FRAMEWORK DE PESISTÊNCIA DE DADOS COM CRIPTOGRAFIA DE CHAVE ASSIMÉTRICA USANDO HIBERNATE

Acadêmico: Jean Carlos Pereira
Orientador: Alexander Roberto Valdameri

www.furb.br

Roteiro

- Introdução
- Objetivos do trabalho
- Fundamentação teórica
- Desenvolvimento do trabalho
- Conclusão
- Extensões

Introdução

- Processo de desenvolvimento de software demanda tempo e esforço
- Investimentos em segurança são proporcionais ao nível de segurança que se deseja obter
- Falhas de implementação ou do modelo de negócio podem permitir acesso não autorizado a informações confidenciais
- A adoção do *framework* Hibernate como mecanismo de persistência tem um custo significativo e exibe uma boa curva de aprendizagem
- Solução: Adicionar características de segurança ao *framework* Hibernate

Objetivos do trabalho

- Desenvolver um *framework* de persistência de dados que permita que as informações sejam mantidas criptografadas no sistema de armazenamento

- Minimizar o impacto para o emprego da criptografia nas aplicações que já utilizam o Hibernate através de um dialeto

- Prover implementações dos algoritmos de criptografia mais comumente utilizados

- Validar o framework através da implementação de uma aplicação de exemplo que utilize o Hibernate como mecanismo de persistência

Fundamentação teórica

- Conceitos
  - ✓ Banco de dados
  - ✓ *SQL*
  - ✓ *JDBC*
  - ✓ Hibernate
  - ✓ Segurança em banco de dados
  - ✓ Criptografia
  - ✓ Certificado digital

- Trabalhos correlatos
  - ✓ Empress
  - ✓ Frasedare

Banco de dados

É um sistema computadorizado de arquivamento de registros (Date, 1985).

- Mantido e acessado através de um Sistema Gerenciador de Banco de Dados (SGBD)

- O modelo mais utilizado é o relacional onde as estruturas tem formas de tabelas compostas por linhas e colunas

- Outros modelos menos utilizados são o orientado a objetos e o modelo de entidades e relacionamentos

- O mais comum atualmente são aplicações que utilizam o paradigma de orientação a objetos nas camadas de negócio e apresentação e o modelo relacional para a camada de persistência.

SQL

A SQL é uma linguagem padrão para se lidar com bancos de dados relacionais, e é aceita por quase todos os produtos existentes no mercado (Date, 1985)

- É um padrão de banco de dados
- Possui muitas variações e extensões
- Padronizada pela ANSI e ISO:
  - ✓ ANSI: 1986
  - ✓ ISO: 1987
  - ✓ SQL-2: 1992
  - ✓ SQL-3: 1999
  - ✓ SQL-2003: 2003

Subconjuntos da linguagem SQL

- DML – Selecionar, inserir, atualizar e apagar dados
- DDL – Definir tabelas e elementos associados
- DCL – Controle de acesso
- DQL – Consulta de dados

JDBC

- Facilita e padroniza a comunicação de aplicações Java com o banco de dados

- Possibilita a manipulação das informações no banco de dados

- *Drivers* JDBC implementam as interfaces da API JDBC

- A API JDBC suporta quatro tipos de *drivers*:
  - ✓ *driver* JDBC-ODBC (*driver* tipo 1)
  - ✓ *driver* de acesso nativo (*driver* tipo 2)
  - ✓ *driver* de acesso por *middleware* (*driver* tipo 3)
  - ✓ *driver* de acesso direto ao servidor (*driver* tipo 4)

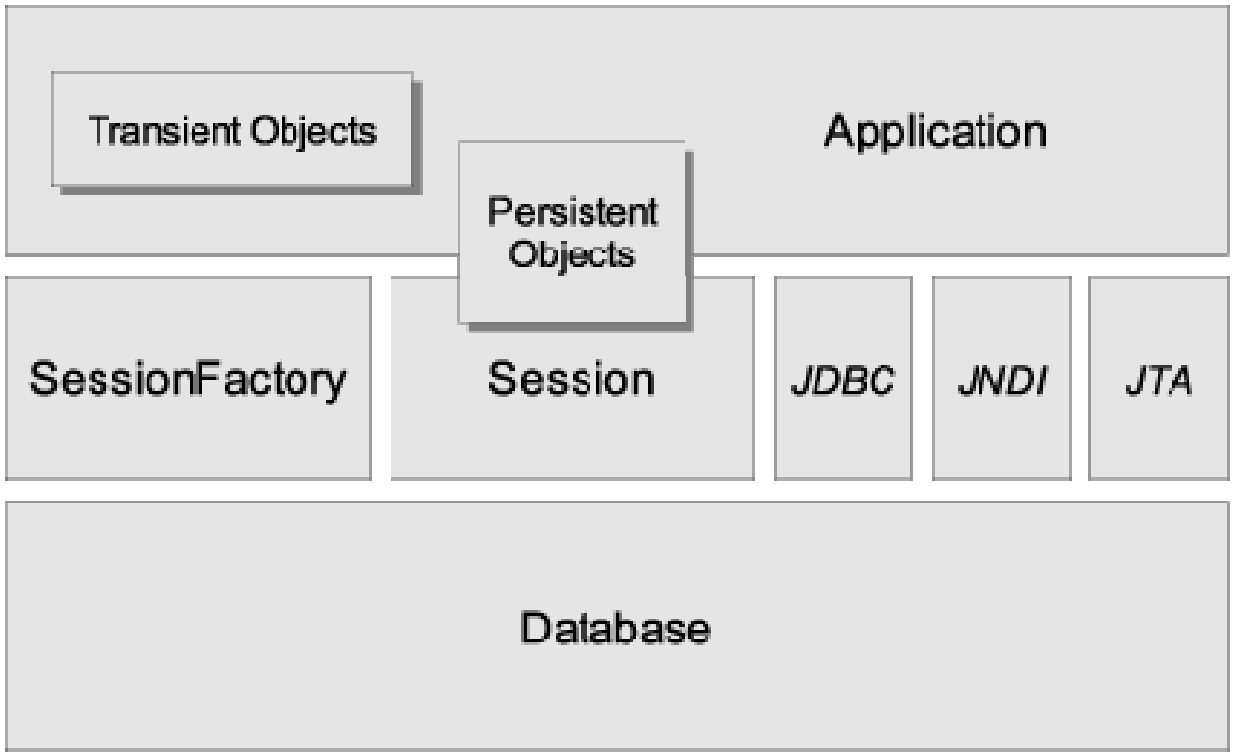
Hibernate

- Framework de persistência e acesso a banco de dados
- Simplifica o desenvolvimento de consultas e atualizações de dados
- Permite o mapeamento objeto-relacional diminuindo a complexidade do modelo híbrido (ORM)
- Permite alternar entre diferentes tipos de bancos de dados sem alterar o código da aplicação
- Pode ser utilizado com Java ou .NET
- Possui uma linguagem específica orientada a objetos (HQL)
- O mapemento pode ser feito através de arquivos XML ou de *anottations*

Bancos de dados suportados pelo Hibernate

- DB2
- Postgree
- MySQL
- Oracle
- Sybase
- Microsoft SQL Server
- SAP DB
- Informix
- Hypersonic DB
- Ingres
- Progress
- Mckoi SQL
- Interbase
- Pointbase
- FrontBase
- Firebird

Arquitetura Hibernate



Fonte: Hibernate, 2007

Segurança em banco de dados

- Geralmente é omitida, a segurança fica restrita a aplicação
- Existem poucas soluções prontas que integrem segurança ao SGBD
- Proteção contra:
  - ✓ Divulgação
  - ✓ Alteração
  - ✓ Destruição

Criptografia

- Torna a informação sigilosa a um adversário que possa vir a interceptá-la

- Cifragem é o processo pelo qual a criptografia torna dados codificados ou secretos, chamados de texto cifrado

- Decifragem é o processo inverso onde o texto cifrado é descodificado para possibilitar sua leitura

- A cifragem pode ser feita através de duas maneiras:
  - ✓ Chave simétrica ou chave secreta
  - ✓ Chave assimétrica ou chave pública

Certificado digital

- Liga uma entidade a uma chave pública
- É assinado digitalmetne
- Pode ser assinado por uma Autoridade Certificadora (CA)
- Possui validade e pode ser expirado pela CA
- O padrão mais utilizado é O X.509

Trabalhos correlatos

- SGBD Empress com tipos de dados com suporte a criptografia

- Frasedare: *Framework* orientado a objetos para segurança de dados em repolso.

Desenvolvimento do trabalho

- Requisitos funcionais e não funcionais
- Especificação do framework
- Diagrama de pacotes
- Diagrama de classes
- Diagrama de pacotes
- Implementação
- Operacionalidade

Requisitos funcionais

- Suportar a utilização de algoritmos de criptografia fornecidos pela aplicação que utiliza o *framework* através de interfaces

- Prover uma implementação padrão do algoritmo RSA

- Garantir que um objeto transiente seja persistido com os atributos criptografados quando assim definido

- Garantir que um objeto persistido seja materializado com os atributos decriptografados

- Permitir que um atributo seja assinado digitalmente

- Garantir a autenticidade da assinatura digital em um atributo que foi assinado

- Permitir que seja definido um atributo diferente para a persistência do valor cifrado ou da assinatura digital

Requisitos não funcionais

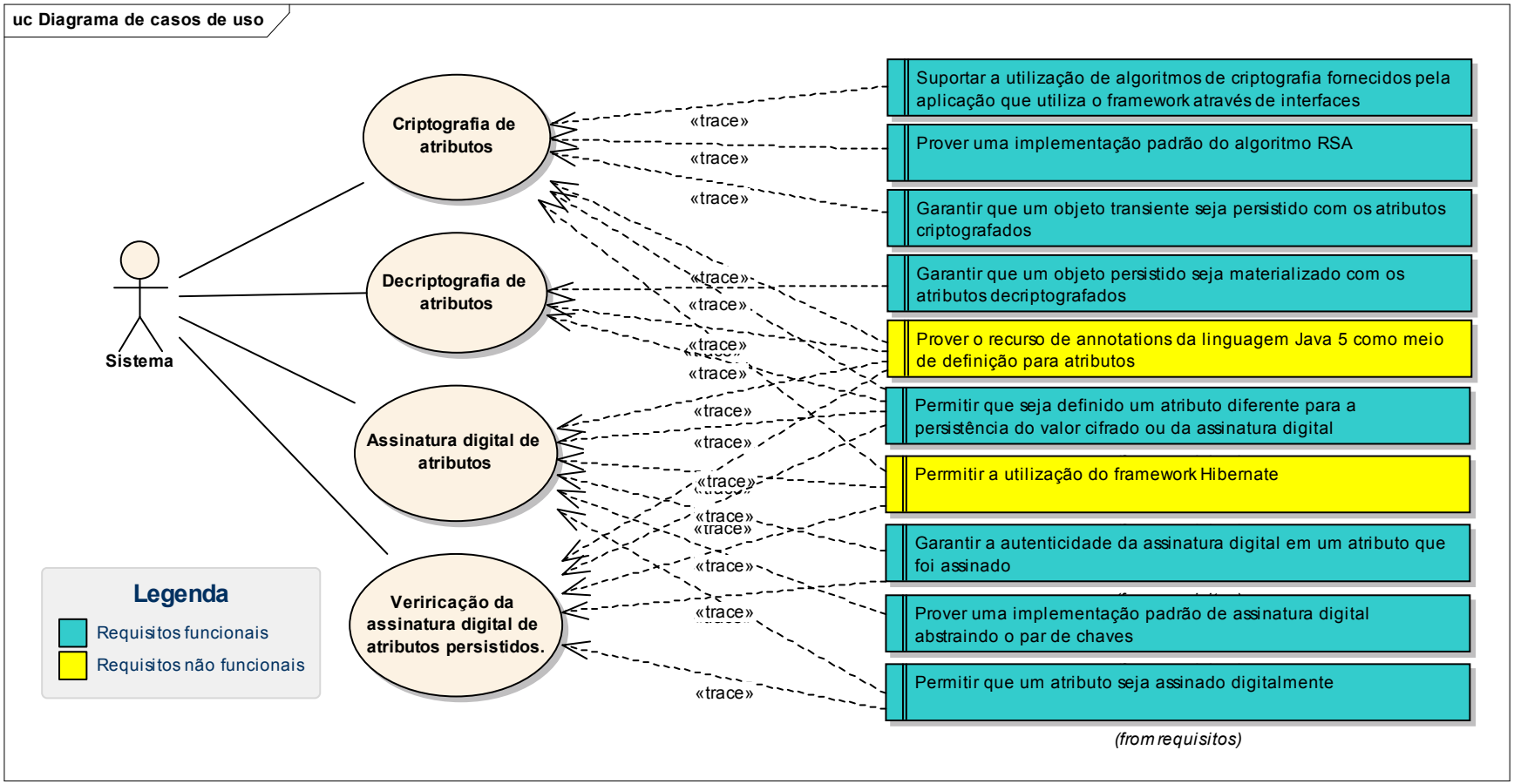
- Permitir a utilização do *framework* Hibernate

- Prover o recurso de *annotations* da linguagem Java 5 como meio de definição para atributos (RNF)

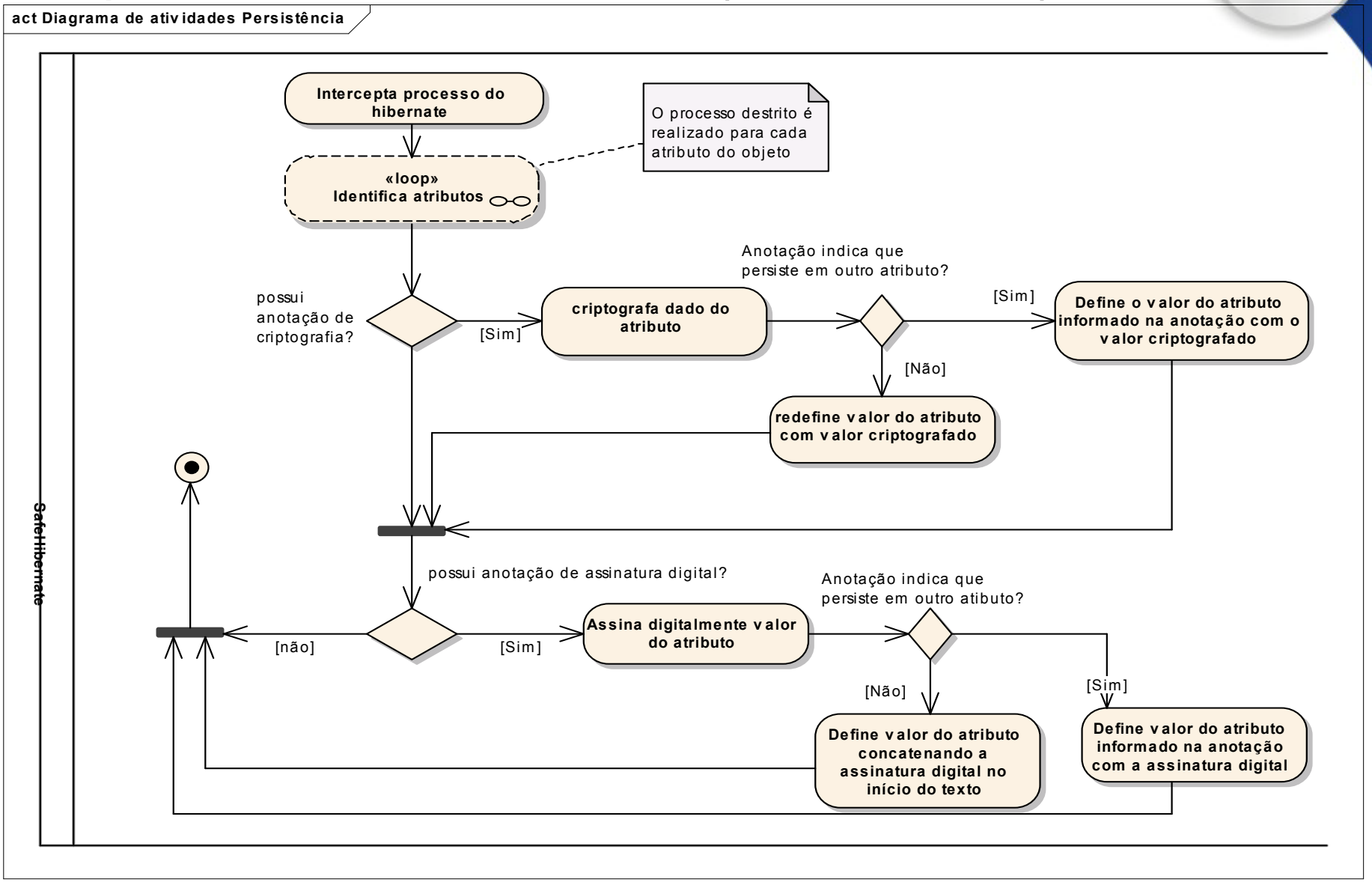
- Ser implementado em Java utilizando o ambiente de programação Eclipse

Especificação

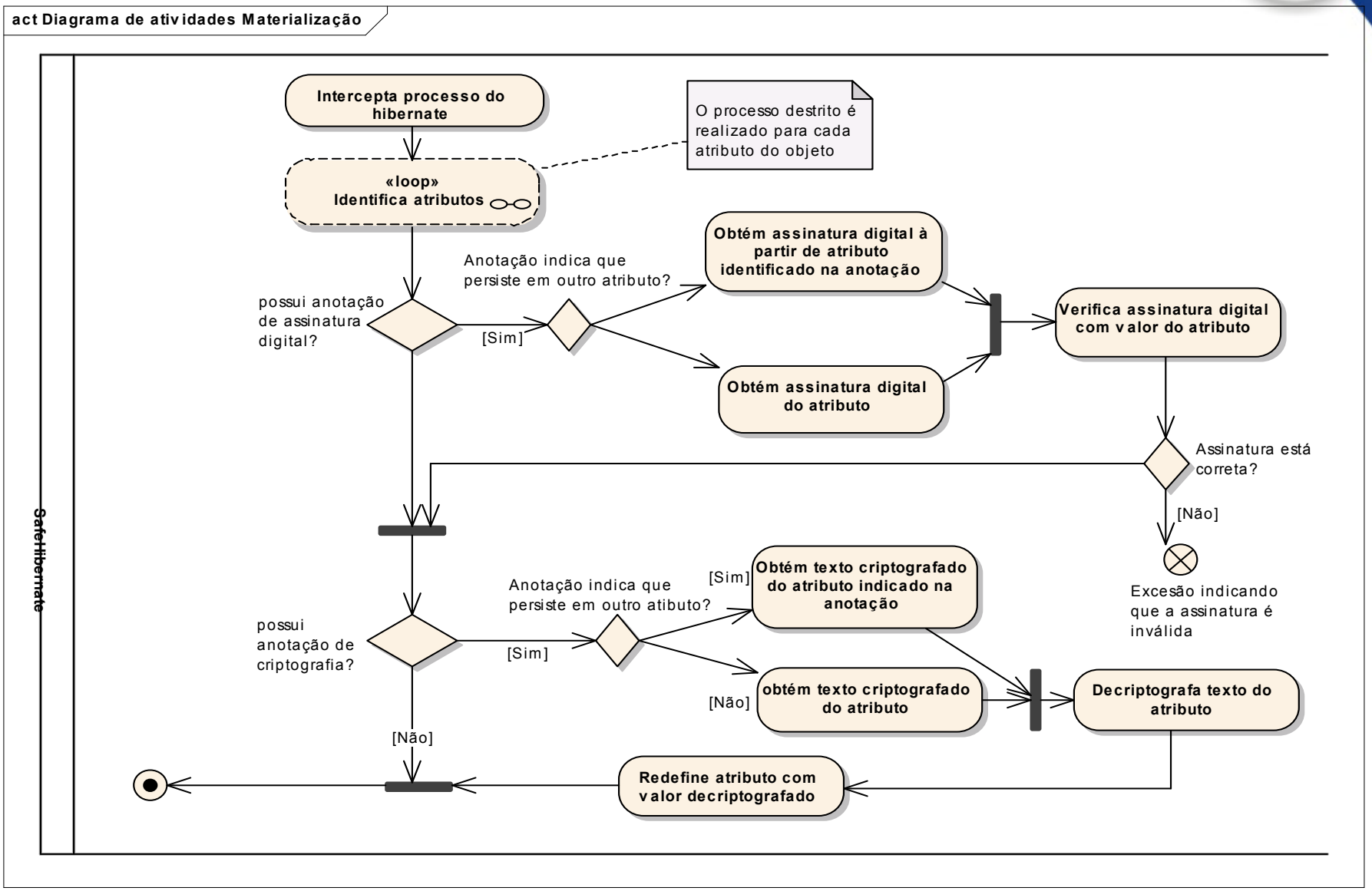
Casos de uso:



Especificação – atividades (Persistência)

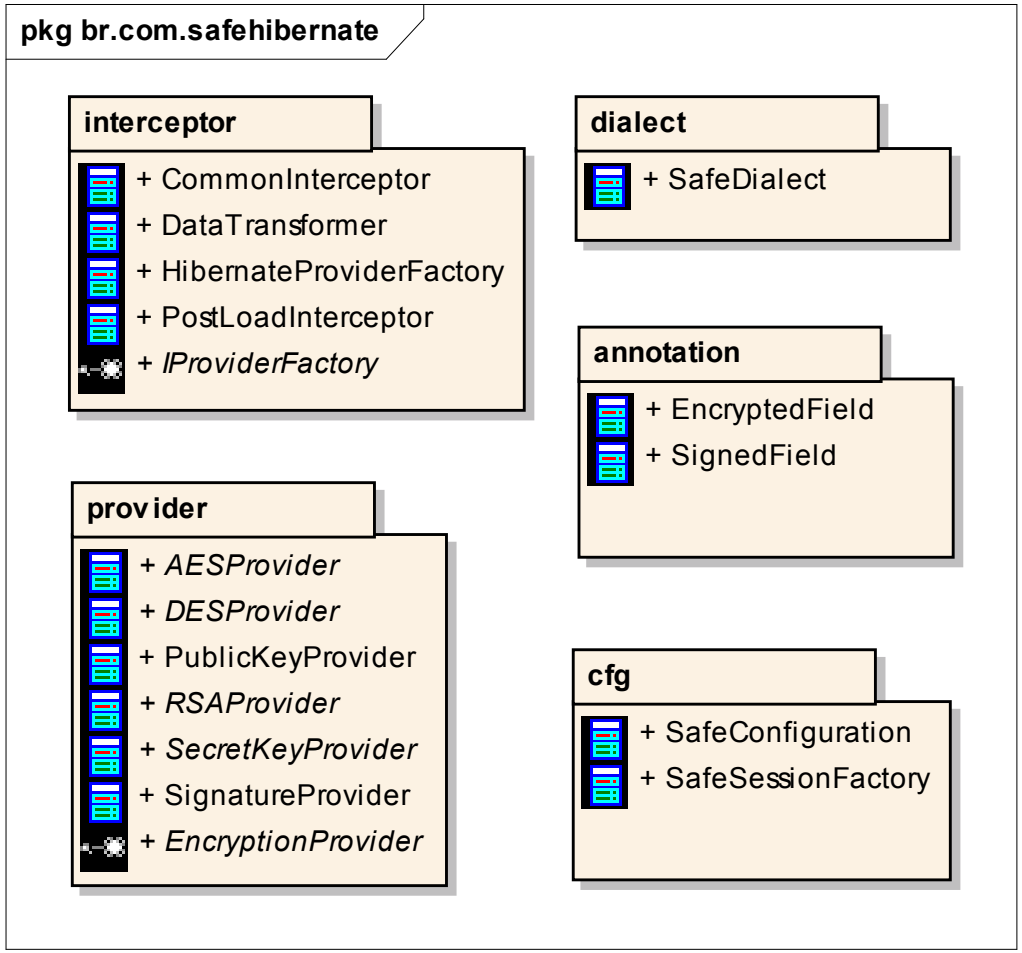


Especificação – Atividades (Materialização)



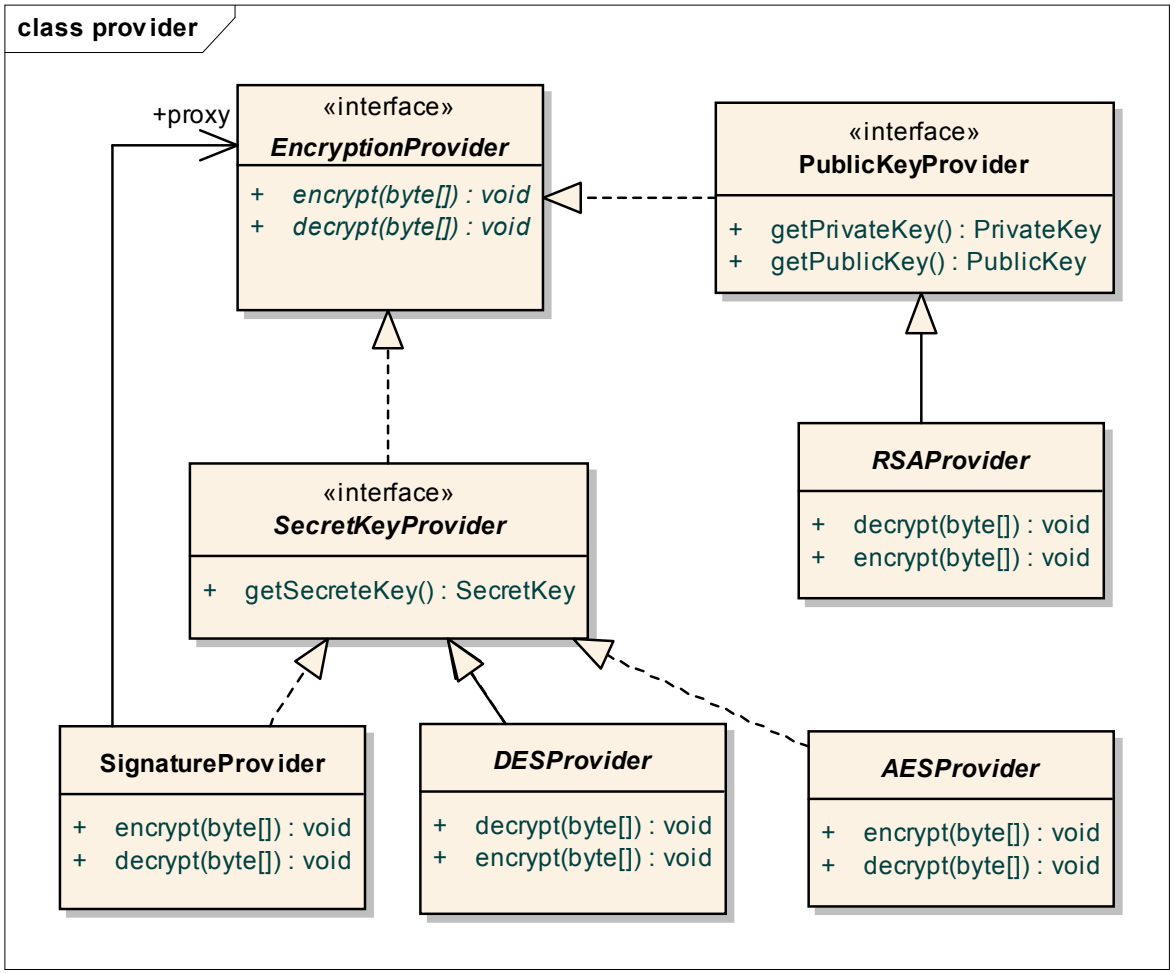
Especificação

- Diagrama de pacotes



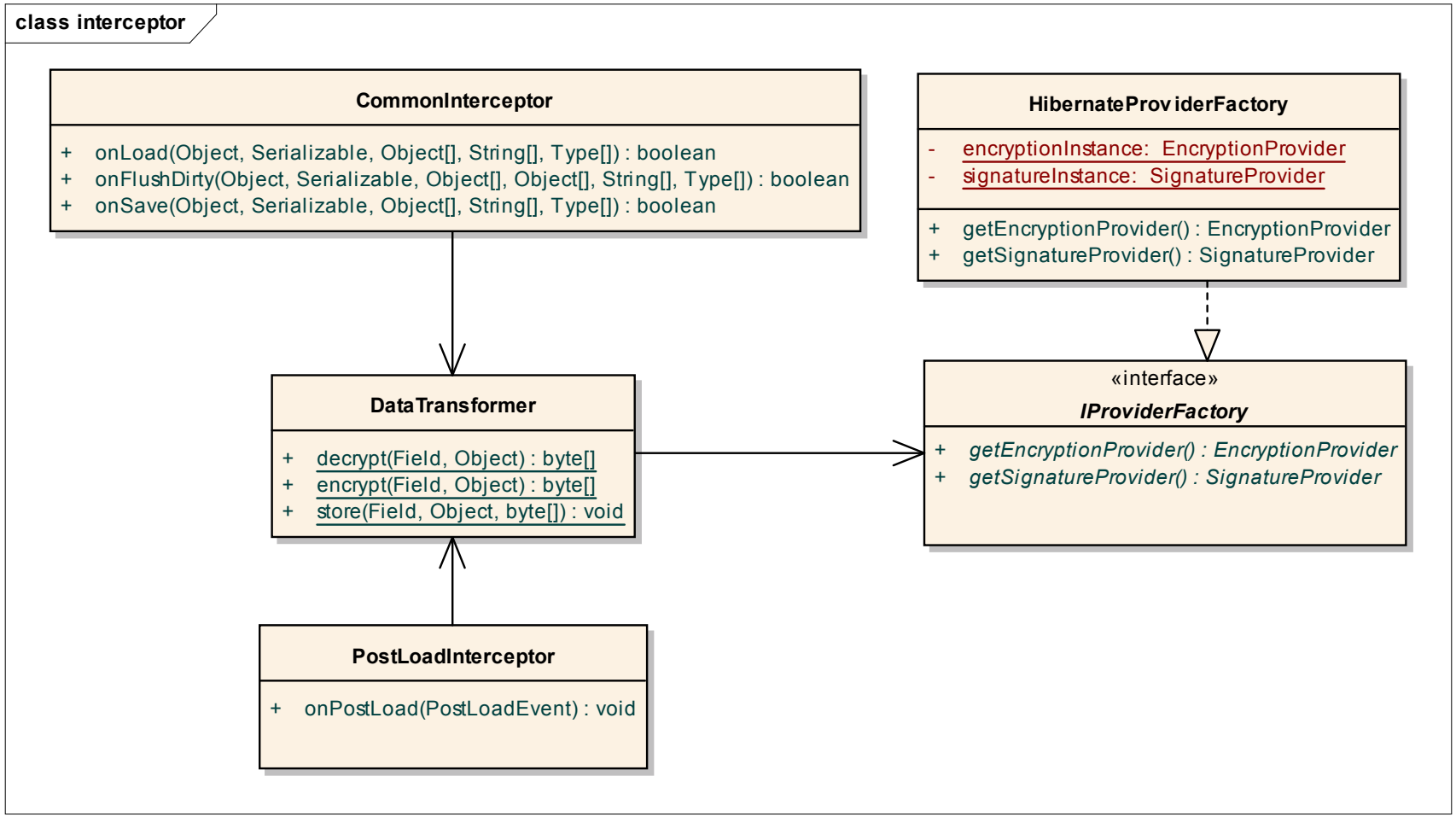
Especificação

- Diagrama de classes do pacote `provider`:



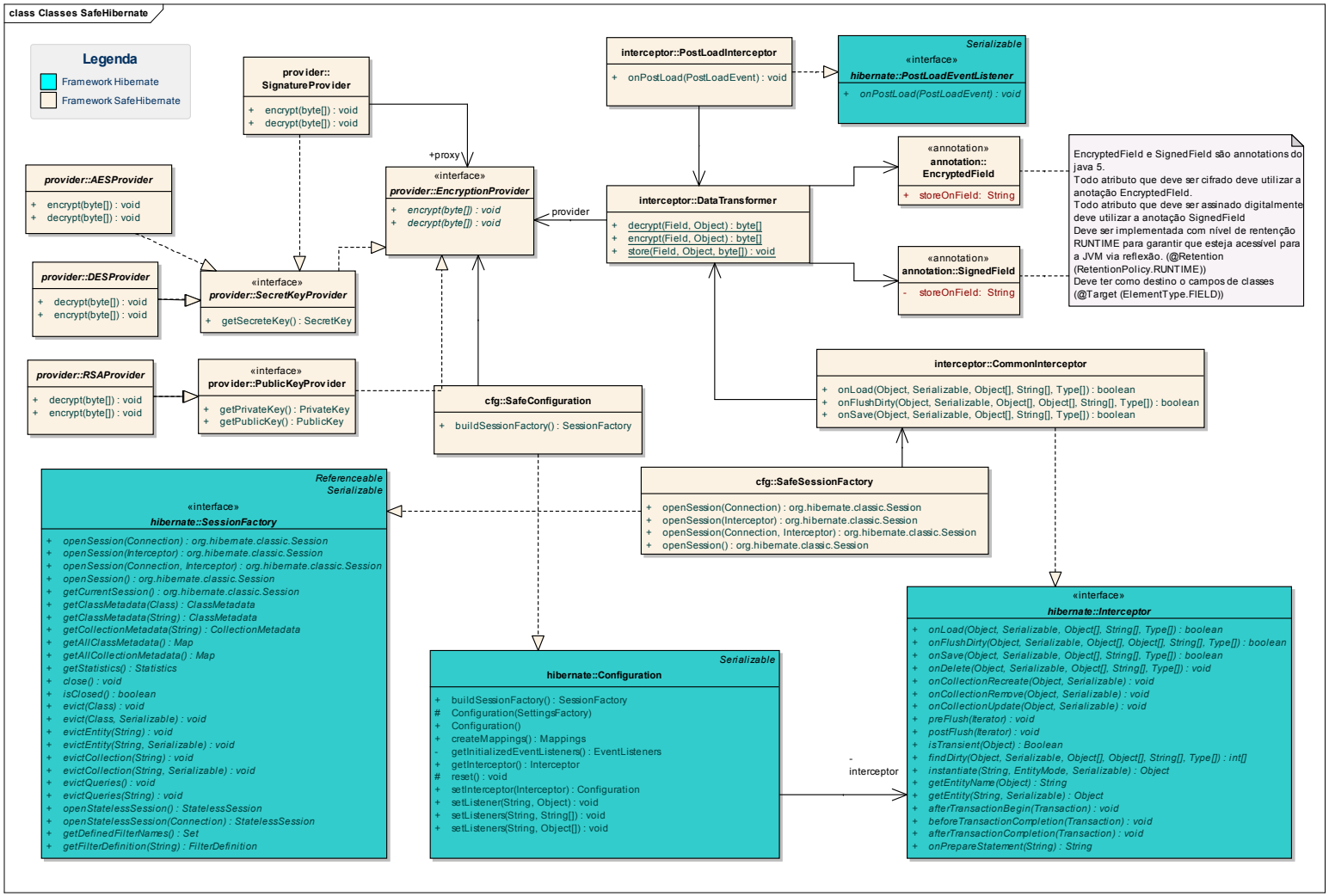
Especificação

- Diagrama de classes do pacote `interceptor`:



Especificação

- Diagrama de classes do framework:



Implementação

- Técnicas e ferramentas utilizadas:
  - ✓ Java 6.0
  - ✓ Eclipse 3.3
  - ✓ Hibernate 3.2.5
  - ✓ API Bouncy Castle
  - ✓ JBoss 4.2.1

Implementação

- *Annotation* para definição de atributos criptografados:

```
package br.com.safehibernate.annotation;

...
//Define a política de rentenção desta anotação.
//Definindo como Runtime esta anotação será mantida pela JVM
//durante a execução do programa, permitindo que seja obtida
//via reflexão.
@Retention (RetentionPolicy.RUNTIME)

//Define que esta anotação só pode ser utilizada em campos de classes
@Target (ElementType.FIELD)

@Documented
public @interface EncryptedField {

String storeOnField() default "this";

}
```

Implementação

- Método onSave da classe CommonInterceptor:

```
@Override
public boolean onSave(Object entity,
    Serializable id,
    Object[] state,
    String[] propertyNames,
    Type[] types) {

    Class<?> clazz = entity.getClass();
    for (int i = 0; i < propertyNames.length; i++) {
        String propertyName = propertyNames[i];
        try {
            Field field = clazz.getDeclaredField(propertyName);
            field.setAccessible(true);
            Object newValue = new DataTransformer().encrypt(field, entity);
            state[i] = newValue;
        } catch (SecurityException e) {
            throw new RuntimeException(e);
        } catch (NoSuchFieldException e) {
            throw new RuntimeException(e);
        } catch (IllegalArgumentException e) {
            throw new RuntimeException(e);
        }
    }
    return true;
}
```

Implementação

- DataTransformer: criptografa e assina um atributo ao persistir objetos:

```
public Object encrypt(Field field, Object object) {
    try {
        EncryptedField encryptionAnnotation = field.getAnnotation(EncryptedField.class);
        if (encryptionAnnotation != null) {
            Field encryptedField = field;
            String encryptFieldName = encryptionAnnotation.storeOnField();
            boolean sameField = "this".equals(encryptFieldName);
            if (!sameField) {
                encryptedField = object.getClass().getDeclaredField(encryptFieldName);
            }
            encryptedField.setAccessible(true);
            byte[] decryptedData =  String.valueOf(encryptedField.get(object)).getBytes();
            byte[] tempData = this.certificateLookup.getEncryptionProvider().encrypt(decryptedData);
            store(encryptedField, object, tempData);
        }
        //Assina o campo, se for necessário.
        SignedField signatureAnnotation = field.getAnnotation(SignedField.class);
        if (signatureAnnotation != null) {
            Field signatureField = field;
            String signatureFieldName = signatureAnnotation.storeOnField();
            boolean sameField = "this".equals(signatureFieldName);
            if (!sameField) {
                signatureField = object.getClass().getDeclaredField(signatureFieldName);
            }
            field.setAccessible(true);
            Object o = field.get(object);
            if (o != null) {
                byte[] signedData = String.valueOf(o).getBytes();
                byte[] tempData = this.certificateLookup.getSignatureProvider().encrypt(signedData);
                store(signatureField, object, tempData);
            }
        }
        field.setAccessible(true);
        return field.get(object);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
```

Operacionalidade

- A operacionalidade da aplicação se dá através da utilização do dialeto SafeHibernate em substituição ao dialeto Hibernate

- Como estudo de caso, foi implementada uma aplicação de exemplo que controla registros médicos de pacientes utilizando a seguinte arquitetura:

  - ✓ Java 6.0
  - ✓ Hibernate 3.2.5
  - ✓ JBoss 4.2.1
  - ✓ Swing
  - ✓ MySQL

Operacionalidade

- Adições realizadas no arquivo hibernate.properties

```
#Configuração para acesso ao bando de dados
connection.driver_class=com.mysql.jdbc.Driver
connection.url=jdbc:mysql://localhost/medicalapp
connection.username=hibernate
connection.password=hibernate
#Configura o safeHibernate para utilizar MySql como dialeto primário
safeHibernate.wrappedDialect=org.hibernate.dialect.MySQLDialect
#Define qual é a classe que provê acesso ao certificado do usuário logado.
safeHibernate.encryptionProvider=br.com.medical.util.MedicalEncryptionProvider
safeHibernate.signatureProvider=br.com.medical.util.MedicalSignatureProvider
#Debug
show_sql=true
hibernate.format_sql=true
hibernate.hbm2ddl.auto=update
```

Operacionalidade

- Adição das anotações a classe de Histórico do paciente

```
...
pubic class History implements Serializable {
    ...
        //O médico que realizou o atendimento.
        @SignedField(storeOnField="doctorCodeSignature")
        private Doctor doctor;

        private String doctorCodeSignature;
    ...
        @EncryptedField
        //Contém a descrição do atendimento realizado.
        private String description;
    ...
};
```

Resultados e discussão

- Atendimento dos requisitos propostos
- Impacto reduzido para o desenvolvedor que utiliza o framework
- Comparativo entre as ferramentas:

	Vários SGBDs?	Delphi	Java	.NET	Gerencia chaves	Algoritmos configuráveis	Facilidade de uso
SGBD Empress							X
Frasedare		X			X	X	
SafeHibernate	X		X	X		X	X

Conclusão

- Aplicação:
  - ✓ Baixo impacto no código legado através das anotações
  - ✓ Não requer mudanças arquiturais
  - ✓ Permite a adoção de bancos de dados diferentes sem alterações na criptografia
- Flexibilidade
  - ✓ Permite a utilização de qualquer algoritmo de criptografia fornecido pelo desenvolvedor
  - ✓ Pode ser adaptado para utilização sem o Hibernate
- Limitações
  - ✓ Tipos de dados complexos não suportados
  - ✓ Não suporta infra-estrutura de PKI

Extensões

- Construir uma estrutura de chave pública que permita a obtenção de certificados digitais e listas de certificados revogados

- Verificar a validade e autenticidade dos certificados associados as chaves públicas utilizadas na criptografia

- Fornecer um meio padrão para o gerenciamento de usuários e de chaves através do framework

- Interceptar consultas e projeções que envolvam somente alguns atributos de objetos e realizar a criptografia e ou assinatura digital nestes

Relevância pessoal

- Aplicação do aprendizado: Levantamento de requisitos, especificação, etc.
- Uso de recursos não documentados do Hibernate
- Aplicação real da tecnologia por usuários durante o desenvolvimento
- Conclusão da graduação como projeto pessoal
- Interesse em especialização em tecnologias para web