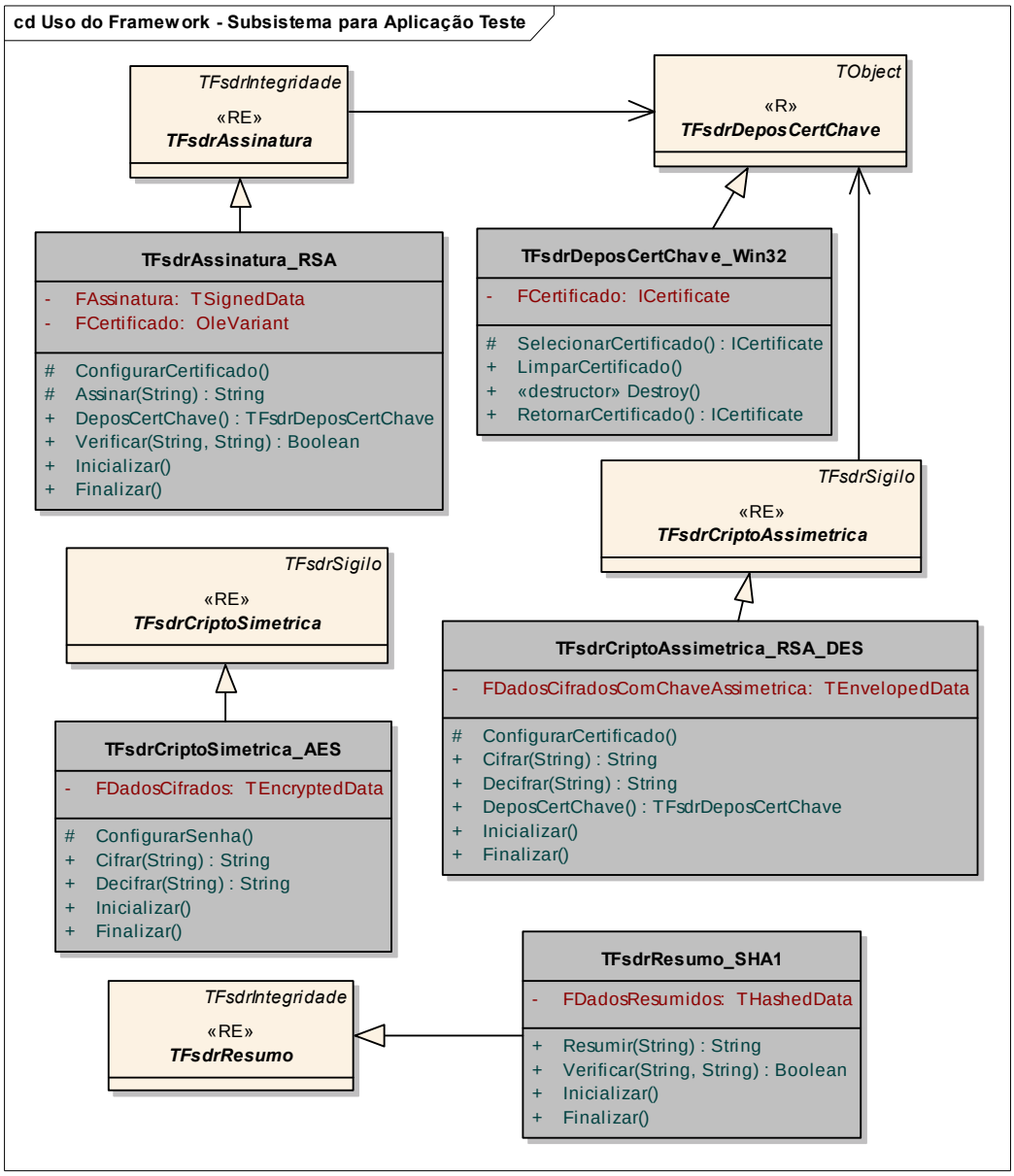

Figura 5.22 – Diagrama de classes contendo as classes do framework Frasedare e subclasses concretas estendidas para o subsistema da aplicação teste.

A figura 5.22 apresenta as subclasses estendidas do framework Frasedare para implementação do subsistema utilizado pela aplicação teste. Os métodos que implementam as operações nas subclasses utilizam APIs de criptografia (CAPI) fornecidas pela biblioteca de classe CAPICOM. Segue abaixo a descrição da responsabilidade atribuída a cada subclasse estendida:

- ***TFsdrDeposCertChave_Win32*** – É uma subclasse concreta de *TFsdrDeposCertChave* do framework Frasedare. Implementa as APIs de criptografia que fazem acesso ao depósito de certificados e chaves do sistema operacional Windows.
- ***TFsdrCriptoSimetrica_AES*** – É uma subclasse concreta de *TFsdrCriptoSimetrica* do framework Frasedare. Implementa as APIs de criptografia que fornecem o mecanismo de criptografia de chave simétrica AES.
- ***TFsdrCriptoAssimetrica_RSA_DES*** – É uma subclasse concreta de *TFsdrCriptoAssimetrica* do framework Frasedare. Implementa as APIs de criptografia que fornecem o mecanismo de criptografia de chave assimétrica RSA e DES.
- ***TFsdrAssinatura_RSA*** – É uma subclasse concreta de *TFsdrAssinatura* do framework Frasedare. Implementa as APIs de criptografia que fornecem o mecanismo de assinatura digital RSA.
- ***TFsdrResumo_SHA1*** – É uma subclasse concreta de *TFsdrResumo* do framework Frasedare. Implementa as APIs de criptografia que fornecem o mecanismo de resumo de dados SHA1.

Além da implementação das subclasses, também foram incorporadas ao subsistema as classes concretas *TFsdrSegurancaDados, TFsdrMemoField, TFsdrStringField* e *TFsdrBlobField* do framework Frasedare.

O apêndice IV apresenta a implementação das subclasses concretas do framework Frasedare definidas para o subsistema utilizado pela aplicação teste.

5.2.2.2. Utilizando o Subsistema na Aplicação Teste

Conforme apresentado na seção 5.2.1.2., após a construção do subsistema, suas classes devem ser instaladas no ambiente Delphi para que fiquem disponíveis para uso de uma aplicação. Nesta seção será apresentada uma descrição sobre o uso do subsistema estendido do framework Frasedare pela aplicação teste.

A aplicação teste utilizou o subsistema que implementou todos os mecanismos de criptografia definidos no framework Frasedare e a combinação destes. Assim, para melhor ilustrar o uso do subsistema na aplicação teste, algumas interfaces de usuário que foram implementadas na aplicação teste serão mostras.

A figura 5.23 apresenta a tela principal da aplicação teste que faz a chamada das oito possibilidades de se aplicar segurança usando-se um subsistema que estende o framework Frasedare.

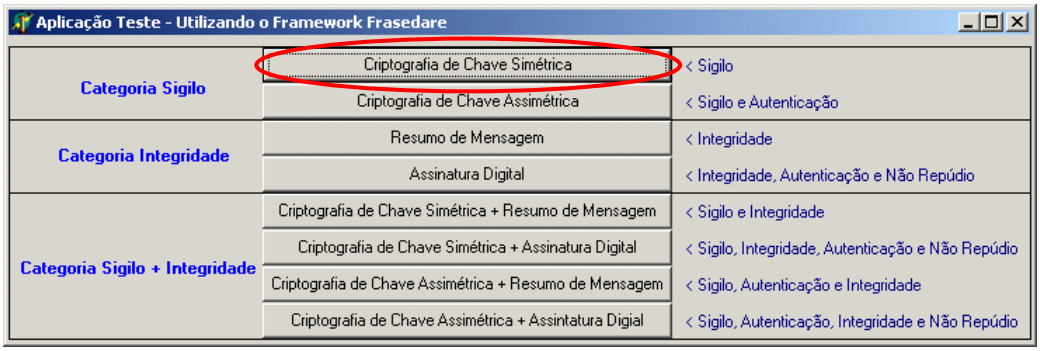

Figura 5.23 – Tela principal da aplicação teste.

Na tela principal da aplicação teste, pode-se observar os botões que chamam as possibilidades para se aplicar segurança, com destaque para o botão circulado. Este botão chama uma tela que faz acesso a um dispositivo de armazenamento. Entre a apresentação dos dados e o dispositivo de armazenamento está o controle da segurança de dados em repouso, baseado no mecanismo de criptografia de chave simétrica, implementado através do framework Frasedare.

Utilizando os recursos da orientação a objetos, proporcionados pela linguagem *Object Pascal* do ambiente Delphi, foram definidas três telas bases, uma para implementar a categoria de mecanismos de criptografia de sigilo, outra para a categoria de integridade e uma terceira para a combinação das categorias de sigilo e integridade.

Depois, para implementar cada mecanismo de criptografia individualmente e combinados, foram estendidas estas telas bases e definidas novas telas, sendo que estas novas telas possuíam uma implementação específica de segurança. Por exemplo, uma tela para mecanismo de criptografia de chave simétrica estenderia a tela base de sigilo. Uma tela para mecanismos de criptografia de chave simétrica e assinatura digital, estenderia a tela base de sigilo e integridade, e assim por diante.

A seguir, serão apresentadas as telas bases utilizadas na aplicação teste, com o objetivo de proporcionar um melhor entendimento quanto ao uso do framework Frasedare.

- **Uso dos Mecanismos da Categoria Sigilo**

A figura 5.24 apresenta a tela base da aplicação teste utilizando o subsistema que implementa os mecanismos de criptografia da categoria sigilo.

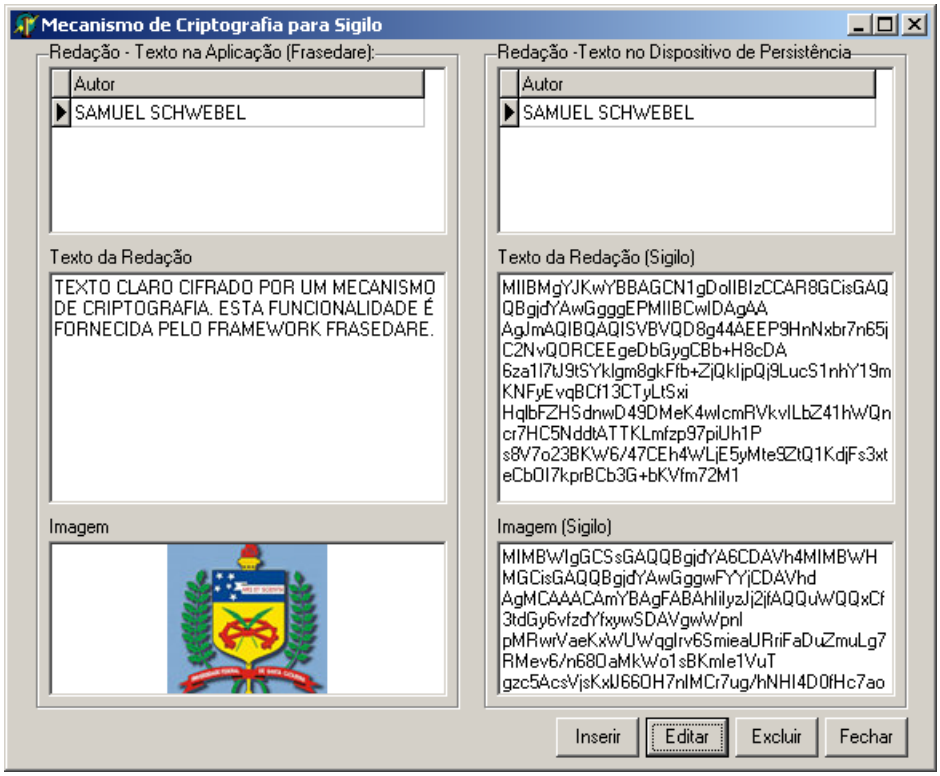

Figura 5.24 – Tela base da aplicação teste para mecanismos de criptografia da categoria de sigilo.

Os mecanismos de criptografia de chave simétrica e de criptografia de chave assimétrica foram implementados cada um em sua respectiva tela que são subclasses desta tela base.

O campo de edição de texto e o campo de imagem localizados no lado esquerdo inferior, rotulados respectivamente como "*Texto da Redação*" e "*Imagem*", apresentam o texto digitado e a imagem adicionada pelo usuário. Os campos do lado direito inferior, rotulados respectivamente como "*Texto da Redação (Sigilo)*" e "*Imagem (Sigilo)*", apresentam o texto e a imagem cifrados pelo mecanismo de criptografia implementado no subsistema que estendeu o framework Frasedare, o qual a aplicação teste utilizou.

- **Uso dos Mecanismos da Categoria Integridade**

A figura 5.25 apresenta a tela base da aplicação teste que utiliza os mecanismos de criptografia da categoria de integridade.

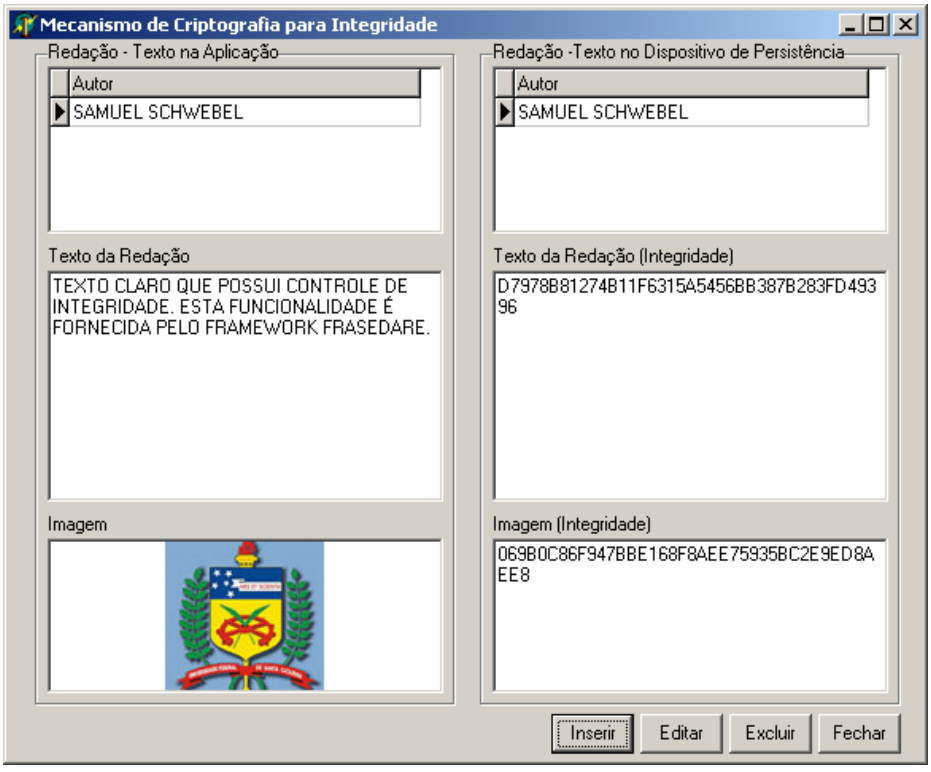

Figura 5.25 – Tela base da aplicação teste para mecanismos de criptografia da categoria integridade.

Os mecanismos de criptografia de resumo de dados e o de assinatura digital foram implementados cada um em sua respectiva tela que são subclasses desta tela base.

O campo de edição de texto e o campo de imagem do lado esquerdo inferior, rotulados respectivamente como "*Texto da Redação*" e "*Imagem*", apresentam o texto digitado e a imagem adicionada pelo usuário. Os campos do lado direito inferior, rotulados respectivamente como "*Texto da Redação (Integridade)*" e "*Imagem (Integridade)*", apresentam o texto integridade (resumo de dados ou assinatura digital) gerado pelo mecanismo de criptografia implementado no subsistema que estendeu o framework Frasedare, o qual a aplicação teste utilizou.

- **Uso dos Mecanismos da Categoria Sigilo e Integridade**

A figura 5.26 apresenta a tela base da aplicação teste que utiliza os mecanismos de criptografia da categoria de sigilo e integridade.

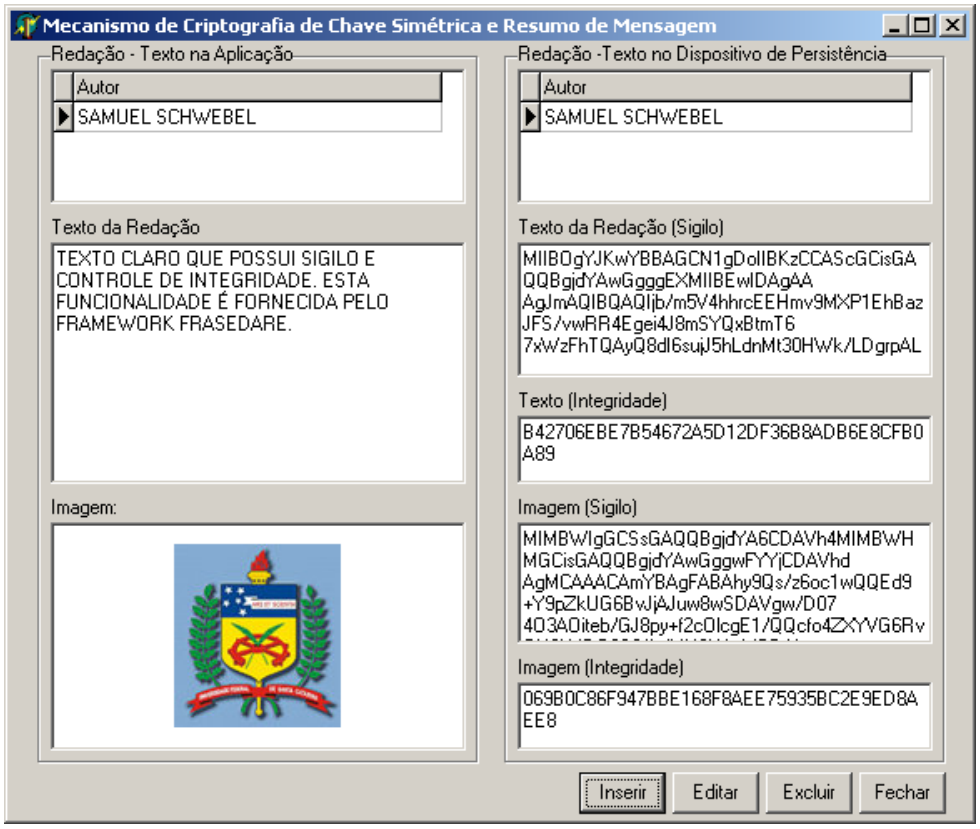

Figura 5.26 – Tela base da aplicação teste para mecanismos de criptografia que combinam as categorias de sigilo e integridade.

As combinações dos mecanismos de criptografia das categorias de sigilo e integridade foram implementadas cada uma em sua respectiva tela que são subclasses desta tela base.

O campo de edição de texto e o campo de imagem do lado esquerdo inferior, rotulados respectivamente como "*Texto da Redação*" e "*Imagem*", apresentam o texto digitado e a imagem adicionada pelo usuário. Segundo uma ordem descendente do lado direito inferior, o primeiro campo, rotulado como "*Texto da Redação (Sigilo)*", apresenta o texto cifrado da redação, e o segundo campo, rotulado como "*Texto (Integridade)*", apresenta o texto integridade da redação. Seguindo a ordem, o campo rotulado como "*Imagem (Sigilo)*" apresenta a imagem cifrada e o quarto campo, rotulado como "*Imagem (Integridade)*" apresenta o texto integridade gerado para a imagem, todos gerados pelos mecanismos de criptografia implementados pelo subsistema construído estendendo o framework Frasedare.

A figura 5.27 apresenta a tela de seleção de certificados digitais. Esta é apresentada pelo framework Frasedare quando um subsistema que o estendeu é requisitado a utilizar um mecanismo de criptografia que necessite de certificados digitais, como por exemplo, assinatura digital ou criptografia de chave assimétrica.

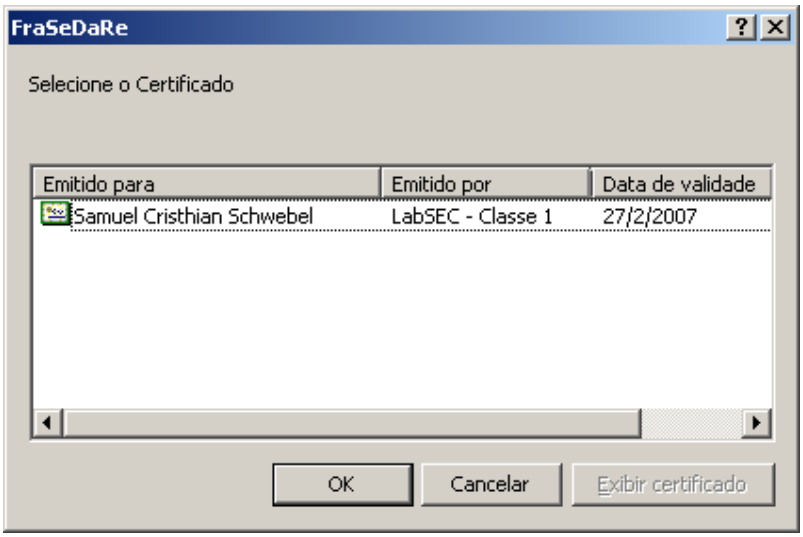

Figura 5.27 – Tela para seleção de certificados digitais.

Com a aplicação teste, puderam-se testar todas as funcionalidades propostas pelo framework Frasedare, além disso, ao fazer uso do framework Frasedare, esta aplicação identificou necessidades de melhorias quanto à flexibilidade e extensibilidade do projeto do framework, as quais foram analisadas e implementadas.

5.2.3. Utilizando o Framework Frasedare no Sistema Blendus

O framework Frasedare foi utilizado na funcionalidade de autorização de procedimentos médicos do sistema Blendus. A seção 5.1.3 apresenta maiores detalhes sobre a funcionalidade e o sistema Blendus.

Antes da construção do subsistema, foram identificados os requisitos de segurança para a funcionalidade de autorização de procedimentos médicos do sistema Blendus, mais especificamente o parecer da perícia nesta funcionalidade. Desta forma, puderam-se identificar quais os serviços de segurança seriam necessários para a aplicação.

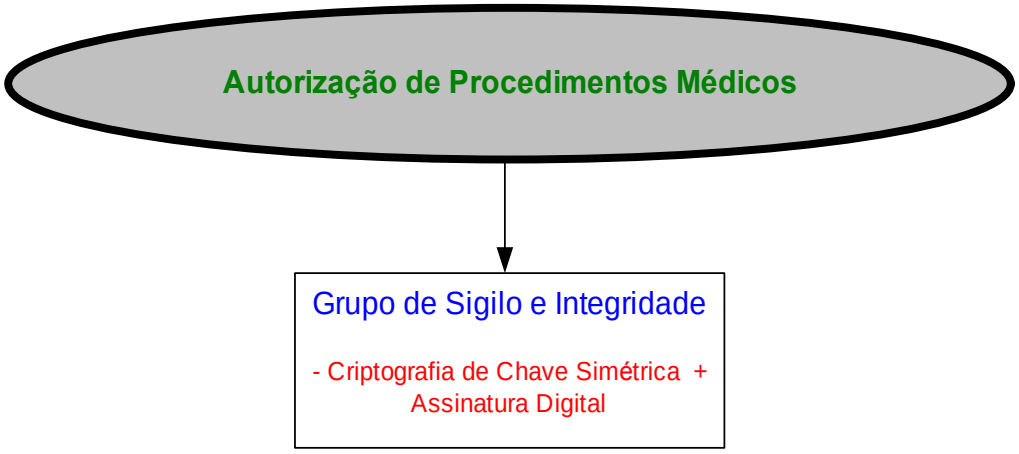

Figura 5.28 – Mecanismos de criptografia utilizados na autorização de procedimentos médicos.

Com as informações de serviços de segurança identificadas, foram definidos quais mecanismos de criptografia seriam implementados estendo o framework Frasedare, construindo-se assim o subsistema. A figura 5.28 apresenta os mecanismos de criptografia utilizados na funcionalidade de autorização de procedimentos médicos do sistema Blendus.

5.2.3.1. Construindo o Subsistema para o Sistema Blendus

Conforme apresentado na seção 5.2.1.1, o uso do framework Frasedare ocorre ao se estender suas classes para construção de um subsistema.

A figura 5.29 apresenta as subclasses estendidas do framework Frasedare para a construção do subsistema que foi utilizado pela funcionalidade de autorização de procedimentos médicos no sistema Blendus.

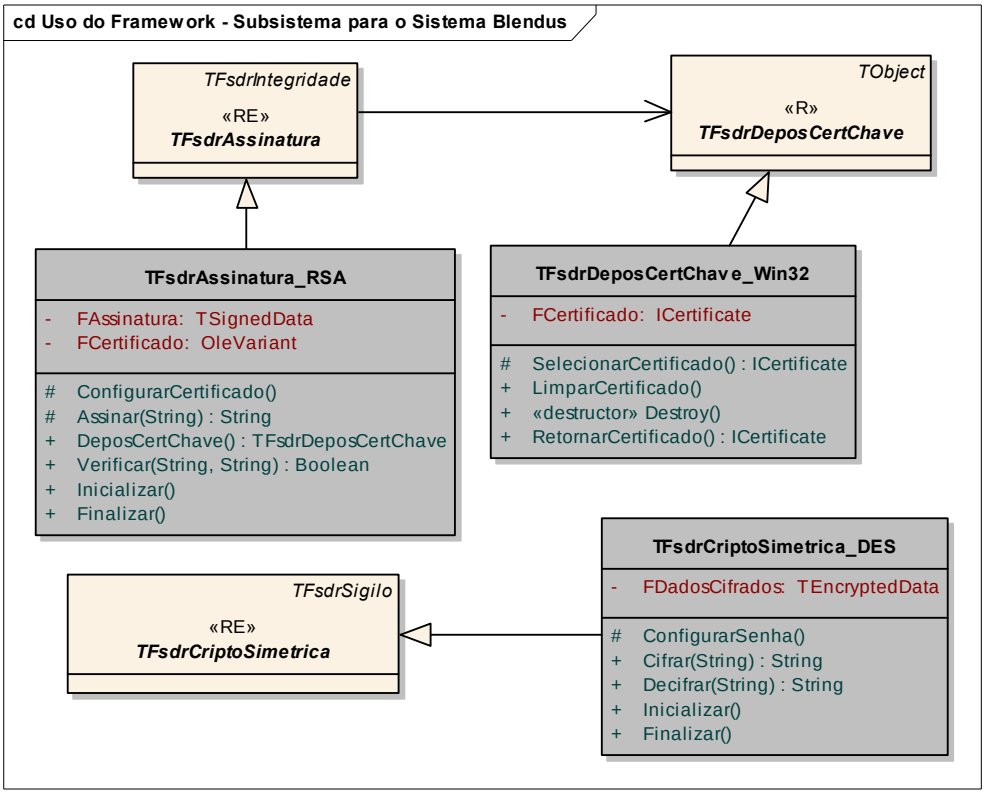

Figura 5.29 – Diagrama de classes contendo as classes do framework Frasedare e subclasses concretas estendidas para o subsistema do sistema Blendus.

Os métodos que implementam as operações nas subclasses utilizam APIs de criptografia (CAPI) fornecidas pela biblioteca de classe CAPICOM. Segue abaixo a descrição da responsabilidade atribuída a cada subclasse estendida:

- ***TFsdrDeposCertChave_Win32*** – É uma subclasse concreta de *TFsdrDeposCertChave* do framework Frasedare. Implementa as APIs que

fazem acesso ao depósito de certificados e chaves do sistema operacional Windows.

- ***TFsdrCriptoSimetrica_DES*** – É uma subclasse concreta de *TFsdrCriptoSimetrica* do framework Frasedare. Implementa as APIs de criptografia que fornecem o mecanismo de criptografia de chave simétrica DES.
- ***TFsdrAssinatura_RSA –*** É uma subclasse concreta de *TFsdrAssinatura* do framework Frasedare. Implementa as APIs de criptografia que fornecem o mecanismo de assinatura digital RSA.

As classes *TFsdrDeposCertChave_Win32* e *TFsdrAssinatura_RSA* possuem a mesma implementação que foi utilizada no subsistema da aplicação teste.

Além da implementação das subclasses, também foram incorporadas ao subsistema as classes concretas *TFsdrSegurancaDados* e *TFsdrMemoField* do framework Frasedare.

O apêndice V apresenta a implementação das subclasses concretas do framework Frasedare definidas para o subsistema utilizado pelo sistema Blendus.

5.2.3.2. Utilizando o Subsistema no Sistema Blendus

Conforme apresentado na seção 5.2.1.2., após a construção do subsistema, suas classes devem ser instaladas no ambiente Delphi para que fiquem disponíveis para uso de uma aplicação ou sistema.

O sistema Blendus fez uso do subsistema na funcionalidade de autorização de procedimentos médicos para garantir a segurança de dados em repouso no parecer da perícia que é registrado por esta funcionalidade.

A figura 5.30 apresenta a tela da funcionalidade de autorização de procedimentos médicos do sistema Blendus.

Conforme pode ser observado na tela, os passos para se utilizar esta funcionalidade normalmente são os seguintes. Primeiramente se informam os dados do

beneficiário e dados da autorização, e depois se deve informar os dados do procedimento médico.

Dependendo dos critérios de valor e complexidade destes procedimentos médicos, esta autorização deve passar por uma perícia médica, onde é lavrado um parecer da perícia. Este parecer da perícia é informado no campo texto da tela de autorização de procedimentos médicos, conforme apresentado na figura 5.30.

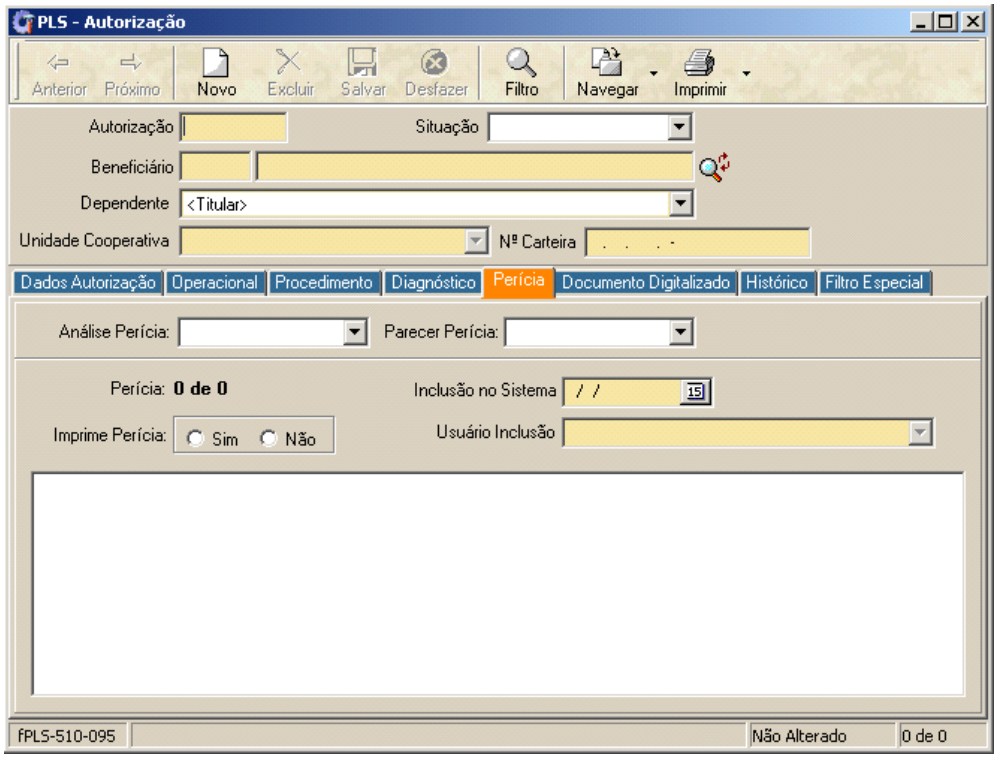

Figura 5.30 – Tela de autorização de procedimento médicos do sistema Blendus.

Para este parecer da perícia médica, se utilizaram as categorias de sigilo e integridade. A categoria de sigilo utilizou o mecanismo de criptografia de chave simétrica e a categoria de integridade utilizou a assinatura digital fornecidos pelo subsistema.

Além da tela de autorização de procedimentos médicos, o subsistema de segurança de dados em repouso também foi utilizado para gerar relatórios e e-mails das autorizações de procedimentos médicos.

Com o sistema Blendus utilizando o subsistema construído, estendendo o framework Frasedare, pôde-se verificar a aplicabilidade do framework Frasedare em um sistema corporativo já consolidado.

5.3. Fase de Evolução e Manutenção do Framework Frasedare

No decorrer do desenvolvimento do framework Frasedare ocorreram muitas mudanças, geralmente devido à busca da flexibilidade necessária para as interfaces das classes, objetivando proporcionar um maior reuso. O framework Frasedare evoluiu conforme ocorriam as mudanças, e as diferenças entre a parte adaptável (*hot spots*) e a parte estável (*frozen spots*) ficavam cada vez mais claras.

Um dos principais motivadores para a mudança foi o próprio reuso do framework Frasedare. Nesta ocasião foi que se observou a necessidade de alguns ajustes, e desta forma, a evolução do framework Frasedare se deu incorporando as fases de desenvolvimento e uso.

5.4. Considerações

Com objetivo de tornar o framework Frasedare flexível e também atender de forma adequada ao seu propósito de segurança, uma importante decisão foi agrupar os mecanismos de criptografia nas categorias de sigilo e integridade, as quais definiram claramente as responsabilidades das classes no projeto e implementação, e facilitaram o entendimento no uso do framework Frasedare.

Este uso, por sua vez, foi feito estendendo-se as classes do framework Frasedare. Estas classes foram implementadas utilizando-se as APIs de criptografia (CAPIs) da biblioteca de classes CAPICOM. Com isto, ocorreu muitas vezes o re-projeto do framework Frasedare, sempre buscando adequar as contribuições fornecidas por esta biblioteca de classes. Portanto, outras contribuições para o projeto do framework Frasedare poderão surgir no futuro, quando forem utilizadas outras bibliotecas de classes.

6. Conclusão

Este trabalho teve seu objetivo geral alcançado com a construção do framework Frasedare: um framework orientado a objetos que fornece funcionalidades de segurança para dados em repouso em aplicações novas e existentes. É classificado como um framework de subsistema quanto ao seu escopo e como caixa-cinza, quanto à técnica de estendê-lo.

Assim como o objetivo geral, os objetivos específicos também foram atingidos:

- Foram identificadas as necessidades comuns em aplicações que precisavam de segurança de dados em repouso, definindo-se e delimitando-se o domínio do framework construído;
- Foi definida uma arquitetura, a qual forneceu a base para o projeto do framework Frasedare;
- O projeto do framework Frasedare foi definido conforme os conceitos estudados e o domínio definido;
- O projeto do framework Frasedare foi implementado na linguagem de programação *Object Pascal* do ambiente Delphi;
- O uso do framework Frasedare proporcionou a evolução do framework através dos ajustes necessários frente ao domínio e reuso.

O desenvolvimento contemplou as atividades de análise do domínio, projeto arquitetural, projeto do framework e teste. A análise de domínio foi uma importante atividade na qual pôde se determinar claramente o domínio do framework Frasedare referente à segurança de dados em repouso e sua delimitação. A segurança de dados em repouso compreendeu o fornecimento dos serviços de segurança de sigilo, integridade, autenticação e não-repúdio. A implementação dos serviços de segurança foi alcançada através dos mecanismos de criptografia. Na atividade de projeto arquitetural, baseada na análise de domínio, foi definida a arquitetura do framework Frasedare. A atividade de projeto do framework Frasedare foi a que consumiu maior esforço do trabalho, por se buscar uma solução flexível e reusável. A atividade de implementação deu-se através da

codificação do projeto do framework Frasedare e a atividade de teste através do teste deste código.

O uso do framework Frasedare ocorreu estendendo-o para a construção de subsistemas que foram utilizados em uma aplicação teste e em um sistema corporativo. Com o uso, pôde-se constatar a aplicabilidade do framework Frasedare. O subsistema utilizado na aplicação teste explorou todos os recursos proporcionados pelo framework Frasedare, verificando-se a aplicabilidade, flexibilidade e reuso. Já o subsistema utilizado em um sistema corporativo verificou a aplicabilidade do framework Frasedare quanto à facilidade da inclusão dos recursos de segurança de dados em repouso nas aplicações ou sistemas, e seu uso em um sistema corporativo consolidado comprovou esta facilidade e a utilização do framework Frasedare como uma solução reusável.

A evolução e manutenção ocorreram paralelamente com a fase de desenvolvimento e uso do framework Frasedare, pois com o uso do framework Frasedare foram detectadas melhorias quanto às funcionalidades e a flexibilidade, o que ocasionava novamente a fase de desenvolvimento, assim resultando na evolução. Desta forma, a evolução do framework Frasedare se deu de forma iterativa, uma vez que nas várias iterações de desenvolvimento e de uso é que ocorreu a evolução, tornando-o mais estável.

Pode se concluir que devido a busca pelo alto grau de flexibilidade e reuso de frameworks orientados a objetos, o seu desenvolvimento é complexo. Contudo, a complexidade não se restringe somente ao desenvolvimento. O uso de frameworks orientados a objetos também se caracterizam pela complexidade, pois para se utilizar o framework deve-se conhecer muito bem a parte adaptável (*hot spots*) e a parte estável (*frozen spots*).

Entretanto, utilizar frameworks orientados a objetos está relacionado à possibilidade de ganhos em produtividade e qualidade do desenvolvimento de aplicações ou sistemas, ao proporcionar a facilidade na inclusão de funcionalidades já testadas.

Quanto à segurança de dados em repouso, algumas questões inerentes foram observadas, e devem ser levadas em consideração. Os dados, quando cifrados, são consideravelmente maiores que os dados originais; desta forma, quando se definir

tamanho de campos, colunas ou atributos em um dispositivo de armazenamento, deve-se estar atento para esta questão. Também, pelo fato dos dados ficarem armazenados por muito tempo na maioria das vezes, se exige a necessidade de gerenciamento e armazenamento de chaves e certificados em longo prazo.

Os seguintes resultados deste trabalho puderam ser identificados:

- Um estudo sobre questões de segurança aplicada para os dados em repouso, considerando os serviços de segurança de sigilo, integridade, autenticação e não-repúdio;
- A implementação de um framework orientado a objetos, que permite a adição de segurança de dados em repouso nas aplicações ou sistemas;
- Um projeto de um framework orientado a objetos, que pode ser implementado em outras linguagens de desenvolvimento.
- Um trabalho científico apresentando um estudo sobre frameworks aplicados a um domínio específico.

6.1. Limitações

São apresentadas algumas limitações para o framework Frasedare, como segue:

- Como a análise de domínio se deu sobre aplicações com arquitetura cliente/servidor, a estrutura de classes do framework Frasedare foi desenvolvida para esta arquitetura, tornando assim o framework dependente dela.
- Quando os dados com segurança forem armazenados, estes devem ser armazenados somente nas colunas, campos ou atributos de tipo binário ou texto nos dispositivos de armazenamento, pois colunas, campos ou atributos de outros tipos, como tipos numérico e datas, possuem restrições inerentes ao seu tipo. Por exemplo, um tipo numérico somente recebe dados numéricos; desta forma, este tipo não suporta dados com segurança que possuem um conjunto de caracteres diversos. A segurança para estes tipos de dados diferentes de binário ou texto, não foi considerada no framework Frasedare.

6.2. Trabalhos Futuros

Algumas questões, apresentadas abaixo, podem ser consideradas como trabalhos futuros para o framework Frasedare:

- Definir classes concretas para os mecanismos de criptografia, preferencialmente com algoritmos próprios para evitar acoplamento com bibliotecas de classes externas de criptografia (CAPIs). Isto tornaria o framework Frasedare mais próximo a componentização, facilitando ainda mais a inclusão da segurança de dados nas aplicações e sistemas.
- Desenvolver uma forma de criar uma infra-estrutura própria para gerenciar chaves e certificados digitais, assim criando uma maior independência das bibliotecas de classes externas de criptografia (CAPIs) utilizadas.
- Construir um projeto do framework Frasedare e implementá-lo para arquiteturas Web e Móvel.
- Desenvolver uma documentação adicional para o framework Frasedare, como por exemplo, a utilização de *Cookbook* e contratos de operações com pré e pós-condições. Esta documentação poderá fornecer informações com outras visões além da documentação já existente, auxiliando quem fizer uso do framework Frasedare.

7. Referências Bibliográficas

(ALEXANDER, 1977) ALEXANDER, Christopher; et. al. *apud* GAMMA (1995). **A Pattern Language.** New York: Oxford University Press, 1977.

(ALUR, 2001) ALUR, Deepak; CRUPI, John; MALKS, Dan. Tradução de: MORAIS, Altair Dias Caldas de; DIAS, Cláudio Belleza; MORAES, Guilherme Dias Caldas de. **Core J2EE Patterns: As melhores práticas e estratégias de design.** Rio de Janeiro: Campus, 2002.

(BOOCH, 1999) BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar, 1999. Tradução de Fábio Freitas. **UML: Guia do Usuário.** Rio de Janeiro: Campus, 2000.

(BUCHMANN, 2001) BUCHMANN, Johannes A., 2001. Tradução de Bazán Tecnologia e Lingüística. **Introdução à Criptografia.** São Paulo: Berkeley, 2002.

(BURNETT & PAINE, 2002) BURNETT, Steve; PAINE, Stephen. Tradução de Docware Traduções Técnicas e Edson Furmankiewicz. **Criptografia e Segurança: O Guia Oficial RSA.** Rio de Janeiro: Campus, 2002.

(BUSCHMANN, 1996) BUSCHMANN, Frank, et. al. **Pattern-Oriented Software Architecture: A System of Patterns.** Chichester: John Wiley & Sons Ltd., 1996.

(CARVALHO, 2001) CARVALHO, Daniel Barlparda de, **Segurança de Dados com Criptografia: Métodos e Algoritmos.** Rio de Janeiro: Book Express, 2001, 2ª ed.

(CERTICOM, 2004) **Security Builder Crypto: Cross-plataform cryptographic toolkit.** Certicom Securing Information, 2004. Disponível em: < http://www.certicom.com/download/aid-72/SBC_datasheet.pdf> Acessado em: 07/08/2004.

(COMPUTERWORLD, 2004) **IDC aponta crescimento dos mercados de storage.** Computerworld. São Paulo, Agosto 2004. Disponível em: <http://computerworld.uol.com.br/AdPortalV3/adCmsDocumentoShow.aspx?Documento=28963> Acessado em: 22/08/2004.

(COMPUTERWORLD, 2005a) **Laptop with 98,000 names stolen at UC-Berkeley.** Computerworld. March 2005. Disponível em: <http://www.computerworld.com/hardwaretopics/hardware/story/0,10801,100712,00.html> Acessado em: 29/03/2005.

(COMPUTERWORLD, 2005b) **Update: Stolen computers contain data on 185,000 patients.** Computerworld. April 2005. Disponível em: <http://www.computerworld.com/databasetopics/data/story/0,10801,100961,00.html> Acessado em: 08/04/2005.

(CONSTITUIÇÃO, 1988) **Constrituição da República Federativa do Brasil.** Outubro 1988.

(DB4O, 2005a) **db4o – Banco de objetos de código aberto.** Resources, db4object, 2005.

<http://www.db4o.com/portugues/db4o%20Product%20Information%20V5.0(Portuguese).pdf> Acessado em: 09/12/2005.

(DB4O, 2005b) **db4o – Tutorial.** Resources, db4object, 2005. <http://www.db4o.com/about/productinformation/resources/db4o-5.0-tutorial-java.pdf> Acessado em: 09/12/2005.

(D'SOUZA & WILLS, 1998) D'SOUZA, Desmond Francis; WILLS, Alan Cameron. **Objects, Components, and Frameworks with UML: The Catalysis Approach.** Upper Saddle River: Addison Wesley, 1998.

(EU, 2002) **Data Protection: Directive on privacy and electronic communications.** Freedom, Security and Justice. July 2002. Disponível em: <http://europa.eu.int/comm/justice_home/fsj/privacy/law/index_en.htm> Acessado em: 09/12/2005.

(EXTERKOETTER, 2003) EXTERKOETTER, Flávio. **Blendwork: Framework Orientado a Objetos para Desenvolvimento.** 2003. 107 f. Dissertação (Dissertação em Ciência da Computação) - Programa de Pós-graduação em Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis.

(FAYAD & SCHMIDT, 1997) FAYAD, Mohamed E.; SCHMIDT, Douglas C. **Object-Oriented Application Frameworks.** Communications of the ACM, Special Issue on Object-Oriented Application Frameworks, v. 40, n. 10, October 1997.

(FAYAD, 1999a) FAYAD, Mohamed E.; SCHMIDT, Douglas C.; JOHNSON, Ralph E. **Building Application Frameworks: Object-Oriented Foundations of Framework Design.** New York: John Wiley & Sons, Inc., 1999.

(FAYAD, 1999b) FAYAD, Mohamed E.; JOHNSON, Ralph E. **Domain-Specific Application Frameworks: Frameworks Experience by Industry.** New York: John Wiley & Sons, Inc., 1999.

(FAYAD, 1999c) FAYAD, Mohamed E.; SCHMIDT, Douglas C.; JOHNSON, Ralph E. **Implementing Application Frameworks: Object-Oriented Frameworks at Work.** New York: John Wiley & Sons, Inc., 1999.

(FOWLER, 1997) FOWLER, Martin. **Analysis Patterns: Reusable Object Models.** Addison Wesley, 1997.

(GAMMA, 1995) GAMMA, Erich; et al. Tradução de SALGADO, Luiz A. Meireles. **Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos.** Porto Alegre: Bookman, 2000.

(GSS, 2000a) **Generic Security Service Application Program Interface.** RFC 2743, Version 2, Update 1, January 2000. Disponível em: <http://www.ietf.org/rfc/rfc2743.txt> Acessado em: 07/08/2004.

(GSS, 2000b) **Generic Security Service API: Java Bindings.** RFC 2853, Version 2, Disponível em: <http://www.ietf.org/rfc/rfc2853.txt> Acessado em: 11/02/2005.

(GURP & BOSCH, 2001) GURP, Jilles van; BOSCH, Jan. **Design, Implementation and Evolution of Object Oriented Frameworks: Concepts and Guidelines.** Software – Practice & Experience, v. 31. New York: John Wiley & Sons, Inc., 2001, p. 277-300.

(JEON, 2002) JEON, Taewoong; SEUNG, Hyon Woo; LEE, Sungyoung. **Embedding Built-in Tests in Hot Spots of an Object-Oriented Framework.** ACM SIGPLAN Notices, v.37, August 2002, p. 25-34.

(JOHNSON & FOOTE, 1988) JOHNSON, Ralph E.; FOOTE, Brian. **Designing reusable classes.** Journal of Object-Oriented Programming, v.1, june and july 1988, p. 22-35.

(JOHNSON, 1993) JOHNSON, Ralph E. **How to Design Frameworks.** Tutorial Notes, OOPSLA '93, Washington, 1993. Disponível em: <www.cse.msu.edu/~cse870/Materials/Frameworks/how-todesign-fw-tutorial.ps>. Acessado em: 14/07/2004.

(JOHNSON, 1997) JOHNSON, Ralph E., **How Frameworks Compare to Other Object-Oriented Reuse Techniques – Frameworks = (Components + Patterns).** Communications of the ACM, v. 40, n. 10, October 1997, p. 39-42.

(JOHNSON & RUSSO, 1991) JOHNSON, Ralph E.; RUSSO, Vincent F. *apud* FAYAD (1999a). **Reusing Object-Oriented Design.** Technical Report UIUCDCS 91-1696, University of Illinois, 1991.

(KIRK, 2005) KIRK, Douglas; ROPER, Marc; WOOD, Murray. **Identifying and Addressing Problems in Framework Reuse.** 13th International Workshop on Program Comprehension, IEEE, May 2005.

(MAIWALD, 2001) MAIWALD, Eric. **Network Security: A Beginner´s Guide.** New York: McGraw-Hill/Osborne, 2001.

(MATTSSON, 1996) MATTSON, Michael. **Object-Oriented Frameworks: A survey of methodological issues.** 1996, 130p. Thesis (Licentiate) - Department of Computer Science, Lund University, Lund.

(MATTSSON, 2000) MATTSSON, Michael, **Evolution and Composition of Object-Oriented Frameworks.** 2000, 219p. Thesis (Engineering Doctor) - Department of Software Engineering and Computer Science. University of Karlskrona/Ronneby.

(MCILROY, 1968) MCILROY, M. Douglas, **Mass Produced Software Components.** Working Conference on Software Engineering: NATO Science Committee, Garmisch, Germany, October 1968.

(MENEZES, 1997) MENEZES, Alfred; OORSCHOT, Paul van; VANSTONE, Scott. **Handbook of Applied Cryptography.** CRC Press, 1997. Disponível em: <http://www.cacr.math.uwaterloo.ca/hac/> Acessado em 17/07/2004.

(MICROSOFT, 2003) **Microsoft Plataform SDK.** Microsoft, February 2003. Disponível em: <http://www.microsoft.com/msdownload/platformsdk/sdkupdate/> Acessado em 23/07/2004.

(OLIVEIRA, 2004) OLIVEIRA, Toacy C.; et al. **Software Process Representation and Analysis for Framework Instantiation.** IEEE Transactions on Software Engineering, v. 30, n. 3, March, 2004.

(OPENGROUP, 1996) **Generic Cryptographic Service API (GCS-API) Base.** The Open Group, Berkshire: X/Open Company Ltd., June, 1996. Disponível em: <http://www.opengroup.org/onlinepubs/008355799/toc.pdf> Acessado em: 27/07/2004.

(ORACLE, 2003) **Privacy Protections in Oracle Database 10g.** Oracle Database, September, 2003. Disponível em: <http://www.oracle.com/solutions/security/docs/privacy10g.pdf> Acessado em: 09/12/2005.

(ORACLE, 2005) **Oracle Advanced Security.** Oracle Database, May, 2005. Disponível em: < http://www.oracle.com/technology/deploy/security/db_security/pdf/ds_security_db_advancedsecurity_10r2_0509.pdf > Acessado em: 09/12/2005.

(PREE, 1995) PREE, Wolfgang. **Design Patterns for Object-Oriented Software Development.** Reading, MA: Addison-Wesley, 1995.

(PRESSMAN, 2002) PRESSMAN, Roger S., **Engenharia de Software.** McGraw-Hill, 2002, 5ª ed.

(RIVEST, 1978) RIVEST, Ronald, SHAMIR, Adi, ADLEMAN, Leonard. **A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.** Programming Techniques, ACM, 1978.

(RSA, 2002) **FAQ Frequently Asked Questions about Today´s Cryptography.** CD-ROM de Referência, RSA Laboratories, Campus, 2002.

(SARBANES, 2002) **Sarbanes-Oxley Act of 2002.** Public Law, 107th Congress, USA, July 2002. Disponível em: <http://www.sec.gov/about/laws/soa2002.pdf> Acessado em: 09/12/2005.

(SARDINHA, 2003) SARDINHA, José A. R. P.; et. al. **An Object-Oriented Framework for Building Software Agents.** Journal of Object Technology, vol. 2, n. 1, January-February 2003, p. 85-97.

(SCHÄFER, 1994) SCHÄFER, Wilhelm; PRIETO-DIAZ, Ruben; MATSUMOTO, Masao. **Software Reusability.** Ellis-Horwood Ltd., 1994.

(SCHNEIER, 1996) SCHNEIER, Bruce. **Applied Cryptography: Protocols, Algorthms and Source Code in C.** John Wiley & Sons, Inc., 1996, 2ª ed.

(SHAW, 1996) SHAW, Mary; GARLAN, David. **Software Architecture: Perspectives on an Emerging Discipline.** Prentice Hall, Inc., 1996.

(SILVA, 2000) SILVA, Ricardo Pereira e. **Suporte ao Desenvolvimento e Uso de Frameworks e Componentes.** 2000. 262 f. Tese (Doutorado em Ciência da Computação) - Programa de Pós-graduação em Ciência da Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre.

(SILVA, 2004) SILVA, Ricardo Pereira e. FREIBERGER, Evandro Cezar. **Helping Object-Oriented Framework Use and Evaluation by means of Historical Use Information.** 19th International Conference on Automated Software Engineering, IEEE, 2004, pp. 278-281.

(SQL, 2005) **Improving Data Security by Using SQL Server 2005.** Technical White Paper, October 2005. Disponível em: <http://www.microsoft.com/technet/itsolutions/msit/security/sqldatsec.mspx> Acessado em: 09/12/2005.

(STALLINGS, 2003) STALLINGS, William. **Cryptography and Network Security: Principles and Practice.** Upper Saddle River: Prentice Hall, 2003, 3ª ed.

(STINSON, 2002) STINSON, Douglas R. **Cryptography: Theory and Practice.** Boca Raton: Chapman & Hall/CRC, 2002, 2ª ed.

(SUN, 2002) **Java Cryptography Architecture: API Specification & Reference.** Sun Microsystems, Inc., August 2002. Disponível em: < http://java.sun.com/j2se/1.4.2/docs/guide/security/CryptoSpec.html> Acessado em: 27/07/2004.

(SYBASE, 2005a) **Protecting Personal Data in Sybase Adaptive Server Enterprise.** Sybase, July 2005. Disponível em: <http://www.sybase.com/content/1036206/L02694_ASE_EncryptionWP.pdf> Acessado em: 09/12/2005.

(SYBASE, 2005b) **New Features Adaptive Server Enterprise.** Sybase, October 2005. Disponível em: <http://www.sybase.com/detail?id=1038160> Acessado em: 09/12/2005.

(TANENBAUM, 2003) TANENBAUM, Andrew S. **Computer Networks.** Upper Saddle River: Prentice Hall, 2003, 4ª ed.

(TALIGENT, 1993) **Leveraging Object-Oriented Frameworks.** White-paper: Taligent Inc, 1993.

(TALIGENT, 1995) **Building Object-Oriented Frameworks.** White-paper: Taligent Inc, 1995.

(UML, 2005) **Unified Modeling Language: Superstructure.** Specification: OMG, August 2005.

Apêndice I

Atributos e operações das classes de mecanismos de criptografia.

A seguir serão descritas as responsabilidades dos atributos e o comportamento dos métodos que são implementados nas operações concretas ou que deverão ser implementados nas operações abstratas das classes *TFsdrMecanismoCriptografia*, *TFsdrIntegridade*, *TFsdrSigilo*, *TFsdrAssinatura*, *TFsdrResumo*, *TFsdrCriptoSimetrica*, *TFsdrCriptoAssimetrica* e *TFsdrDeposCertChave*.

- Operações da classe ***TFsdrMecanismoCriptografia***:
  - *Create()* – É a operação implementada pelo método construtor da classe. Este método é sobrescrito para implementar a invocação da operação *Inicializar()*. Assim, este método é definido como um padrão de projeto Método *Template*.
  - *Destroy()* – É a operação implementada pelo método destruidor da classe. Este método é sobrescrito para implementar a invocação da operação *Finalizar()*. Assim, este método também é definido como um padrão de projeto Método *Template.*
  - *Inicializar()* – É uma operação protegida e abstrata que deverá ser implementada por um método na subclasse concreta. Na subclasse concreta, o método deverá implementar a criação de objetos que precisam ser utilizados desde a criação da instância ou mesmo definir configurações iniciais para ela. A criação dos objetos de um mecanismo de criptografia pode ser um exemplo de utilização desta operação.
  - *Finalizar()* – É uma operação protegida e abstrata que deverá ser implementada por um método na subclasse concreta. O método deverá implementar a destruição de objetos criados com o método *Inicializar().*
- Atributo da classe ***TFsdrIntegridade***

- *FOnIntegridadeViolada: TNotifyEvent* – É um atributo privado do tipo *TNotifyEvent*. Este atributo foi definido para ser utilizado pelo evento *OnIntegridadeViolada*. Este evento será chamado quando a integridade dos dados em repouso for violada.

- Operações da classe ***TFsdrIntegridade*:**
  - *Aplicar(String): String* – Esta operação é pública e abstrata. A subclasse concreta deverá implementar um método de mecanismo de criptografia para a operação, que forneça o serviço de segurança de integridade. Este método recebe como argumento um texto claro e retorna um texto integridade, que poderá ser um resumo de dados ou assinatura digital, dependendo do mecanismo de criptografia implementado na subclasse.
  - *Verificar(String, String): Boolean* – Esta operação é pública e abstrata. Deverá ser implementada por um método nas subclasses concretas, que irá verificar se um texto claro está íntegro. O primeiro argumento passado é um texto claro e o segundo é um texto integridade. Este método deverá retornar um valor verdadeiro caso o texto integridade do primeiro argumento corresponder ao segundo argumento, caso contrário, retornará um valor falso.
  - *IntegridadeViolada()* – Esta operação é pública e é invocada quando a integridade dos dados for violada. Ela é implementada por um método que invoca o evento *OnIntegridadeViolada* caso este esteja referenciando algum método de evento, caso contrário, lança uma exceção.

- Atributo da classe ***TFsdrSigilo:***
  - *FOnDadosCifradosCorrompidos: TNotifyEvent* – É um atributo privado do tipo *TNotifyEvent*. Este atributo foi definido para ser utilizado pelo evento *OnDadosCifradosCorrompidos*. Este evento será chamado quando os dados cifrados em repouso estiverem corrompidos.

- Operações da classe ***TFsdrSigilo:***
  - *Cifrar(String): String* – É uma operação pública e abstrata. Esta operação deverá ser implementada nas subclasses concretas com um método de

mecanismo de criptografia que forneça o serviço de segurança de sigilo, como um mecanismo de criptografia de chave simétrica ou criptografia de chave assimétrica. Recebe como argumento um texto claro e deverá retornar o texto cifrado.

- *Decifrar(String): String* – É uma operação pública e abstrata. Nas subclasses concretas, deverá ser implementada por um método de mecanismo de criptografia que forneça o serviço de segurança de sigilo. Recebe como argumento o texto cifrado e retorna o texto claro (dados originais).
- *DadosCifradosCorrompidos()* – Esta operação é pública e é invocada quando os dados cifrados estiverem corrompidos no momento de decifrá-los. Ela é implementada por um método que invoca o evento *OnDadosCifradosCorrompidos* caso este esteja referenciando algum método, caso contrário, lança uma exceção.

- Operações da classe ***TFsdrAssinatura:***
  - *Create(TComponent)* – Esta operação é implementada pelo método construtor da classe. O método foi sobrescrito para invocar a operação *DeposCertChave(): TFsdrDeposCertChave* de uma instância da classe *TFsdrDeposCertChave* que retorna um objeto referenciando um depósito de chaves e certificados.
  - *Destroy()* – Esta operação é implementada pelo método destruidor da classe. O método foi sobrescrito para destruir a instância da classe *TFsdrDeposCertChave* criada no método da operação *Create(TComponent).*
  - *ConfigurarCertificado()* – É uma operação protegida e abstrata. Deverá ser implementada por um método nas subclasses concretas. Este método deverá implementar a delegação do certificado para um método de assinatura digital.
  - *Assinar(String): String* – Esta operação é protegida. Seu método deve ser sobrescrito nas subclasses concretas com um algoritmo de assinatura

digital, ou utilizar uma classe que implemente assinatura digital. Recebe como argumento o texto claro e deve retornar a assinatura conforme a implementação.

- *Aplicar(String): String* – É uma operação pública definida na superclasse. É implementada por um método que delega algumas responsabilidades invocando as operações *ConfigurarCertificado()* e *Assinar()*. É um padrão de projeto método *Template*.
- *DeposCertChave(): TFsdrDeposCertChave* – Esta operação é pública e abstrata. Na subclasse concreta deverá ser implementada por um método que retorne um objeto do tipo *TFsdrDeposCertChave* que representa todos os certificados e chaves disponíveis e acessíveis no depósito de certificados e chaves. É um padrão de projeto *Factory Method.*
- *RetornarDeposCertChave(): TFsdrDeposCertChave* – Esta operação é pública. É implementada por um método que retorna um objeto representante do depósito de certificados e chaves que está associado à instância da classe.

- Operações da classe ***TFsdrResumo:***
  - *Resumir(String): String* – Esta operação é pública e abstrata. Deverá ser implementada por um método nas subclasses com um mecanismo de criptografia para resumo de dados. Recebe como argumento um texto claro e retorna um resumo de dados.
  - *Aplicar(String): String* – É uma operação pública definida na superclasse. Nesta classe, o método é sobrescrito para delegar a responsabilidade, invocando a operação *Resumir(String): String*.
- Atributo da classe ***TFsdrCriptoSimetrica:***
  - *FSenha: String* – É um atributo privado que irá manter a senha para o mecanismo de criptografia de chave simétrica.
- Operações da classe ***TFsdrCriptoSimetrica:***

- *SetSenha(String)* – É uma operação privada. É implementada por um método que configura a senha para o mecanismo de criptografia de chave simétrica. Recebe como argumento uma senha configurando o atributo *FSenha* através da propriedade[1] *Senha* da classe. É um padrão de projeto método *Template*.
- *ConfigurarSenha()* – É uma operação protegida e abstrata. Deve ser implementada por um método na subclasse concreta. Este método deverá configurar a senha na classe em que implementar o mecanismo de criptografia de chave simétrica.
- *Cifrar(String): String* – É uma operação pública definida na superclasse. Nesta classe ela é implementada por um método que irá verificar se a senha foi informada.

- Operações da classe ***TFsdrCriptoAssimetrica:***
  - *Create(TComponent)* – Esta operação é implementada pelo construtor da classe. O seu método foi sobrescrito para invocar a operação *DeposCertChave(): TFsdrDeposCertChave* que retorna uma instância da classe *TFsdrDeposCertChave,* que representa o depósito de certificados e chaves.
  - *Destroy()* – Esta operação é implementada pelo método destruidor da classe. Foi sobrescrito para destruir a instância da classe *TFsdrDeposCertChave* criada no construtor.
  - *ConfigurarCertificado()* – É uma operação protegida e abstrata que deverá ser implementada por um método nas subclasses concretas. Este método deverá configurar o certificado para a classe ou algoritmo que implemente um mecanismo de criptografia de chave assimétrica.
  - *Cifrar(String): String* – É uma operação pública definida pela superclasse. O seu método foi sobrescrito para invocar a operação

[1] A linguagem de programação *Object Pascal* do ambiente Delphi utiliza propriedade para definir atributo de classe implementando comportamento ao ler e configurar seus dados.

ConfigurarCertificado() da própria classe. É um padrão de projeto método *Template*.

- *DeposCertChave(): TFsdrDeposCertChave* – Esta operação é pública e abstrata. Deverá ser implementada nas subclasses por um método que retorne uma instância da classe *TFsdrDeposCertChave* que representa todos os certificados e chaves disponíveis e acessíveis no depósito de certificados e chaves. É um padrão de projeto *Factory Method.*
- *RetornarDeposCertChave(): TFsdrDeposCertChave* – Esta operação é pública. É implementada por um método que retorna um objeto que representa o depósito de certificados e chaves que está associada à instância da classe.

- Operações da classe ***TFsdrDeposCertChave:***
  - *LimparCertificado()* – Esta operação é pública e abstrata. Deve ser implementada por um método nas subclasses. Este método deverá possuir o comportamento de limpar o certificado que foi configurado na classe de segurança em uma interação anterior, quando foi aplicada ou retirada a segurança de um conjunto de dados.
  - *NewInstance(): TObject* – Esta é uma operação pública e estática, invocada pelo método construtor da classe. Seu método foi sobrescrito para definir o comportamento do padrão de projeto *Singleton.*
  - *FreeInstance()* – Esta é uma operação pública invocada pelo método destruidor da classe. Seu método foi sobrescrito para definir o comportamento do padrão de projeto *Singleton*.

Apêndice II

Atributos e operações das classes de segurança de dados e persistência.

A seguir serão descritas as responsabilidades dos atributos e o comportamento dos métodos que são implementados nas operações concretas ou que deverão ser implementados nas operações abstratas das classes *TFsdrTextoSeguro*, *TFsdrSegurancaDados*, *TFsdrBlobField, TFsdrMemoField* e *TFsdrStringField*.

- Atributos da estrutura tipo registro ***TFsdrTextoSeguro:***
  - *Sigilo* – Recebe o texto sigilo.
  - *Integridade* – Recebe o texto integridade.
- Operações da classe ***TFsdrSegurancaDados:***
  - *SetIntegridade(TFsdrIntegridade)* – É uma operação privada implementada por um método que associa a instância desta classe a uma instância da classe *TFsdrIntegridade*. Recebe como argumento uma instância da classe *TFsdrIntegridade*. Esta operação é invocada pela propriedade *Integridade* definida na classe *TFsdrSegurancaDados*.
  - *SetSigilo(TFsdrSigilo)* – É uma operação privada implementada por um método que associa a instância desta classe a uma instância da classe *TFsdrSigilo*. Recebe como argumento uma instância de *TFsdrSigilo*. Esta operação é invocada pela propriedade *Sigilo* definida na classe *TFsdrSegurancaDados*.
  - *AplicarIntegridade(String): String* – É uma operação privada implementada por um método que recebe como argumento o texto claro e retorna o texto integridade, ou seja, retorna uma assinatura digital ou o resumo de dados, conforme o mecanismo de criptografia implementado na classe da instância associada a propriedade *Integridade*.

- *AplicarSigilo(String): String* – É uma operação privada implementada por um método que recebe como argumento o texto claro e retorna o texto cifrado, conforme o mecanismo de criptografia implementado na classe da instância associada a propriedade *Sigilo*.
- *RetirarSigilo(String): String* – É uma operação privada que é implementada por um método para decifrar um texto sigiloso. Recebe como argumento um texto cifrado e decifra este texto conforme o mecanismo de criptografia implementado na classe da instância associada à propriedade *Sigilo*.
- *VerificarIntegridade(String, String): Boolean* – Esta operação privada é implementada por um método que verifica a integridade do texto passado no primeiro parâmetro. O primeiro parâmetro é o texto claro e o segundo é o texto integridade. Retorna verdadeiro se o texto integridade processado sobre o texto claro corresponder ao texto integridade do segundo parâmetro; caso não corresponder, retorna falso.
- *AplicarSeguranca(String): TFsdrTextoSeguro* – Esta é uma operação pública implementada por um método que aplica a segurança nos dados. Recebe como argumento o texto claro e retorna o texto com a segurança aplicada. O texto com segurança é criado dependendo do comportamento implementado nas instâncias dos mecanismos de criptografia associadas às propriedades *Sigilo* e *Integridade*. Seguindo o fluxo de execução do método, primeiro é verificado se a instância de *TFsdrSegurancaDados* está referenciando algum mecanismo de criptografia que necessite de certificado digital. Em caso positivo, é invocada a operação *LimparCertificado()* de uma instância de classe do tipo *TFsdrDeposCertChave* que implementa o acesso ao depósito de certificados digitais e chaves, referenciado pelo mecanismo de criptografia que necessita de certificado digital. Após é invocada a operação *AplicarSigilo(String): String*. Se houver uma instância referenciada pela propriedade *Sigilo*, então será invocada a operação *Cifrar(String): String*. Seguindo a execução, a operação

AplicarIntegridade(String): String é invocada. Se houver uma instância referenciada pela propriedade *Integridade,* será invocada a operação *Aplicar(String): String* desta instância. Por fim, o texto sigilo e o texto integridade gerados são retornados no formato da estrutura de dados de registro *TFsdrTextoSeguro*.

- *RetirarSeguranca(TFsdrTextoSeguro): String* – Esta operação é implementada por um método que recupera o texto claro (dados originais) retirando a segurança do texto seguro que é passado como argumento. O comportamento deste método é inverso ao comportamento do método *AplicarSeguranca(String): String*. Inicialmente é invocada a operação *RetirarSigilo(String): String.* Se houver alguma instância sendo referenciada pela propriedade *Sigilo*, então será invocada desta instância a operação *Decifrar(String): String* que recebe como argumento o texto cifrado e retorna o texto claro. Seguindo o fluxo de execução, a operação *VerificarIntegridade(String, String): String* é invocada. Se houver uma instância sendo referenciada pela propriedade *Integridade*, será invocada desta instância a operação *Verificar(String, String): Boolean* que é implementada por um método para verificar se o texto está íntegro, retornando verdadeiro se o texto estiver íntegro, caso contrário, retornando falso.

- Operações comuns para as classes ***TFsdrStringField***, ***TFsdrMemoField*** e ***TFsdrBlobField:***
  - *GetDataFieldIntegridade(): TField* – É uma operação privada implementada por um método que retorna uma instância de um tipo *TField* do campo de integridade. Esta operação é invocada pela propriedade *DataFieldIntegridade* de cada uma das classes.
  - *GetSegurancaDados(): TFsdrSegurancaDados* – É uma operação privada que retorna uma instância de *TFsdrSegurancaDados* associada a uma instância destas classes. Esta operação é invocada pela propriedade *SegurancaDados* das classes.

- *SetDataFieldIntegridade(TField)* – É uma operação privada implementada por um método que associa uma instância do tipo *TField* as instâncias destas classes. Esta instância associada é utilizada para a identificação do campo de integridade. Esta operação é invocada pela propriedade *DataFieldIntegridade* de cada uma das classes.
- *SetSegurancaDados(TFsdrSegurancaDados)* – É uma operação privada implementada por um método que associa uma instância de *TFsdrSegurancaDados*. Esta operação é invocada pela propriedade *SegurancaDados* das classes.
- *Destroy()* – Esta operação é implementada pelo método destruidor da classe, sendo este método sobrescrito para implementar a anulação das instâncias associadas as instâncias destas classes.

- Operações comuns das classes ***TFsdrStringField*** e ***TFsdrMemoField:***
  - *GetAsString(): String* – É uma operação protegida. O método desta operação é sobrescrito nestas classes para adicionar o comportamento de segurança de dados. Se a propriedade *SegurancaDados* estiver referenciando uma instância, retira a segurança conforme o estado da instância de *TFsdrSegurancaDados*, caso contrário, o comportamento do método é o mesmo da sua superclasse.
  - *SetAsString(String)* – É uma operação protegida. O método desta operação é sobrescrito nestas classes para adicionar o comportamento de segurança de dados. Se a propriedade *SegurancaDados* estiver referenciando uma instância, adiciona a segurança conforme o estado da instância de *TFsdrSegurancaDados*, caso contrário, o comportamento do método é o mesmo da sua superclasse.
- Operações da classe ***TFsdrBlobField:***
  - *AssignTo(TPersistent)* – É uma operação protegida. O método desta operação foi sobrescrito nesta classe para invocar as operações *SaveToStrings(TStrings)* e *SaveToBitmap(TBitmap)* redefinidas nesta

classe. Esta operação é invocada quando os dados são recuperados de um dispositivo de armazenamento.

- *Assign(TPersistent)* – É uma operação pública. O método desta operação foi sobrescrito nesta classe para invocar as operações *LoadFromStrings(TStrings), LoadFromBitmap(TBitmap)* e *LoadFromBlob(TBlobField)* redefinidas nesta classe. Esta operação é invocada quando os dados são salvos em um dispositivo de armazenamento.
- *LoadFromStrings(TStrings)*, *LoadFromBitmap(TBitmap)* e *LoadFromBlob(TBlobField)* – O método destas operações foi redefinido nesta classe para adicionar o comportamento de segurança de dados. Se a propriedade *SegurancaDados* estiver referenciando uma instância, adiciona a segurança conforme o estado da instância de *TFsdrSegurancaDados*, caso contrário, o comportamento do método é o mesmo da sua superclasse.
- *SaveToBitmap(TBitmap) e SaveToStrings(TStrings)* – O método destas operações foi redefinido nesta classe para adicionar o comportamento de segurança de dados. Se a propriedade *SegurancaDados* estiver referenciando uma instância, retira a segurança conforme o estado da instância de *TFsdrSegurancaDados*, caso contrário, o comportamento do método é o mesmo da sua superclasse.

Apêndice III

Implementação do framework Frasedare

Na linguagem *Object Pascal*, uma unidade de implementação é dividida principalmente em duas estruturas, a *interface* e *implementation (implementação)*. Abaixo segue a implementação do framework Frasedare, considerando as duas principais estruturas de uma unidade de implementação.

- **Interfaces**

```
{ TFsdrTextoSeguro }

// Tipo de dados registro para agrupar Sigilo e Integridade
TFsdrTextoSeguro = record
  Sigilo: String;
  Integridade: String;
end;

{ TFsdrMecanismoCriptografia }

// Super classe para os métodos criptográficos do framework
TFsdrMecanismoCriptografia = class(TComponent)
protected
  // Hook
  procedure Inicializar(); virtual; abstract;
  // Hook
  procedure Finalizar(); virtual; abstract;
public
  // Padrão de Projeto Method Template
  constructor Create(AOwner: TComponent); override;
  // Padrão de Projeto Method Template
  destructor Destroy(); override;
end;

{ TFsdrSigilo }

// Super classe para os métodos criptográficos que implementam sigilo
TFsdrSigilo = class(TFsdrMecanismoCriptografia)
private
  FOnDadosCifradosCorrompidos: TNotifyEvent;
public
  function Cifrar(const aTextoClaro: String): String; virtual; abstract;
  function Decifrar(const aTextoCifrado: String): String; virtual; abstract;
  procedure DadosCifradosCorrompidos;
published
  property OnDadosCifradosCorrompidos: TNotifyEvent read FOnDadosCifradosCorrompidos
write FOnDadosCifradosCorrompidos;
end;

{ TFsdrIntegridade }

// Super classe para os métodos criptográficos que implementam integridade
TFsdrIntegridade = class(TFsdrMecanismoCriptografia)
```

```
private
  FOnIntegridadeViolada: TNotifyEvent;
public
  function Aplicar(const aTextoClaro: String): String; virtual; abstract;
  function Verificar(const aTextoClaro, aTextoIntegridade: String): Boolean; virtual;
abstract;
  procedure IntegridadeViolada;
published
  property OnIntegridadeViolada: TNotifyEvent read FOnIntegridadeViolada write
FOnIntegridadeViolada;
end;

{ TFsdrDeposCertChave }

// Padrão de Projeto Singleton
// Super classe para depósito de certificados e chaves
TFsdrDeposCertChave = class(TObject)
private
public
  procedure LimparCertificado(); virtual; abstract;
  class function NewInstance: TObject; override;
  procedure FreeInstance; override;
end;

{ TFsdrCriptoSimetrica }

// Sigilo
TFsdrCriptoSimetrica = class(TFsdrSigilo)
private
  FSenha: String;
  // Template Method
  procedure SetSenha(aSenha: String);
protected
  // Hook
  procedure ConfigurarSenha(); virtual; abstract;
public
  // Deve ser reimplementado nas subclasses
  function Cifrar(const aTextoClaro: String): String; override;
published
  property Senha: String read FSenha write SetSenha;
end;

{ TFsdrCriptoAssimetrica }

// Sigilo e Autenticação
TFsdrCriptoAssimetrica = class(TFsdrSigilo)
private
  FDeposCertChave: TFsdrDeposCertChave;
protected
  // Hook
  procedure ConfigurarCertificado(); virtual; abstract;
public
  // Templated Method
  constructor Create(AOwner: TComponent); override;
  destructor Destroy(); override;
  // Template Method – Deve ser reimplementado nas subclasses
  function Cifrar(const aTextoClaro: String): String; override;
  // 1. Factory Method 2. Hook
  function DeposCertChave(): TFsdrDeposCertChave; virtual; abstract;
  function RetornarDeposCertChave(): TFsdrDeposCertChave;
end;

{ TFsdrAssinatura }

// Integridade, Autenticação e Não Repúdio
TFsdrAssinatura = class(TFsdrIntegridade)
private
  FDeposCertChave: TFsdrDeposCertChave;
protected
  // Hook
  procedure ConfigurarCertificado(); virtual; abstract;
  // Hook
  function Assinar(const aTextoClaro: String): String; virtual; abstract;
public
```

```
  constructor Create(AOwner: TComponent); override;
  destructor Destroy(); override;
  // Template Method
  function Aplicar(const aTextoClaro: String): String; override;
  // Factory Method
  function DeposCertChave(): TFsdrDeposCertChave; virtual; abstract;
  function RetornarDeposCertChave(): TFsdrDeposCertChave;
end;

{ TFsdrResumo }

// Integridade
TFsdrResumo = class(TFsdrIntegridade)
private
protected
  function Resumir(const aTextoClaro: String): String; virtual; abstract;
  public
  function Aplicar(const aTextoClaro: String): String; override;
  end;

{ TFsdrSegurancaDados }

// Aplicar e Retirar Segurança
TFsdrSegurancaDados = class(TComponent)
private
  FIntegridade: TFsdrIntegridade;
  FSigilo: TFsdrSigilo;
  function AplicarIntegridade(aTextoClaro: String): String;
  function AplicarSigilo(aTextoClaro: String): String;
  function RetirarSigilo(aTextoSeguro: String): String;
  function VerificarIntegridade(aTextoClaro, aTextoIntegridade: String): Boolean;
  procedure SetIntegridade(aIntegridade: TFsdrIntegridade);
  procedure SetSigilo(aSigilo: TFsdrSigilo);
public
  function AplicarSeguranca(aTextoClaro: String): TFsdrTextoSeguro;
  function RetirarSeguranca(aTextoSeguro: TFsdrTextoSeguro): String;
published
  property Integridade: TFsdrIntegridade read FIntegridade write SetIntegridade;
  property Sigilo: TFsdrSigilo read FSigilo write SetSigilo;
end;

{ TFsdrBlobField }

// Campo persistente e seguro para manipular dados binários
TFsdrBlobField = class(TBlobField)
private
  FDataFieldIntegridade: TField;
  FSegurancaDados: TFsdrSegurancaDados;
  function GetDataFieldIntegridade(): TField;
  function GetSegurancaDados(): TFsdrSegurancaDados;
  procedure LoadFromStrings(Strings: TStrings);
  procedure LoadFromBitmap(Bitmap: TBitmap);
  procedure LoadFromBlob(Blob: TBlobField);
  procedure SaveToStrings(Strings: TStrings);
  procedure SaveToBitmap(Bitmap: TBitmap);
  procedure SetDataFieldIntegridade(Value: TField);
  procedure SetSegurancaDados(Value: TFsdrSegurancaDados);
protected
  procedure AssignTo(Dest: TPersistent); override;
public
  destructor Destroy; override;
  procedure Assign(Source: TPersistent); override;
published
  // Esta propriedade deverá ser utilizada somente se a propriedade Integridade
  // de SegurancaDados tiver algum objeto associado.
  property DataFieldIntegridade: TField read GetDataFieldIntegridade write
SetDataFieldIntegridade;
  property SegurancaDados: TFsdrSegurancaDados read GetSegurancaDados write
SetSegurancaDados;
end;

{ TFsdrMemoField }

// Campo persistente e seguro para manipular dados tipo memo
```

```
TFsdrMemoField = class(TMemoField)
private
  FDataFieldIntegridade: TField;
  FSegurancaDados: TFsdrSegurancaDados;
  function GetDataFieldIntegridade(): TField;
  function GetSegurancaDados(): TFsdrSegurancaDados;
  procedure SetDataFieldIntegridade(Value: TField);
  procedure SetSegurancaDados(Value: TFsdrSegurancaDados);
protected
  function GetAsString: String; override;
  procedure SetAsString(const Value: String); override;
public
  destructor Destroy; override;
published
  // Esta propriedade deverá ser utilizada somente se a propriedade Integridade
  // de SegurancaDados tiver algum objeto associado.
  property DataFieldIntegridade: TField read GetDataFieldIntegridade write
SetDataFieldIntegridade;
  property SegurancaDados: TFsdrSegurancaDados read GetSegurancaDados write
SetSegurancaDados;
end;

{ TFsdrStringField }

// Campo persistente e seguro para manipular dados tipo string
TFsdrStringField = class(TStringField)
private
  FDataFieldIntegridade: TField;
  FSegurancaDados: TFsdrSegurancaDados;
  function GetDataFieldIntegridade(): TField;
  function GetSegurancaDados(): TFsdrSegurancaDados;
  procedure SetDataFieldIntegridade(Value: TField);
  procedure SetSegurancaDados(Value: TFsdrSegurancaDados);
protected
  function GetAsString: String; override;
  procedure SetAsString(const Value: String); override;
public
  destructor Destroy; override;
published
  // Esta propriedade deverá ser utilizada somente se a propriedade Integridade
  // de SegurancaDados tiver algum objeto associado.
  property DataFieldIntegridade: TField read GetDataFieldIntegridade write
SetDataFieldIntegridade;
  property SegurancaDados: TFsdrSegurancaDados read GetSegurancaDados write
SetSegurancaDados;
end;
```

- **Implementação**

```
var
  // Utilizado para o padrão de projeto Singleton - Delphi não possui atributos
  // estáticos, por este motivo foram utilizadas variáveis globais.
  DeposCertChave: TObject;
  DeposCertChaveContarRef: Byte;

{ TFsdrMecanismoCriptografia }

constructor TFsdrMecanismoCriptografia.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  Inicializar();
end;

destructor TFsdrMecanismoCriptografia.Destroy();
begin
  Finalizar();
  inherited Destroy();
end;

{ TFsdrSigilo }

procedure TFsdrSigilo.DadosCifradosCorrompidos;
```

```
begin
  if Assigned(FOnDadosCifradosCorrompidos) then
    FOnDadosCifradosCorrompidos(Self)
  else
    raise Exception.Create('Dados Cifrados Estão Corrompidos.');
end;

{ TFsdrIntegridade }

procedure TFsdrIntegridade.IntegridadeViolada;
begin
  if Assigned(FOnIntegridadeViolada) then
    FOnIntegridadeViolada(Self)
  else
    raise Exception.Create('Integridade dos Dados foi Violada.');
end;

{ TFsdrCriptoSimetrica }

procedure TFsdrCriptoSimetrica.SetSenha(aSenha: String);
begin
  FSenha := aSenha;
  ConfigurarSenha();
end;

function TFsdrCriptoSimetrica.Cifrar(const aTextoClaro: String): String;
begin
  if Length(Trim(FSenha)) = 0 then
    raise Exception.Create('Senha para criptografia simétrica deve ser informada.');
end;

{ TFsdrCriptoAssimetrica }

constructor TFsdrCriptoAssimetrica.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FDeposCertChave := DeposCertChave();
end;

destructor TFsdrCriptoAssimetrica.Destroy();
begin
  if Assigned(FDeposCertChave) then
    FDeposCertChave.Free;
  inherited Destroy();
end;

function TFsdrCriptoAssimetrica.Cifrar(const aTextoClaro: String): String;
begin
  ConfigurarCertificado();
end;

function TFsdrCriptoAssimetrica.RetornarDeposCertChave(): TFsdrDeposCertChave;
begin
  if not Assigned(FDeposCertChave) then
    raise Exception.Create('Depósito de Certificado/Chave deve ser informado.');
  Result := FDeposCertChave;
end;

{ TFsdrAssinatura }

constructor TFsdrAssinatura.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  FDeposCertChave := DeposCertChave();
end;

destructor TFsdrAssinatura.Destroy();
begin
  if Assigned(FDeposCertChave) then
    FDeposCertChave.Free;
  inherited Destroy();
end;

function TFsdrAssinatura.RetornarDeposCertChave(): TFsdrDeposCertChave;
```

```
begin
  if not Assigned(FDeposCertChave) then
    raise Exception.Create('Depósito de Certificado/Chave deve ser informado.');
  Result := FDeposCertChave;
end;

function TFsdrAssinatura.Aplicar(const aTextoClaro: String): String;
begin
  ConfigurarCertificado();
  Result := Assinar(aTextoClaro);
end;

{ TFsdrResumo }

function TFsdrResumo.Aplicar(const aTextoClaro: String): String;
begin
  Result := Resumir(aTextoClaro);
end;

{ TFsdrDeposCertChave } // Padrão de Projeto Singleton

class function TFsdrDeposCertChave.NewInstance(): TObject;
begin
  if not Assigned(DeposCertChave) then begin
    DeposCertChave := inherited NewInstance();
  end;
  Inc(DeposCertChaveContarRef);
  Result := DeposCertChave;
end;

procedure TFsdrDeposCertChave.FreeInstance();
begin
  Dec(DeposCertChaveContarRef);
  if (DeposCertChaveContarRef = 0) then begin
    DeposCertChave := nil;
    inherited FreeInstance();
  end;
end;

{ TFsdrSegurancaDados }

function TFsdrSegurancaDados.AplicarSigilo(aTextoClaro: String): String;
begin
  // Sigilo Sem Autenticação
  if Assigned(FSigilo) then begin
    Result := FSigilo.Cifrar(aTextoClaro);
  end
  else begin
    Result := aTextoClaro;
  end;
end;

function TFsdrSegurancaDados.AplicarIntegridade(aTextoClaro: String): String;
begin
  if Assigned(FIntegridade) then
    Result := FIntegridade.Aplicar(aTextoClaro)
  else
    Result := '';
end;

function TFsdrSegurancaDados.RetirarSigilo(aTextoSeguro: String): String;
begin
  if Assigned(FSigilo) then begin
    try
      Result := FSigilo.Decifrar(aTextoSeguro);
    except
      FSigilo.DadosCifradosCorrompidos();
    end;
  end
  else begin
    Result := aTextoSeguro
  end;
end;
```

```
function TFsdrSegurancaDados.VerificarIntegridade(aTextoClaro, aTextoIntegridade:
String): Boolean;
begin
  Result := TRUE;
  // Verifica a Integridade
  if (Assigned(FIntegridade) and (Length(Trim(aTextoIntegridade)) <> 0)) then begin
    if not FIntegridade.Verificar(aTextoClaro, aTextoIntegridade) then
      Result := FALSE;
  end;
end;

function TFsdrSegurancaDados.AplicarSeguranca(aTextoClaro: String): TFsdrTextoSeguro;
var
  TextoSigilo, TextoIntegridade: String;

begin
  if Length(aTextoClaro) <> 0 then begin
    // Caso for cifra assimétrica ou assinatura, limpar o certificado selecionado
    // anteriormente.
    if (FSigilo is TFsdrCriptoAssimetrica) or (FIntegridade is TFsdrAssinatura) then
begin
      // Se os dois tipos de objetos estiverem sendo utilizandos,
      // executar somente um método LimparCertificado(), pois o DeposCertChave é
      // um singleton e ambos referenciam o mesmo objeto.
      if (FSigilo is TFsdrCriptoAssimetrica) then
        (FSigilo as TFsdrCriptoAssimetrica).RetornarDeposCertChave().LimparCertificado()
      else
        (FIntegridade as TFsdrAssinatura).RetornarDeposCertChave().LimparCertificado();
    end;

    TextoSigilo := AplicarSigilo(aTextoClaro);

    TextoIntegridade := AplicarIntegridade(aTextoClaro);

    // Compõem o texto seguro
    Result.Sigilo := TextoSigilo;
    Result.Integridade :=  TextoIntegridade;
  end;
end;

function TFsdrSegurancaDados.RetirarSeguranca(aTextoSeguro: TFsdrTextoSeguro): String;
var
  TextoSigilo, TextoClaro, TextoIntegridade: String;
begin
  if (Length(aTextoSeguro.Sigilo) <> 0) or (Length(aTextoSeguro.Integridade) <> 0) then
begin
    TextoSigilo      := aTextoSeguro.Sigilo;
    TextoIntegridade := aTextoSeguro.Integridade;
    TextoClaro := RetirarSigilo(TextoSigilo);

    if not VerificarIntegridade(TextoClaro, TextoIntegridade) then
      FIntegridade.IntegridadeViolada;

    Result := TextoClaro;
  end;
end;

procedure TFsdrSegurancaDados.SetIntegridade(aIntegridade: TFsdrIntegridade);
begin
  FIntegridade := aIntegridade;
end;

procedure TFsdrSegurancaDados.SetSigilo(aSigilo: TFsdrSigilo);
begin
  FSigilo := aSigilo;
end;
{ BLOB header }

type
  TGraphicHeader = record
    Count: Word;               { Fixed at 1 }
    HType: Word;               { Fixed at $0100 }
    Size: Longint;             { Size not including header }
  end;
```

```
{ TFsdrBlobField }

// Foi necessário redefinir as operações invocadas pela Assign(Source) para
// adicionar a segurança, pois na superclasse eram operações de visibilidade privada.
procedure TFsdrBlobField.Assign(Source: TPersistent);
begin
  if Source is TBlobField then
  begin
    LoadFromBlob(TBlobField(Source));
    Exit;
  end;

  if Source is TStrings then
  begin
    LoadFromStrings(TStrings(Source));
    Exit;
  end;

  if Source is TBitmap then
  begin
    LoadFromBitmap(TBitmap(Source));
    Exit;
  end;

  if (Source is TPicture) and (TPicture(Source).Graphic is TBitmap) then
  begin
    LoadFromBitmap(TBitmap(TPicture(Source).Graphic));
    Exit;
  end;

  inherited Assign(Source);
end;

// Foi necessário redefinir as operações invocadas pela AssignTo(Dest) para
// adicionar a segurança, pois na superclasse eram operações de visibilidade privada.
procedure TFsdrBlobField.AssignTo(Dest: TPersistent);
begin
  if Dest is TStrings then
  begin
    SaveToStrings(TStrings(Dest));
    Exit;
  end;

  if Dest is TBitmap then
  begin
    SaveToBitmap(TBitmap(Dest));
    Exit;
  end;

  if Dest is TPicture then
  begin
    SaveToBitmap(TPicture(Dest).Bitmap);
    Exit;
  end;

  inherited AssignTo(Dest);
end;

procedure TFsdrBlobField.LoadFromStrings(Strings: TStrings);
var
  BlobStream: TStream;
  BlobStreamLeitura: TMemoryStream;
  TextoSeguro: TFsdrTextoSeguro;
  Valor: String;
begin
  // Cria um Stream para aplicar a segurança nos dados em memória
  BlobStreamLeitura := TMemoryStream.Create();
  try
    BlobStreamLeitura.Clear;
    Strings.SaveToStream(BlobStreamLeitura);
    BlobStreamLeitura.Position := 0;
    SetString(Valor, nil, BlobStreamLeitura.Size);
    BlobStreamLeitura.Read(Pointer(Valor)^, BlobStreamLeitura.Size);
```

```
  finally
    BlobStreamLeitura.Free;
  end;

  if Assigned(FSegurancaDados) then begin
    TextoSeguro := FSegurancaDados.AplicarSeguranca(Valor);
    if Assigned(FSegurancaDados.Integridade) and Assigned(FDataFieldIntegridade) then
begin
      FDataFieldIntegridade.AsString := TextoSeguro.Integridade;
    end;
  end
  else
    TextoSeguro.Sigilo := Valor;

  // Copia os dados seguros para um TField do tipo Blob
  BlobStream := DataSet.CreateBlobStream(Self, bmWrite);
  try
    BlobStream.Write(Pointer(TextoSeguro.Sigilo)^, Length(TextoSeguro.Sigilo));
  finally
    BlobStream.Free;
  end;
end;

procedure TFsdrBlobField.LoadFromBitmap(Bitmap: TBitmap);
var
  BlobStreamLeitura: TMemoryStream;
  BlobStream: TStream;
  Header: TGraphicHeader;
  TextoSeguro: TFsdrTextoSeguro;
  Valor: String;
begin
  // Cria um Stream para aplicar a segurança nos dados em memória
  BlobStreamLeitura := TMemoryStream.Create();
  try
    if (DataType = ftGraphic) or (DataType = ftTypedBinary) then
    begin
      Header.Count := 1;
      Header.HType := $0100;
      Header.Size := 0;
      BlobStreamLeitura.Write(Header, SizeOf(Header));
      Bitmap.SaveToStream(BlobStreamLeitura);
      Header.Size := BlobStreamLeitura.Position - SizeOf(Header);
      BlobStreamLeitura.Position := 0;
      BlobStreamLeitura.Write(Header, SizeOf(Header));
    end else
      Bitmap.SaveToStream(BlobStreamLeitura);

    BlobStreamLeitura.Position := 0;
    SetString(Valor, nil, BlobStreamLeitura.Size);
    BlobStreamLeitura.Read(Pointer(Valor)^, BlobStreamLeitura.Size);
  finally
    BlobStreamLeitura.Free;
  end;

  if Assigned(FSegurancaDados) then begin
    TextoSeguro := FSegurancaDados.AplicarSeguranca(Valor);
    if Assigned(FSegurancaDados.Integridade) and Assigned(FDataFieldIntegridade) then
begin
      FDataFieldIntegridade.AsString := TextoSeguro.Integridade;
    end;
  end
  else
    TextoSeguro.Sigilo := Valor;

  // Copia os dados seguros para um TField do tipo Blob
  BlobStream := DataSet.CreateBlobStream(Self, bmWrite);
  try
    BlobStream.Write(Pointer(TextoSeguro.Sigilo)^, Length(TextoSeguro.Sigilo));
  finally
    BlobStream.Free;
  end;
end;

procedure TFsdrBlobField.LoadFromBlob(Blob: TBlobField);
```

```
var
  BlobStreamLeitura: TMemoryStream;
  BlobStream: TStream;
  TextoSeguro: TFsdrTextoSeguro;
  Valor: String;

begin
  // Cria um Stream para aplicar a segurança nos dados em memória
  BlobStreamLeitura := TMemoryStream.Create();
  try
    BlobStreamLeitura.Clear;
    Blob.SaveToStream(BlobStreamLeitura);
    BlobStreamLeitura.Position := 0;
    SetString(Valor, nil, BlobStreamLeitura.Size);
    BlobStreamLeitura.Read(Pointer(Valor)^, BlobStreamLeitura.Size);
  finally
    BlobStreamLeitura.Free;
  end;

  if Assigned(FSegurancaDados) then begin
    TextoSeguro := FSegurancaDados.AplicarSeguranca(Valor);
    if Assigned(FSegurancaDados.Integridade) and Assigned(FDataFieldIntegridade) then
begin
      FDataFieldIntegridade.AsString := TextoSeguro.Integridade;
    end;
  end
  else
    TextoSeguro.Sigilo := Valor;

  // Copia os dados seguros para um TField do tipo Blob
  BlobStream := DataSet.CreateBlobStream(Self, bmWrite);
  try
    BlobStream.Write(Pointer(TextoSeguro.Sigilo)^, Length(TextoSeguro.Sigilo));
  finally
    BlobStream.Free;
  end;
end;

procedure TFsdrBlobField.SaveToStrings(Strings: TStrings);
var
  BlobStream: TMemoryStream;
  BlobStreamSeguro: TStream;
  TextoSeguro: TFsdrTextoSeguro;
  Valor, ValorSemSeguranca: String;

begin
  // Cria um Stream para retirar a segurança nos dados em memória
  BlobStreamSeguro := DataSet.CreateBlobStream(Self, bmRead);
  try
    BlobStreamSeguro.Position := 0;
    SetString(Valor, nil, BlobStreamSeguro.Size);
    BlobStreamSeguro.Read(Pointer(Valor)^, BlobStreamSeguro.Size);

    if Assigned(FSegurancaDados) then begin
      TextoSeguro.Sigilo      := Valor;
      if Assigned(FSegurancaDados.Integridade) and Assigned(FDataFieldIntegridade) then
begin
        TextoSeguro.Integridade := FDataFieldIntegridade.AsString;
      end;
      ValorSemSeguranca := FSegurancaDados.RetirarSeguranca(TextoSeguro)
    end
    else
      ValorSemSeguranca := Valor;

  finally
    BlobStreamSeguro.Free;
  end;

  // Copia os dados sem segurança
  BlobStream := TMemoryStream.Create;
  try
    BlobStream.Write(Pointer(ValorSemSeguranca)^, Length(ValorSemSeguranca));
    BlobStream.Position := 0;
    Strings.LoadFromStream(BlobStream);
```

```
  finally
    BlobStream.Free;
  end;
end;

procedure TFsdrBlobField.SaveToBitmap(Bitmap: TBitmap);
var
  BlobStream: TMemoryStream;
  BlobStreamSeguro: TStream;
  Size: Longint;
  Header: TGraphicHeader;
  TextoSeguro: TFsdrTextoSeguro;
  Valor, ValorSemSeguranca: String;
begin
  // Cria um Stream para retirar a segurança nos dados em memória
  BlobStreamSeguro := DataSet.CreateBlobStream(Self, bmRead);
  try
    BlobStreamSeguro.Position := 0;
    SetString(Valor, nil, BlobStreamSeguro.Size);
    BlobStreamSeguro.Read(Pointer(Valor)^, BlobStreamSeguro.Size);

    if Assigned(FSegurancaDados) then begin
      TextoSeguro.Sigilo      := Valor;
      if Assigned(FSegurancaDados.Integridade) and Assigned(FDataFieldIntegridade) then
begin
        TextoSeguro.Integridade := FDataFieldIntegridade.AsString;
      end;
      ValorSemSeguranca := FSegurancaDados.RetirarSeguranca(TextoSeguro)
    end
    else
      ValorSemSeguranca := Valor;

  finally
    BlobStreamSeguro.Free;
  end;

  // Copia os dados sem segurança
  BlobStream := TMemoryStream.Create;
  try
    BlobStream.Write(Pointer(ValorSemSeguranca)^, Length(ValorSemSeguranca));
    BlobStream.Position := 0;

    Size := BlobStream.Size;
    if Size >= SizeOf(TGraphicHeader) then
    begin
      BlobStream.Read(Header, SizeOf(Header));
      if (Header.Count <> 1) or (Header.HType <> $0100) or
        (Header.Size <> Size - SizeOf(Header)) then
        BlobStream.Position := 0;
    end;
    Bitmap.LoadFromStream(BlobStream);
  finally
    BlobStream.Free;
  end;
end;

destructor TFsdrBlobField.Destroy;
begin
  if Assigned(FSegurancaDados) then
    FSegurancaDados := nil;
  if Assigned(FDataFieldIntegridade) then
    FDataFieldIntegridade := nil;
  inherited Destroy;
end;

function TFsdrBlobField.GetDataFieldIntegridade(): TField;
begin
  Result := FDataFieldIntegridade;
end;

function TFsdrBlobField.GetSegurancaDados(): TFsdrSegurancaDados;
begin
  Result := FSegurancaDados;
end;
```

```
procedure TFsdrBlobField.SetDataFieldIntegridade(Value: TField);
begin
  if FDataFieldIntegridade <> Value then
    FDataFieldIntegridade := Value;
end;

procedure TFsdrBlobField.SetSegurancaDados(Value: TFsdrSegurancaDados);
begin
  if FSegurancaDados <> Value then
    FSegurancaDados := Value;
end;

{ TFsdrMemoField }

destructor TFsdrMemoField.Destroy;
begin
  if Assigned(FSegurancaDados) then
    FSegurancaDados := nil;
  if Assigned(FDataFieldIntegridade) then
    FDataFieldIntegridade := nil;
  inherited Destroy;
end;

function TFsdrMemoField.GetDataFieldIntegridade(): TField;
begin
  Result := FDataFieldIntegridade;
end;

function TFsdrMemoField.GetSegurancaDados(): TFsdrSegurancaDados;
begin
  Result := FSegurancaDados;
end;

procedure TFsdrMemoField.SetDataFieldIntegridade(Value: TField);
begin
  if FDataFieldIntegridade <> Value then
    FDataFieldIntegridade := Value;
end;

procedure TFsdrMemoField.SetSegurancaDados(Value: TFsdrSegurancaDados);
begin
  if FSegurancaDados <> Value then
    FSegurancaDados := Value;
end;

function TFsdrMemoField.GetAsString: String;
var
  TextoSeguro: TFsdrTextoSeguro;
begin
  if Assigned(FSegurancaDados) then begin
    TextoSeguro.Sigilo      := inherited GetAsString();
    if Assigned(FSegurancaDados.Integridade) and Assigned(FDataFieldIntegridade) then
begin
      TextoSeguro.Integridade := FDataFieldIntegridade.AsString;
    end;
    Result := FSegurancaDados.RetirarSeguranca(TextoSeguro)
  end
  else
    Result := inherited GetAsString();
end;

procedure TFsdrMemoField.SetAsString(const Value: String);
var
  TextoSeguro: TFsdrTextoSeguro;
begin
  if Assigned(FSegurancaDados) then begin

    TextoSeguro := FSegurancaDados.AplicarSeguranca(Value);
    if Assigned(FSegurancaDados.Integridade) and Assigned(FDataFieldIntegridade) then
begin
      FDataFieldIntegridade.AsString := TextoSeguro.Integridade;
    end;
```

```
  end
  else
    TextoSeguro.Sigilo := Value;

  inherited SetAsString(TextoSeguro.Sigilo);
end;

{ TFsdrStringField }

destructor TFsdrStringField.Destroy;
begin
  if Assigned(FSegurancaDados) then
    FSegurancaDados := nil;
  if Assigned(FDataFieldIntegridade) then
    FDataFieldIntegridade := nil;
  inherited Destroy;
end;

function TFsdrStringField.GetDataFieldIntegridade(): TField;
begin
  Result := FDataFieldIntegridade;
end;

function TFsdrStringField.GetSegurancaDados(): TFsdrSegurancaDados;
begin
  Result := FSegurancaDados;
end;

procedure TFsdrStringField.SetDataFieldIntegridade(Value: TField);
begin
  if FDataFieldIntegridade <> Value then
    FDataFieldIntegridade := Value;
end;

procedure TFsdrStringField.SetSegurancaDados(Value: TFsdrSegurancaDados);
begin
  if FSegurancaDados <> Value then
    FSegurancaDados := Value;
end;

function TFsdrStringField.GetAsString: String;
var
  TextoSeguro: TFsdrTextoSeguro;
begin
  if Assigned(FSegurancaDados) then begin
    TextoSeguro.Sigilo      := inherited GetAsString();
    if Assigned(FSegurancaDados.Integridade) and Assigned(FDataFieldIntegridade) then
begin
      TextoSeguro.Integridade := FDataFieldIntegridade.AsString;
    end;
    Result := FSegurancaDados.RetirarSeguranca(TextoSeguro)
  end
  else
    Result := inherited GetAsString();
end;

procedure TFsdrStringField.SetAsString(const Value: String);
var
  TextoSeguro: TFsdrTextoSeguro;
begin
  if Assigned(FSegurancaDados) then begin
    TextoSeguro := FSegurancaDados.AplicarSeguranca(Value);
    if Assigned(FSegurancaDados.Integridade) and Assigned(FDataFieldIntegridade) then
begin
      FDataFieldIntegridade.AsString := TextoSeguro.Integridade;
    end;
  end
  else
    TextoSeguro.Sigilo := Value;

  inherited SetAsString(TextoSeguro.Sigilo);
end;

procedure Register;
```

```
begin
  RegisterComponents('FraSeDaRe', [TFsdrSegurancaDados]);
  RegisterFields([TFsdrMemoField, TFsdrStringField, TFsdrBlobField]);
end;
```

Apêndice IV

Uso do framework Frasedare – Aplicação Teste

A explicação sobre a estrutura dada no apêndice III, também deve ser considerada neste apêndice. Abaixo segue a implementação do subsistema estendendo o framework Frasedare utilizado na aplicação teste.

- **Interfaces**

```
{ TFsdrCriptoSimetrica_AES }

// Sigilo - AES
TFsdrCriptoSimetrica_AES = class(TFsdrCriptoSimetrica)
private
  FDadosCifrados: TEncryptedData;
protected
  procedure ConfigurarSenha(); override;
public
  function Cifrar(const aTextoClaro: String): String; override;
  function Decifrar(const aTextoCifrado: String): String; override;
  // Implementação Hook
  procedure Inicializar(); override;
  // Implementação Hook
  procedure Finalizar(); override;
end;

{ TFsdrCriptoAssimetrica_RSA_DES }

// Sigilo e Autenticação - RSA, DES
TFsdrCriptoAssimetrica_RSA_DES = class(TFsdrCriptoAssimetrica)
private
  FDadosCifradosComChaveAssimetrica: TEnvelopedData;
protected
  procedure ConfigurarCertificado(); override;
public
  function Cifrar(const aTextoClaro: String): String; override;
  function Decifrar(const aTextoCifrado: String): String; override;
  // Factory Method
  function DeposCertChave(): TFsdrDeposCertChave; override;
  // Implementação Hook
  procedure Inicializar(); override;
  // Implementação Hook
  procedure Finalizar(); override;
end;

{ TFsdrAssinatura_RSA }

// Integridade, Autenticação e Não Repúdio - RSA
// Padrão de Projeto Adapter
TFsdrAssinatura_RSA = class(TFsdrAssinatura)
private
  FAssinatura: TSignedData;
  FCertificado: OleVariant;
protected
```

```
  procedure ConfigurarCertificado(); override;
  function Assinar(const aTextoClaro: String): String; override;
public
  // Factory Method
  function DeposCertChave(): TFsdrDeposCertChave; override;
  function Verificar(const aTextoClaro, aAssinatura: String): Boolean; override;
  // Implementação Hook
  procedure Inicializar(); override;
  // Implementação Hook
  procedure Finalizar(); override;
end;

{ TFsdrResumo_SHA1 }

// Integridade dos Dados - SHA1
TFsdrResumo_SHA1 = class(TFsdrResumo)
private
  FDadosResumidos: THashedData;
public
  function Resumir(const aTextoClaro: String): String; override;
  function Verificar(const aTextoClaro, aResumo: String): Boolean; override;
  // Implementação Hook
  procedure Inicializar(); override;
  // Implementação Hook
  procedure Finalizar(); override;
end;

{ TFsdrDeposCertChave_Win32 }

// Depósito de chaves/certificados
// Padrão de Projeto Singleton
TFsdrDeposCertChave_Win32 = class(TFsdrDeposCertChave)
private
  FCertificado: ICertificate;
protected
  function SelecionarCertificado(): ICertificate;
public
  procedure LimparCertificado(); override;
  destructor Destroy(); override;
  function RetornarCertificado(): ICertificate;
end;
```

- **Implementação**

```
{ TFsdrCriptoSimetrica_AES }

procedure TFsdrCriptoSimetrica_AES.Inicializar();
begin
  FDadosCifrados := TEncryptedData.Create(nil);
  // Configura com algoritmo AES
  FDadosCifrados.Algorithm.Name := CAPICOM_ENCRYPTION_ALGORITHM_AES;
end;

procedure TFsdrCriptoSimetrica_AES.Finalizar();
begin
  FDadosCifrados.Free;
end;

function TFsdrCriptoSimetrica_AES.Cifrar(const aTextoClaro: String): String;
begin
  inherited Cifrar(aTextoClaro);
  FDadosCifrados.Content := aTextoClaro;
  // Retorna o texto cifrado
  Result := FDadosCifrados.Encrypt(CAPICOM_ENCODE_BASE64);
end;

function TFsdrCriptoSimetrica_AES.Decifrar(const aTextoCifrado: String): String;
begin
  // Decifra o texto
  FDadosCifrados.Decrypt(aTextoCifrado);
  // Retorna o texto claro
  Result := FDadosCifrados.Content;
```

```
end;

procedure TFsdrCriptoSimetrica_AES.ConfigurarSenha();
begin
  FDadosCifrados.SetSecret(Senha, CAPICOM_SECRET_PASSWORD); ;
end;


{ TFsdrCriptoAssimetrica_RSA_DES }

procedure TFsdrCriptoAssimetrica_RSA_DES.Inicializar();
begin
  FDadosCifradosComChaveAssimetrica := TEnvelopedData.Create(nil);
  // Configura com algoritmo DES
  FDadosCifradosComChaveAssimetrica.Algorithm.Name := CAPICOM_ENCRYPTION_ALGORITHM_DES;
  FDadosCifradosComChaveAssimetrica.Algorithm.KeyLength :=
CAPICOM_ENCRYPTION_KEY_LENGTH_56_BITS;
end;

procedure TFsdrCriptoAssimetrica_RSA_DES.Finalizar();
begin
  FDadosCifradosComChaveAssimetrica.Free;
end;

function TFsdrCriptoAssimetrica_RSA_DES.DeposCertChave(): TFsdrDeposCertChave;
begin
  Result := TFsdrDeposCertChave_Win32.Create();
end;

procedure TFsdrCriptoAssimetrica_RSA_DES.ConfigurarCertificado();
begin
  FDadosCifradosComChaveAssimetrica.Recipients.Clear;
  FDadosCifradosComChaveAssimetrica.Recipients.Add((RetornarDeposCertChave() as
TFsdrDeposCertChave_Win32).RetornarCertificado());
end;

function TFsdrCriptoAssimetrica_RSA_DES.Cifrar(const aTextoClaro: String): String;
begin
  inherited Cifrar(aTextoClaro);
  FDadosCifradosComChaveAssimetrica.Content := aTextoClaro;
  Result := FDadosCifradosComChaveAssimetrica.Encrypt(CAPICOM_ENCODE_BASE64);
end;

function TFsdrCriptoAssimetrica_RSA_DES.Decifrar(const aTextoCifrado: String): String;
begin
  // Não precisa preencher o Recipients, pois o método procura a chave privada
  // correspondente a chave pública utilizada para o cifrar a chave de sessão.
  FDadosCifradosComChaveAssimetrica.Decrypt(aTextoCifrado);
  Result := FDadosCifradosComChaveAssimetrica.Content;
end;

{ TFsdrAssinatura_RSA }

procedure TFsdrAssinatura_RSA.Inicializar();
begin
  FAssinatura := TSignedData.Create(nil);
end;

procedure TFsdrAssinatura_RSA.Finalizar();
begin
  FAssinatura.Free;
end;

procedure TFsdrAssinatura_RSA.ConfigurarCertificado();
begin
  // Seleciona o certificado
  FCertificado := (RetornarDeposCertChave() as
TFsdrDeposCertChave_Win32).RetornarCertificado();
end;

function TFsdrAssinatura_RSA.Assinar(const aTextoClaro: String): String;
var
  OVSignatario: OleVariant;
  Signatario: ISigner;
```

```
begin
  // Cria o signatário
  OVSignatario := CreateOleObject('CAPICOM.Signer');
  OVSignatario.Certificate := FCertificado;
  IDispatch(OVSignatario).QueryInterface(ISigner, Signatario);

  with FAssinatura do begin
    Content := aTextoClaro;
    Result  := Sign(Signatario, TRUE, CAPICOM_ENCODE_BASE64);
  end;
end;

function TFsdrAssinatura_RSA.DeposCertChave(): TFsdrDeposCertChave;
begin
  Result := TFsdrDeposCertChave_Win32.Create();
end;

function TFsdrAssinatura_RSA.Verificar(const aTextoClaro, aAssinatura: String): Boolean;
begin
  with FAssinatura do begin
    Content := aTextoClaro;
    try
      Verify(aAssinatura, TRUE, CAPICOM_VERIFY_SIGNATURE_ONLY);
      Result := TRUE;
    except
      Result := FALSE;
    end;
  end;
end;

{ TFsdrResumo_SHA1 }

procedure TFsdrResumo_SHA1.Inicializar();
begin
  FDadosResumidos := THashedData.Create(nil);
  // Configura com algoritmo DES
  FDadosResumidos.Algorithm := CAPICOM_HASH_ALGORITHM_SHA1;
end;

procedure TFsdrResumo_SHA1.Finalizar();
begin
  FDadosResumidos.Free;
end;

function TFsdrResumo_SHA1.Resumir(const aTextoClaro: String): String;
begin
  FDadosResumidos.Hash(aTextoClaro);
  Result := FDadosResumidos.Value;
end;

function TFsdrResumo_SHA1.Verificar(const aTextoClaro, aResumo: String): Boolean;
begin
  FDadosResumidos.Hash(aTextoClaro);
  Result := (FDadosResumidos.Value = aResumo);
end;

{ TFsdrDeposCertChave_Win32 }

destructor TFsdrDeposCertChave_Win32.Destroy();
begin
  if Assigned(FCertificado) then
    FCertificado := Nil;
end;

function TFsdrDeposCertChave_Win32.SelecionarCertificado(): ICertificate;
var
  Deposito, Signatario, Certificados: OleVariant;
begin
  // Cria um objeto Ole de Store e Signer
  Deposito   := CreateOleObject('CAPICOM.Store');
  Signatario := CreateOleObject('CAPICOM.Signer');

  // Abre o repositório do usuário corrente
  Deposito.Open(CAPICOM_CURRENT_USER_STORE, 'MY', CAPICOM_STORE_OPEN_READ_ONLY);
```

```
  // Seleciona um certificado
  Certificados := Deposito.Certificates.Select('FraSeDaRe', 'Selecione o Certificado',
FALSE);

  // Retonar o primeiro certificado selecionado
  IDispatch(Certificados.Item[1]).QueryInterface(ICertificate, Result);
end;

function TFsdrDeposCertChave_Win32.RetornarCertificado(): ICertificate;
begin
  if not Assigned( FCertificado ) then begin
    FCertificado := SelecionarCertificado();
  end;
  Result := FCertificado;
end;

procedure TFsdrDeposCertChave_Win32.LimparCertificado();
begin
  FCertificado := Nil;
end;
```

Apêndice V

Uso do framework Frasedare – Autorização de Procedimentos Médicos no Sistema Blendus.

A explicação sobre a estrutura dada no apêndice III, também deve ser considerada neste apêndice. Abaixo segue a implementação do subsistema estendendo o framework Frasedare utilizado no sistema Blendus.

- **Interfaces**

```
{ TFsdrCriptoSimetrica_DES }

// Sigilo - DES
TFsdrCriptoSimetrica_DES = class(TFsdrCriptoSimetrica)
private
  FDadosCifrados: TEncryptedData;
protected
  procedure ConfigurarSenha(); override;
public
  function Cifrar(const aTextoClaro: String): String; override;
  function Decifrar(const aTextoCifrado: String): String; override;
  // Implementação Hook
  procedure Inicializar(); override;
  // Implementação Hook
  procedure Finalizar(); override;
end;


{ TFsdrAssinatura_RSA }
// Integridade, Autenticação e Não Repúdio - RSA
// Padrão de Projeto Adapter
TFsdrAssinatura_RSA = class(TFsdrAssinatura)
private
  FAssinatura: TSignedData;
  FCertificado: OleVariant;
protected
  procedure ConfigurarCertificado(); override;
  function Assinar(const aTextoClaro: String): String; override;
public
  // Factory Method
  function DeposCertChave(): TFsdrDeposCertChave; override;
  function Verificar(const aTextoClaro, aAssinatura: String): Boolean; override;
  // Implementação Hook
  procedure Inicializar(); override;
  // Implementação Hook
  procedure Finalizar(); override;
end;

{ TFsdrDeposCertChave_Win32 }

// Depósito de chaves/certificados
// Padrão de Projeto Singleton
TFsdrDeposCertChave_Win32 = class(TFsdrDeposCertChave)
```

```
private
  FCertificado: ICertificate;
protected
  function SelecionarCertificado(): ICertificate;
public
  procedure LimparCertificado(); override;
  destructor Destroy(); override;
  function RetornarCertificado(): ICertificate;
end;
```

- **Implementação**

```
{ TFsdrCriptoSimetrica_DES }

procedure TFsdrCriptoSimetrica_DES.Inicializar();
begin
  FDadosCifrados := TEncryptedData.Create(nil);
  // Configura com algoritmo DES
  FDadosCifrados.Algorithm.Name := CAPICOM_ENCRYPTION_ALGORITHM_DES;
end;

procedure TFsdrCriptoSimetrica_DES.Finalizar();
begin
  FDadosCifrados.Free;
end;

function TFsdrCriptoSimetrica_DES.Cifrar(const aTextoClaro: String): String;
begin
  inherited Cifrar(aTextoClaro);
  FDadosCifrados.Content := aTextoClaro;
  // Retorna o texto cifrado
  Result := FDadosCifrados.Encrypt(CAPICOM_ENCODE_BASE64);
end;

function TFsdrCriptoSimetrica_DES.Decifrar(const aTextoCifrado: String): String;
begin
  // Decifra o texto
  FDadosCifrados.Decrypt(aTextoCifrado);
  // Retorna o texto claro
  Result := FDadosCifrados.Content;
end;

procedure TFsdrCriptoSimetrica_DES.ConfigurarSenha();
begin
  FDadosCifrados.SetSecret(Senha, CAPICOM_SECRET_PASSWORD); ;
end;


{ TFsdrAssinatura_RSA }

procedure TFsdrAssinatura_RSA.Inicializar();
begin
  FAssinatura := TSignedData.Create(nil);
end;

procedure TFsdrAssinatura_RSA.Finalizar();
begin
  FAssinatura.Free;
end;

procedure TFsdrAssinatura_RSA.ConfigurarCertificado();
begin
  // Seleciona o certificado
  FCertificado := (RetornarDeposCertChave() as
TFsdrDeposCertChave_Win32).RetornarCertificado();
end;

function TFsdrAssinatura_RSA.Assinar(const aTextoClaro: String): String;
var
  OVSignatario: OleVariant;
  Signatario: ISigner;
```

```
begin
  // Cria o signatário
  OVSignatario := CreateOleObject('CAPICOM.Signer');
  OVSignatario.Certificate := FCertificado;
  IDispatch(OVSignatario).QueryInterface(ISigner, Signatario);

  with FAssinatura do begin
    Content := aTextoClaro;
    Result  := Sign(Signatario, TRUE, CAPICOM_ENCODE_BASE64);
  end;
end;

function TFsdrAssinatura_RSA.DeposCertChave(): TFsdrDeposCertChave;
begin
  Result := TFsdrDeposCertChave_Win32.Create();
end;

function TFsdrAssinatura_RSA.Verificar(const aTextoClaro, aAssinatura: String): Boolean;
begin
  with FAssinatura do begin
    Content := aTextoClaro;
    try
      Verify(aAssinatura, TRUE, CAPICOM_VERIFY_SIGNATURE_ONLY);
      Result := TRUE;
    except
      Result := FALSE;
    end;
  end;
end;

{ TFsdrDeposCertChave_Win32 }

destructor TFsdrDeposCertChave_Win32.Destroy();
begin
  if Assigned(FCertificado) then
    FCertificado := Nil;
end;

function TFsdrDeposCertChave_Win32.SelecionarCertificado(): ICertificate;
var
  Deposito, Signatario, Certificados: OleVariant;
begin
  // Cria um objeto Ole de Store e Signer
  Deposito   := CreateOleObject('CAPICOM.Store');
  Signatario := CreateOleObject('CAPICOM.Signer');

  // Abre o repositório do usuário corrente
  Deposito.Open(CAPICOM_CURRENT_USER_STORE, 'MY', CAPICOM_STORE_OPEN_READ_ONLY);

  // Seleciona um certificado
  Certificados := Deposito.Certificates.Select('FraSeDaRe', 'Selecione o Certificado',
FALSE);

  // Retonar o primeiro certificado selecionado
  IDispatch(Certificados.Item[1]).QueryInterface(ICertificate, Result);
end;

function TFsdrDeposCertChave_Win32.RetornarCertificado(): ICertificate;
begin
  if not Assigned( FCertificado ) then begin
    FCertificado := SelecionarCertificado();
  end;
  Result := FCertificado;
end;

procedure TFsdrDeposCertChave_Win32.LimparCertificado();
begin
  FCertificado := Nil;
end;
```