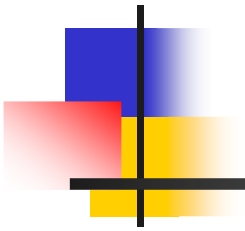
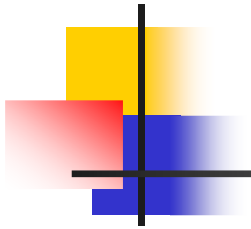


PLUGINS PARA TESTES AUTOMATIZADOS DE CONFORMIDADE COM A NORMA ISO/IEC 15408

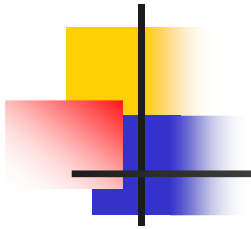
Dionei Herkenhoff

Orientador: Paulo Fernando da Silva



Roteiro

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



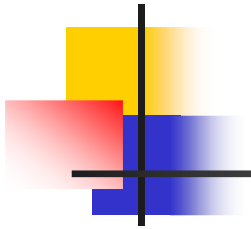
Introdução

- Informatização de processos
- Sistemas Distribuídos
- Segurança da informação
- ISO/IEC 15408

Objetivos do trabalho

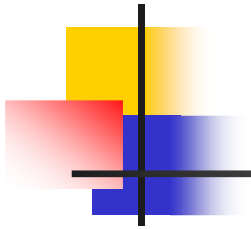
- Realizar testes automatizados FAU_GEN e FAU_SEL
- Realizar testes automatizados FAU_SAR, FAU_SAA e FAU_ARP
- realizar testes automatizados FIA_AFL, FIA_SOS e FIA_ATD

Fundamentação Teórica



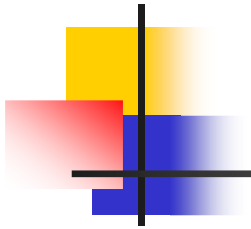
Segurança da Informação

- Confidencialidade
- Integridade
- Disponibilidade



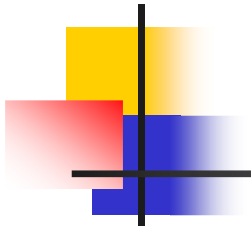
Norma ISO/IEC 15408

- Criado a partir de um conjunto de padrões
- Homologada em dezembro de 1999
- Requisitos de segurança



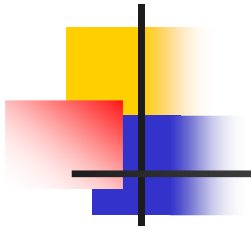
SCAN-CC

- Software de verificação de conformidades de segurança à norma ISO/IEC 15408



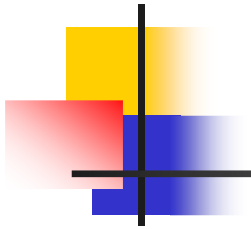
SCAN-CC

- Ferramenta realização e apoio no desenvolvimento de auditoria em software
- Possibilidade de desenvolvimento de testes automatizados de verificação de segurança
- Geração de relatório com o resultado dos testes



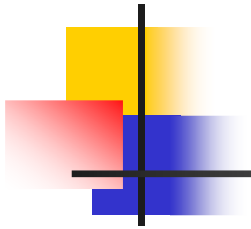
SCAN-CC

- Limitações
  - Testes automatizados
  - Verifica apenas sistemas desenvolvidos em JAVA



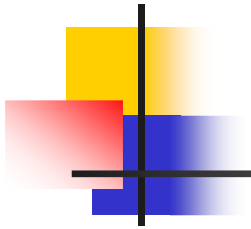
Trabalhos Correlatos

- IBM Rational AppScan
- Lapse



IBM Rational AppScan

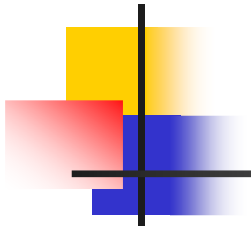
- Ferramenta proprietária
- Verifica softwares desenvolvidos para web
- ISO 17799 e 27001
- Custo alto da licença
- Apenas ambiente web



LAPSE

- Contração de Lightweight Analysis for Program Security in Eclipse
- Licença GNU - General Public License
- Verifica softwares desenvolvidos para web
- Não segue um padrão oficial de segurança

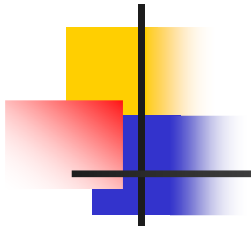
Especificação



Requisitos

- o sistema deve solicitar um arquivo do tipo *eXtensible Markup Language* (XML) com as configurações iniciais do sistema a ser verificado

- o sistema deve realizar a analise automática dos requisitos de auditoria de segurança



Requisitos

- o sistema deve realizar a analise automática dos requisitos de identificação e autenticação

- o sistema deve realizar a auditoria em sistemas desenvolvidos na linguagem JAVA

Casos de Uso

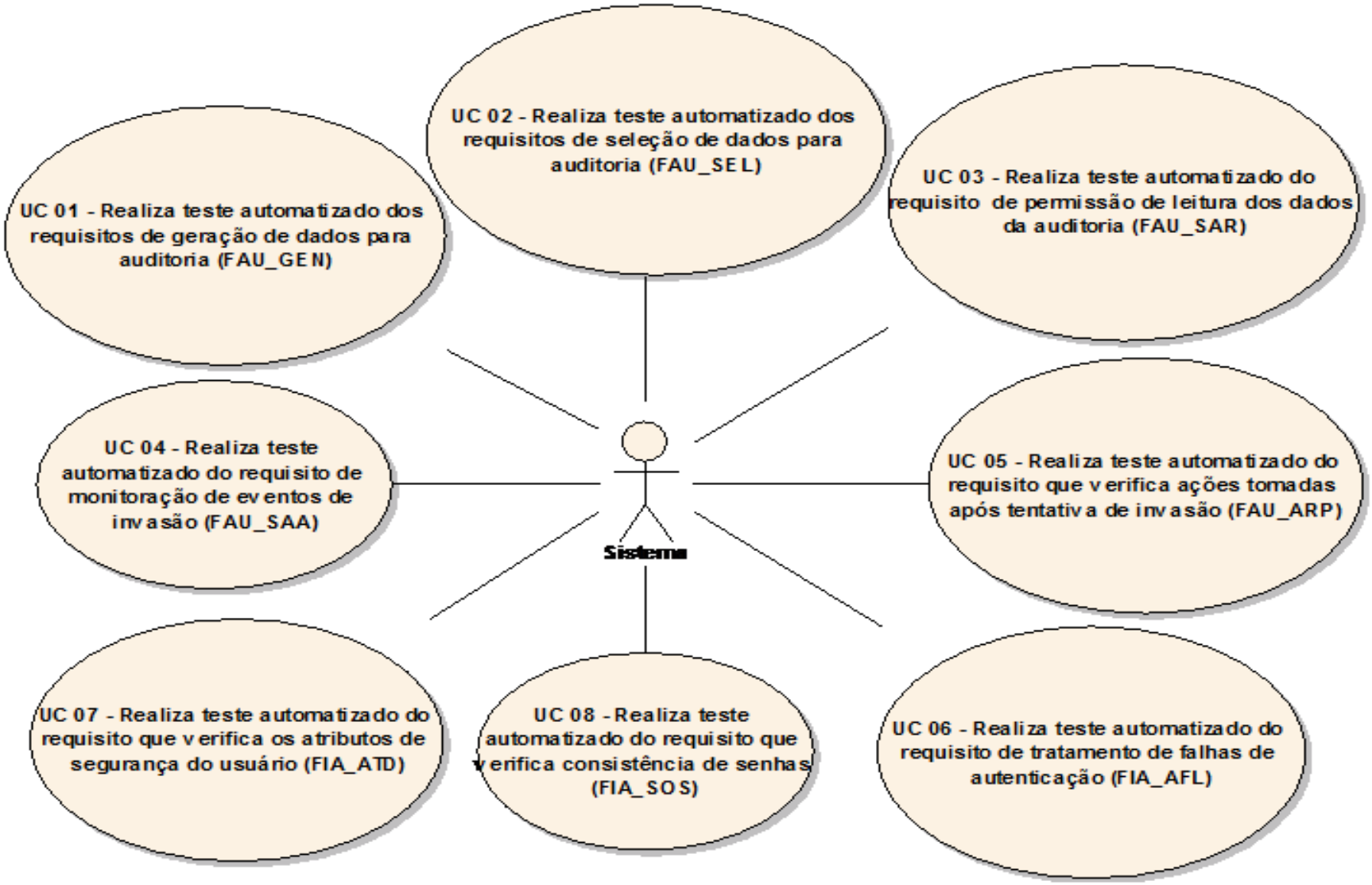

UC 01 - Realiza teste automatizado dos requisitos de geração de dados para auditoria (FAU_GEN)
UC 02 - Realiza teste automatizado dos requisitos de seleção de dados para auditoria (FAU_SEL)
UC 03 - Realiza teste automatizado do requisito de permissão de leitura dos dados da auditoria (FAU_SAR)
UC 04 - Realiza teste automatizado do requisito de monitoração de eventos de invasão (FAU_SAA)
Sistema
UC 05 - Realiza teste automatizado do requisito que verifica ações tomadas após tentativa de invasão (FAU_ARP)
UC 07 - Realiza teste automatizado do requisito que verifica os atributos de segurança do usuário (FIA_ATD)
UC 08 - Realiza teste automatizado do requisito que verifica consistência de senhas (FIA_SOS)
UC 06 - Realiza teste automatizado do requisito de tratamento de falhas de autenticação (FIA_AFL)

Diagrama de classe FAU_ARP

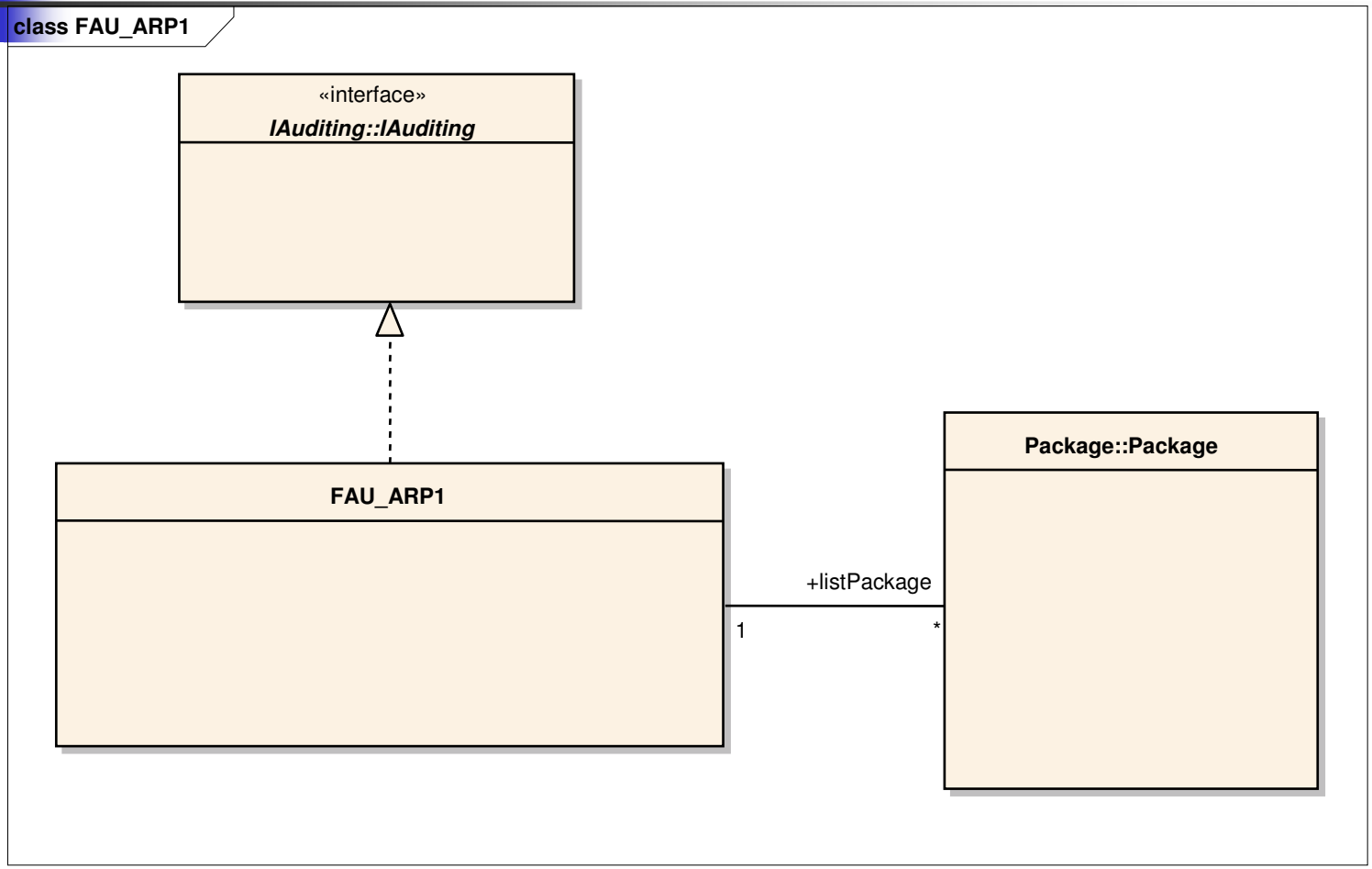


Diagrama de classe FAU_GEN

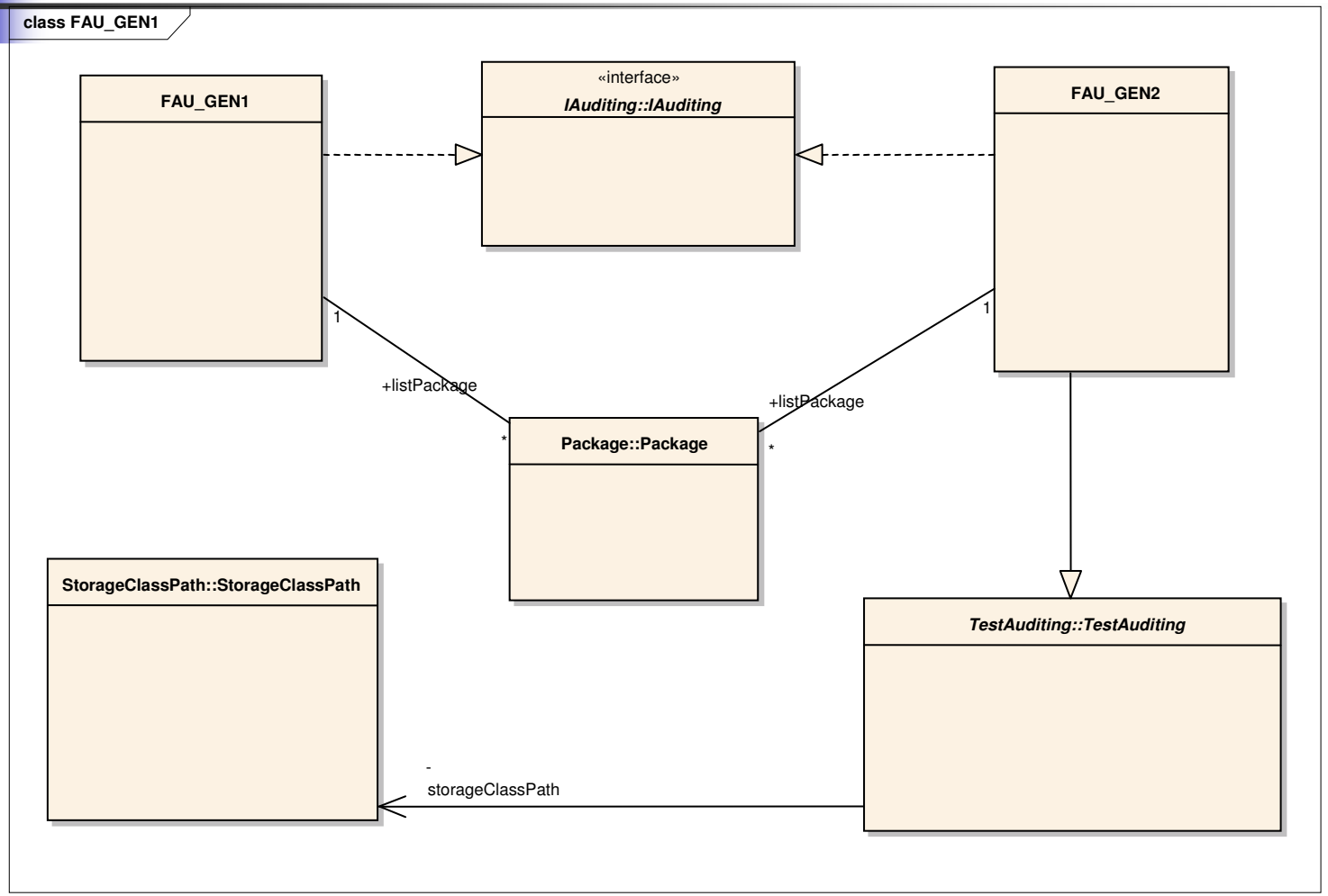


Diagrama de classe FAU_SAA

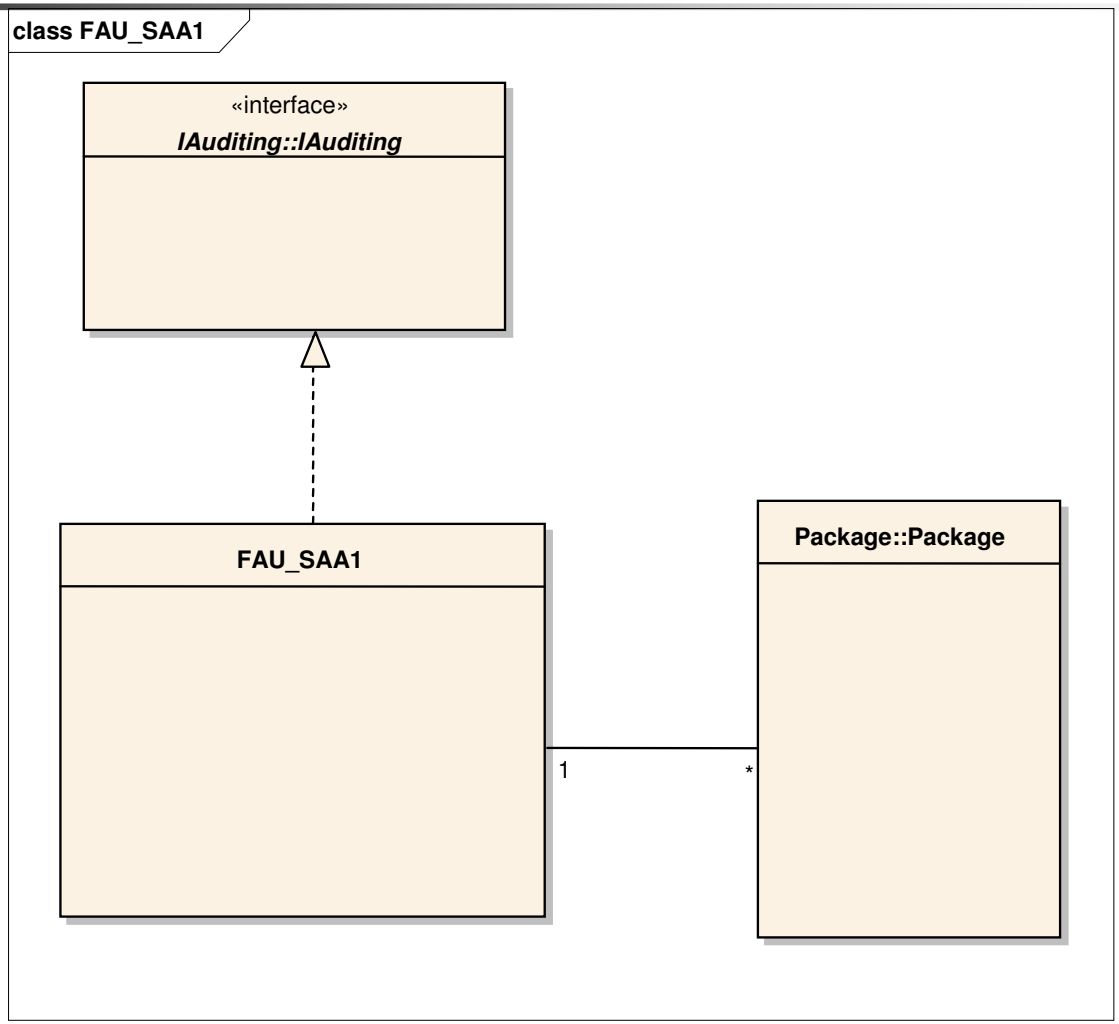


Diagrama de classe FAU_SAR

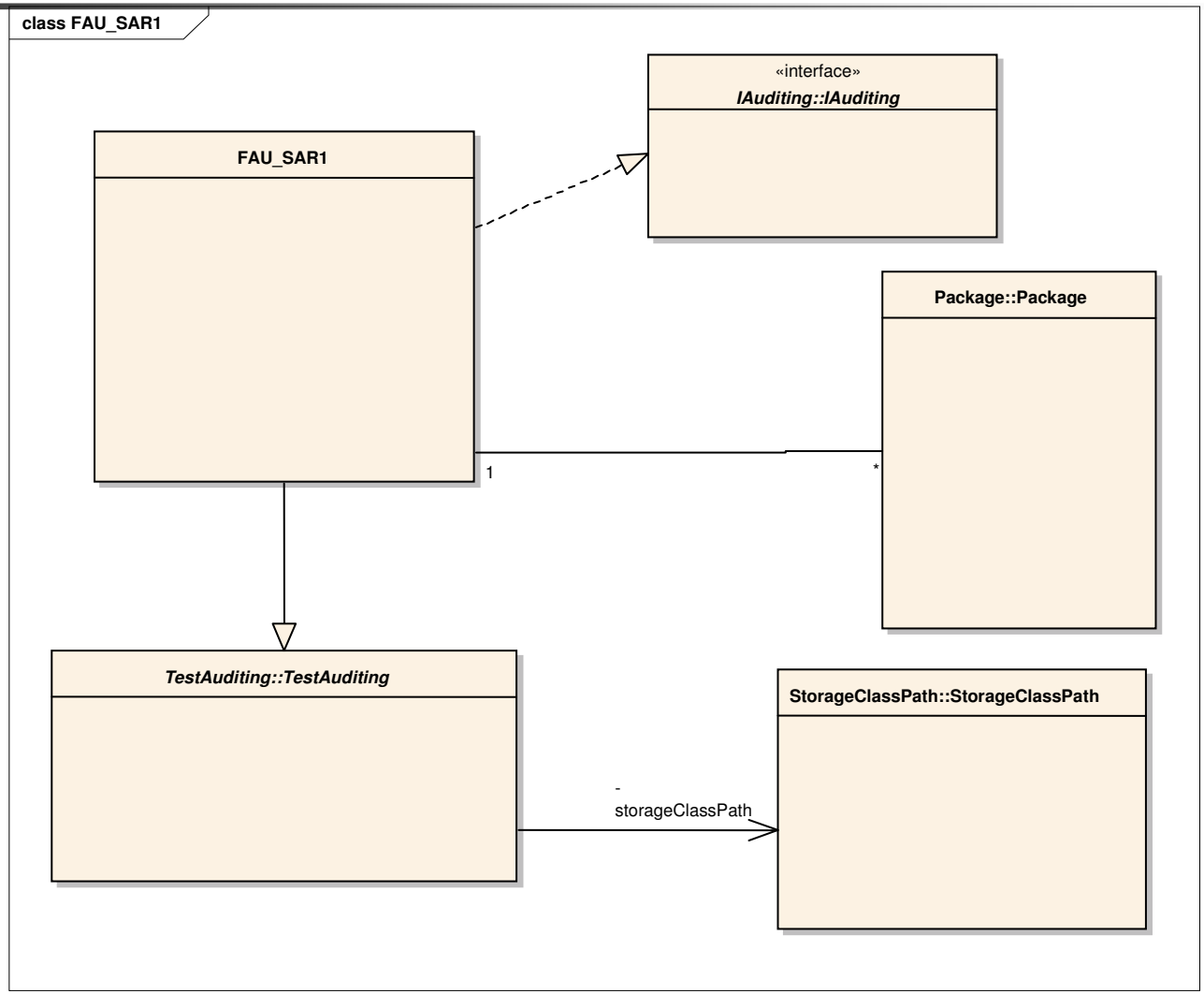


Diagrama de classe FAU_SEL

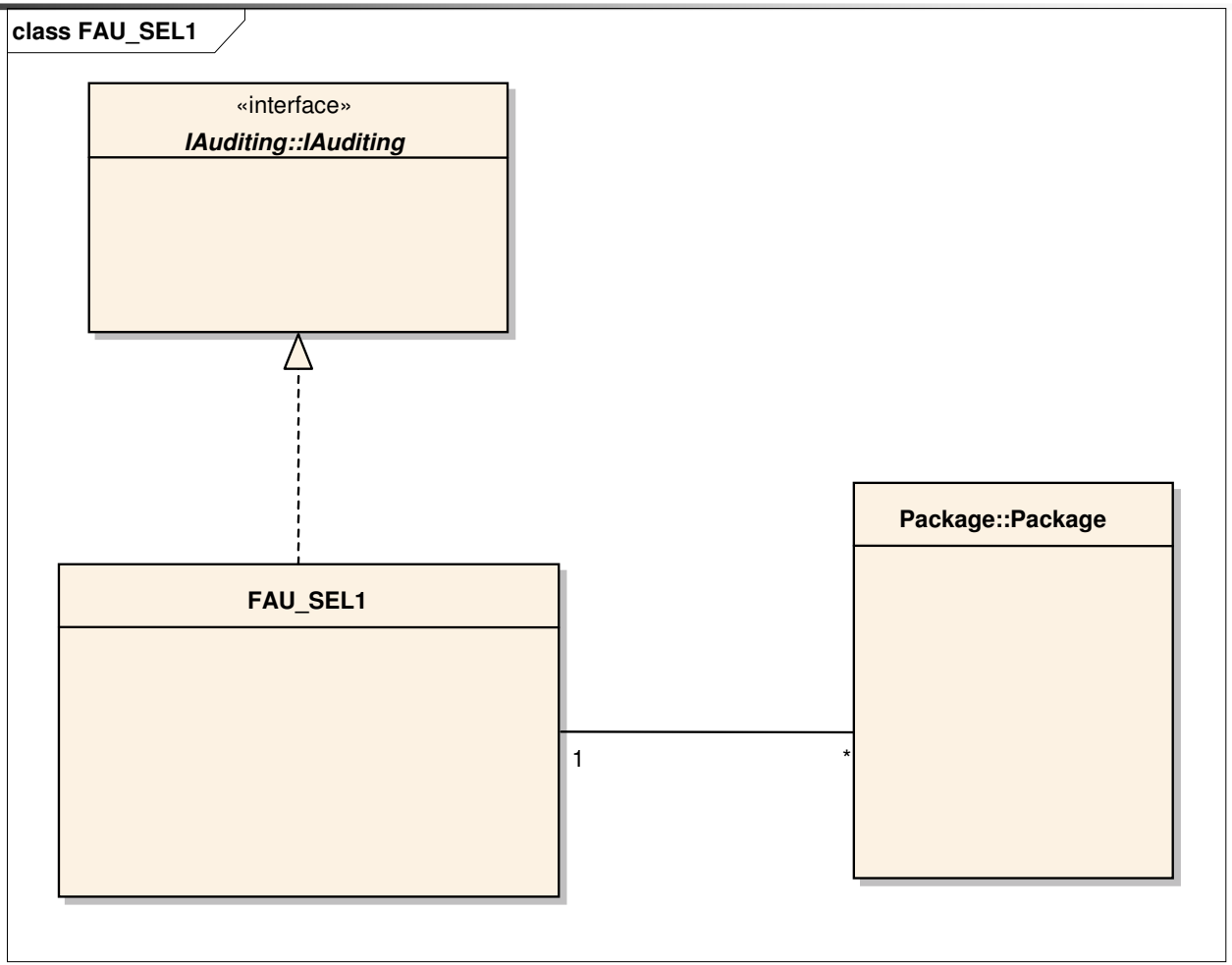


Diagrama de classe FIA_AFL

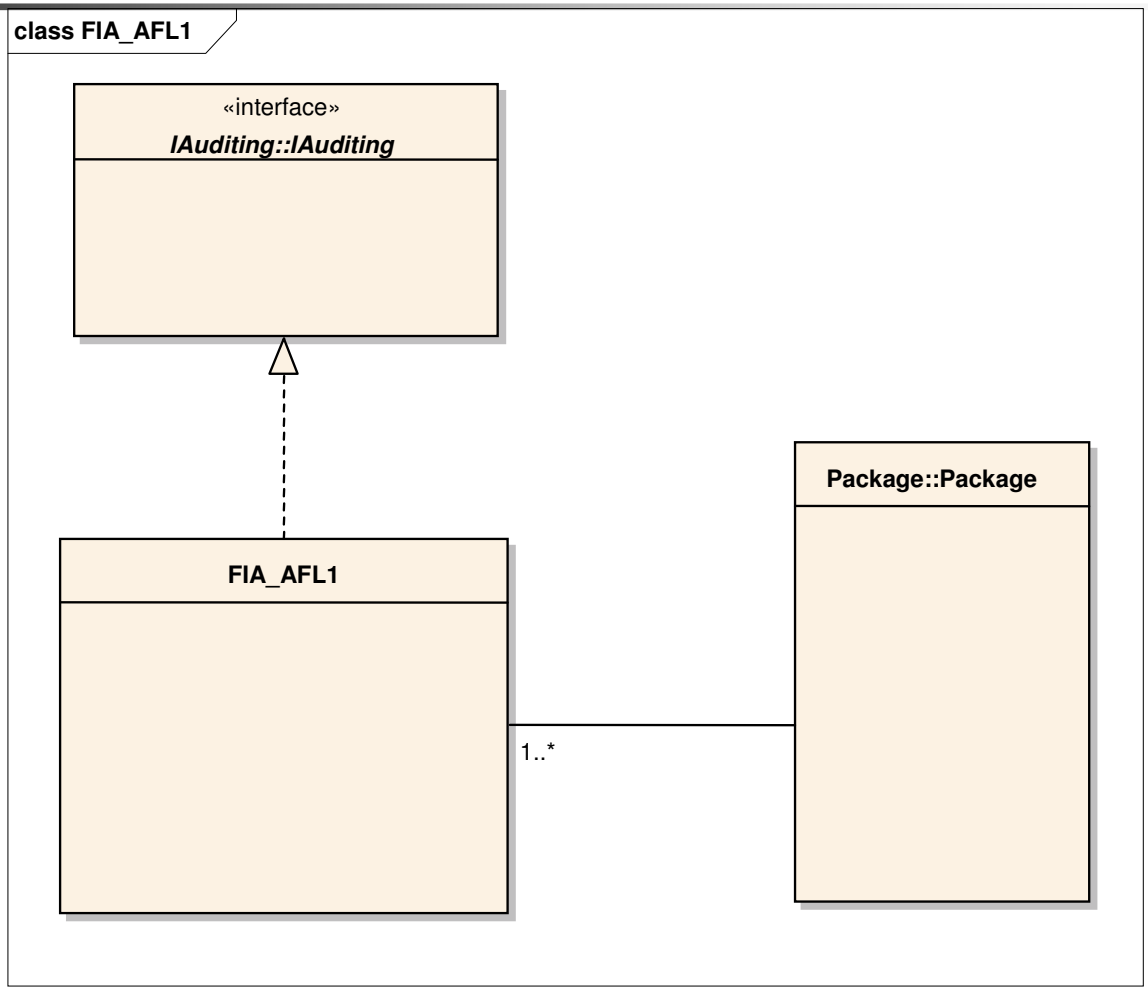


Diagrama de classe FIA_ATD

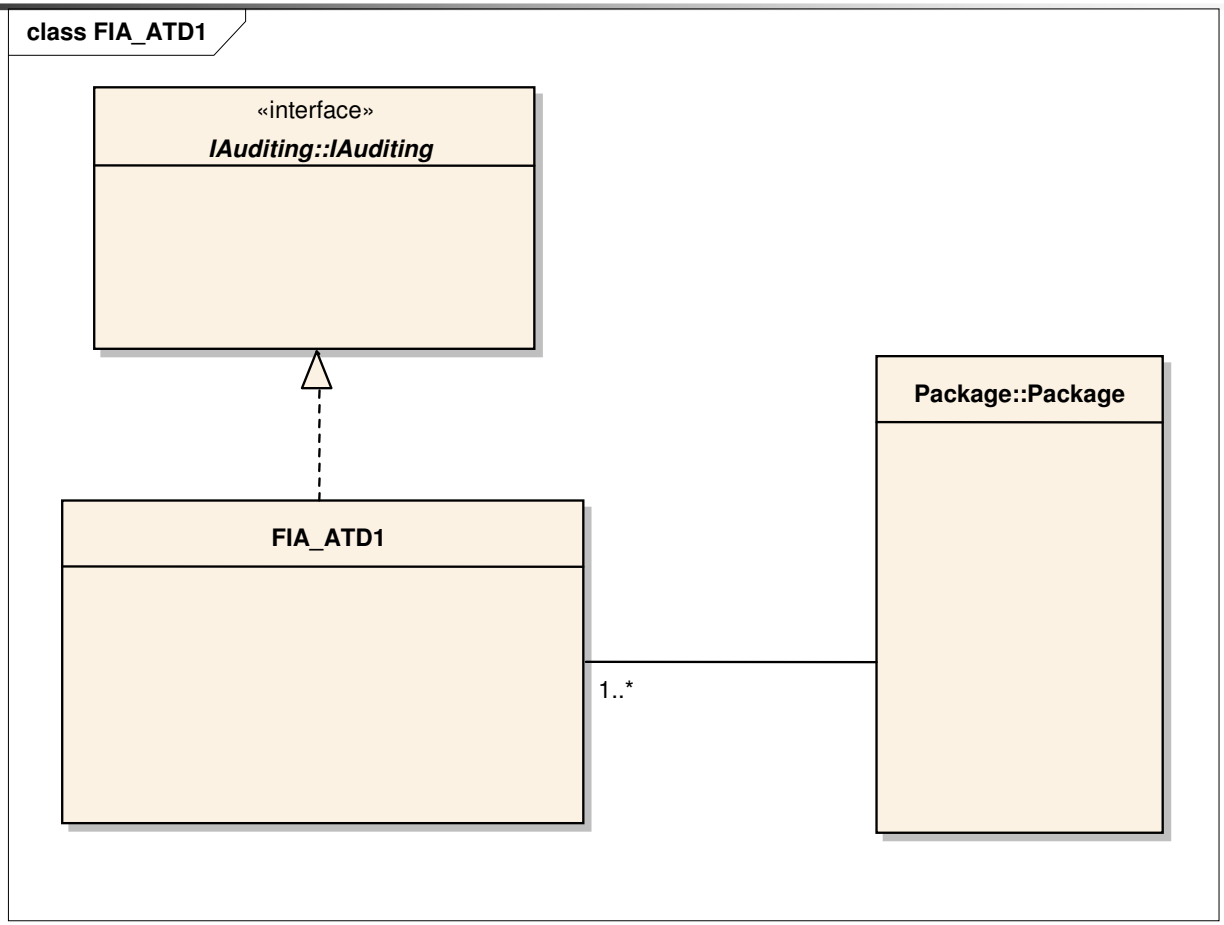
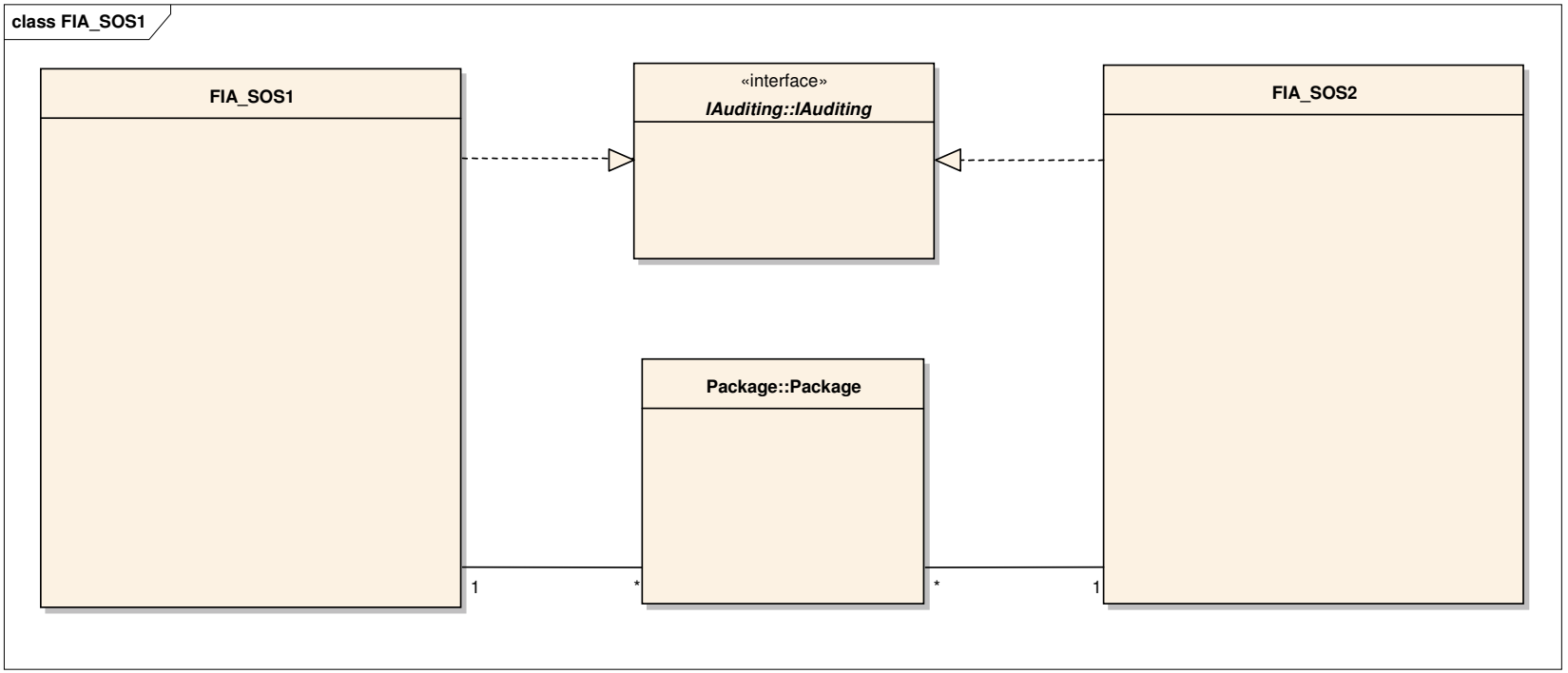
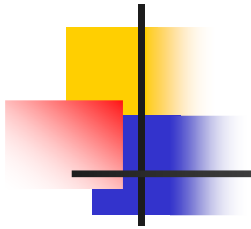


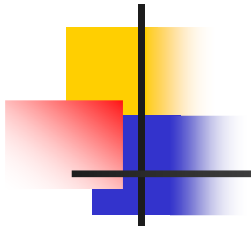
Diagrama de classe FIA_SOS





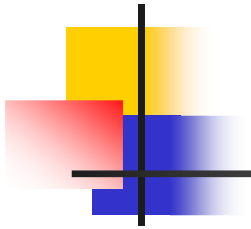
Implementação

- Desenvolvido sistema básico de autenticação de usuário para realização dos testes
- O sistema a ser analisado deve implementar as interfaces disponibilizadas



Biblioteca Dom4J

- Utilizada para realização da leitura do arquivo XML com as configurações iniciais do sistema



Java Reflection

- API utilizada para realizar a análise do sistema em tempo de execução

Código teste FAU_ARP

```
for (int i = 0; i < searchClass.getCLassFound().size(); i++) {
    try {
        System.out.println(package1.getPackageName()
                + "."
                + searchClass.getCLassFound().get(i).getName()
                        .replace(".java", "").toString());
        classVerify = classAnalizer(Class.forName(package1
                .getPackageName()
                + "."
                + searchClass.getCLassFound().get(i).getName()
                        .replace(".java", "")), searchClass
                .getMethod(), interfaceSearch, searchInterface,
                systemDirectory);
        if (classVerify)
            resultado.append("Executado automaticamente com Sucesso - Enviado email para o administrador do sistema");
        else
            resultado.append("Executado automaticamente mas sem o resultado esperado");
    } catch (ClassNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Código teste FAU_ARP

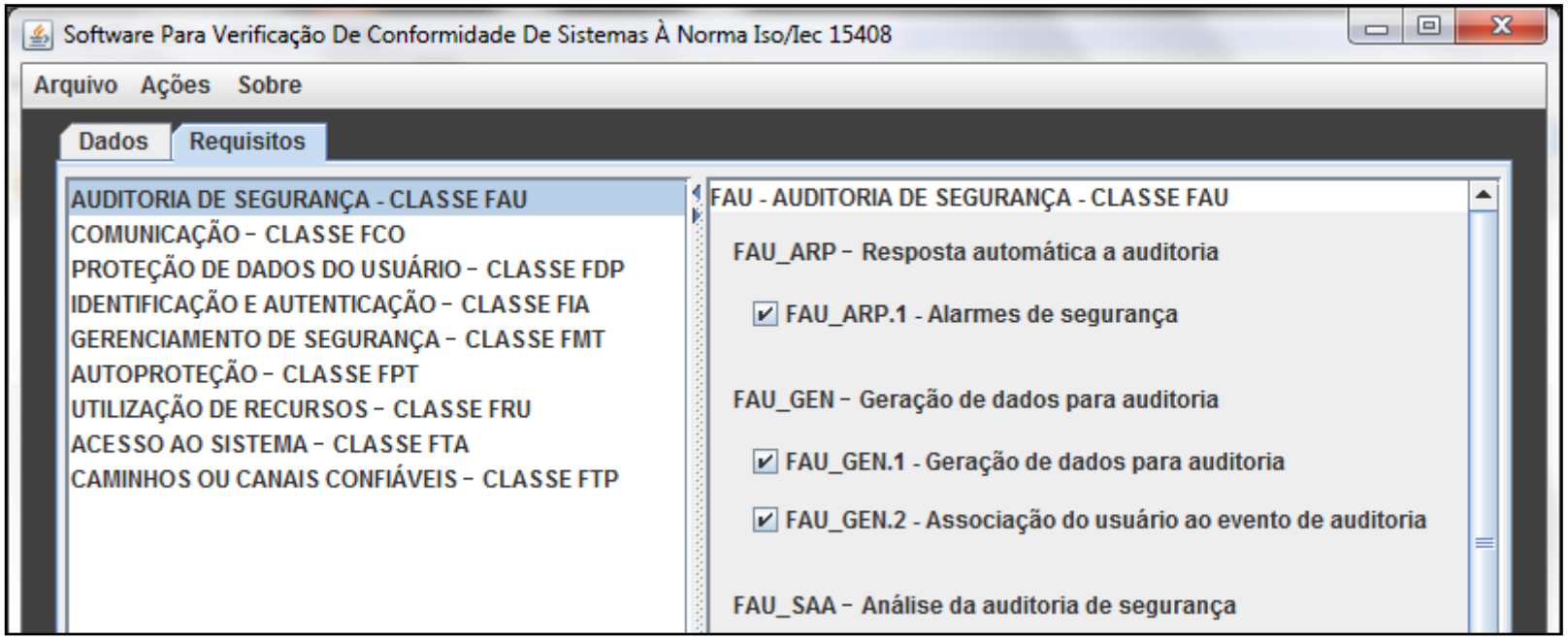
```
for (int i = 0; i < 5; i++) {
    try {
        methodExec = forName.getMethod(methodSearch, new Class[] {
                String.class, char[].class });
        Object c = forName.newInstance();
        methodExec.invoke(c, user, pass.toCharArray());
        System.out.println(Thread.currentThread().getStackTrace()[i]
                .getMethodName());
```

Código teste FAU_ARP

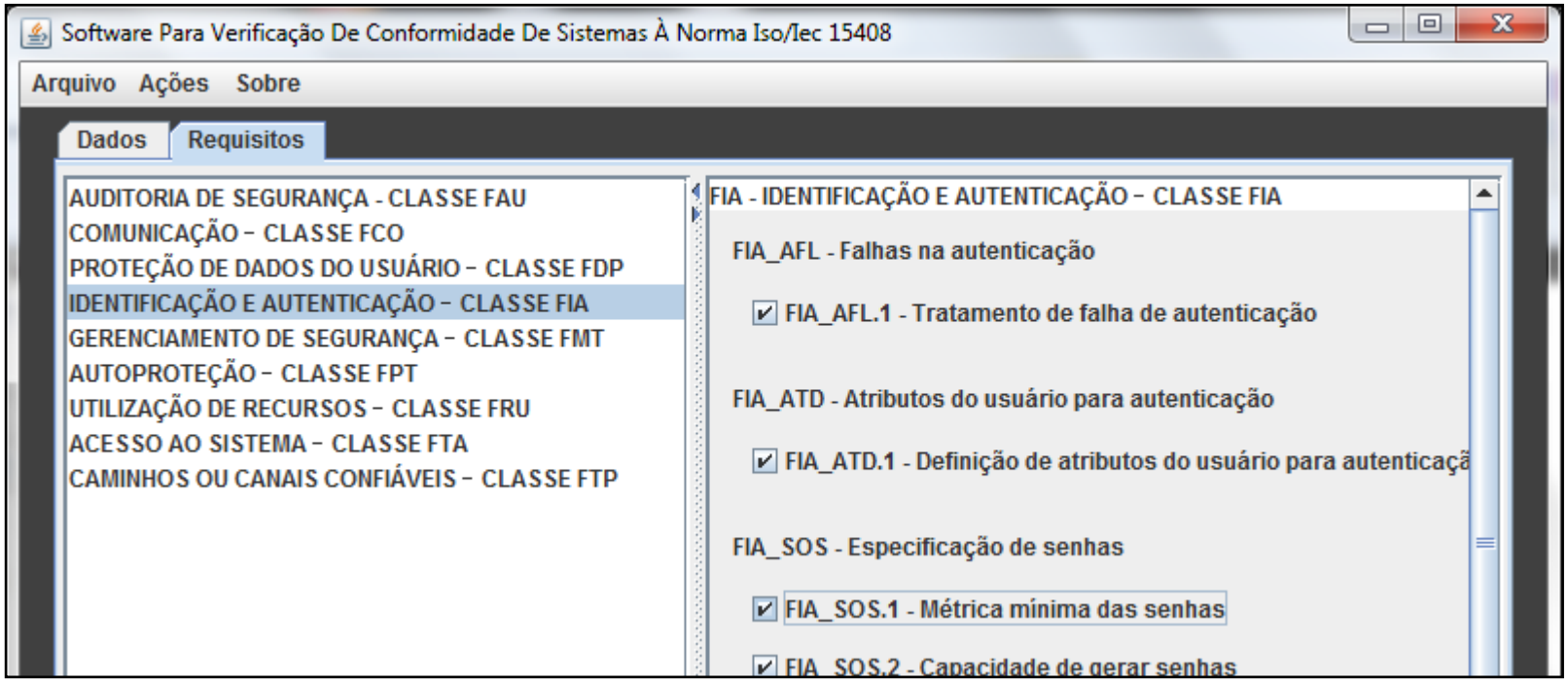
```
for (int k = 0; k < searchInterface.getCLassFound().size(); k++) {
    try {
        Class<?> interfaces[] = forName.getInterfaces();
        for (Class class1 : interfaces) {
            if (class1.getSimpleName()
                    .equalsIgnoreCase(interfaceSearch)) {
                implementsInterface = true;
                Method methods[] = class1.getMethods();
                for (Method method : methods) {
                    newMethod = forName.getMethod(method.getName(),
                            new Class[] {});
                    Object c1 = forName.newInstance();
                    Object inv = newMethod.invoke(c1, null);
                    if (inv.toString().equals("true")) {
                        ret = true;
                    } else
                        ret = false;
                    setOk(ret);
                }
            }
        }
```

OPERACIONALIDADE

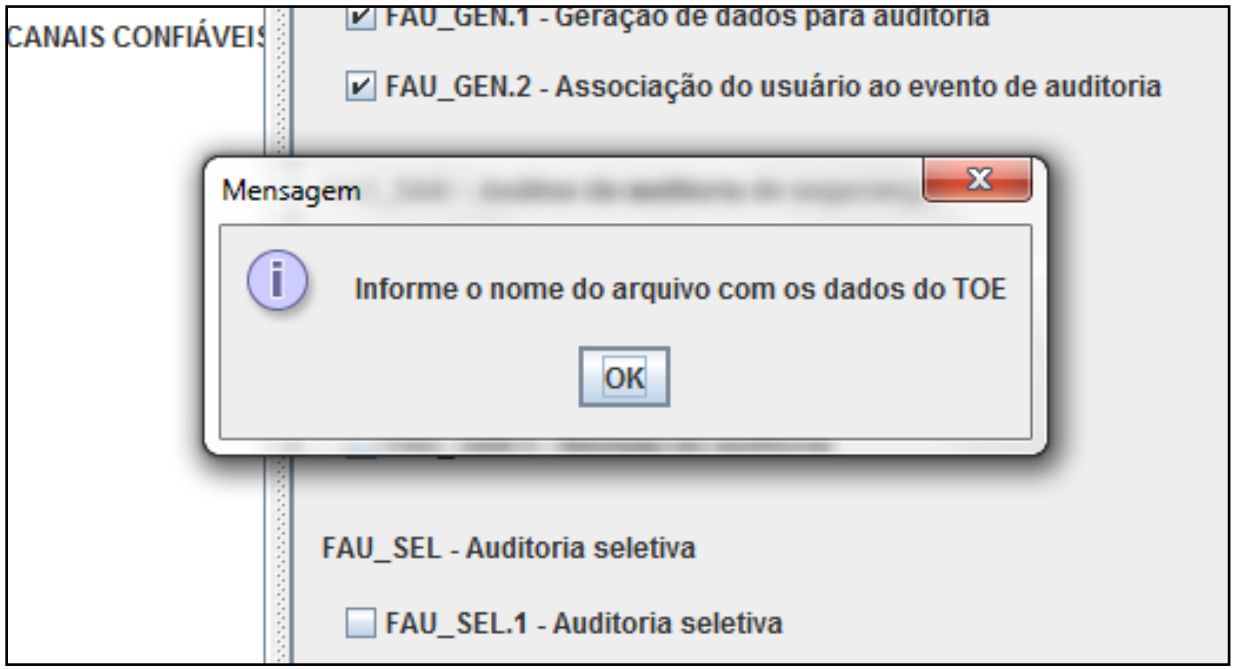
Selecionando os testes



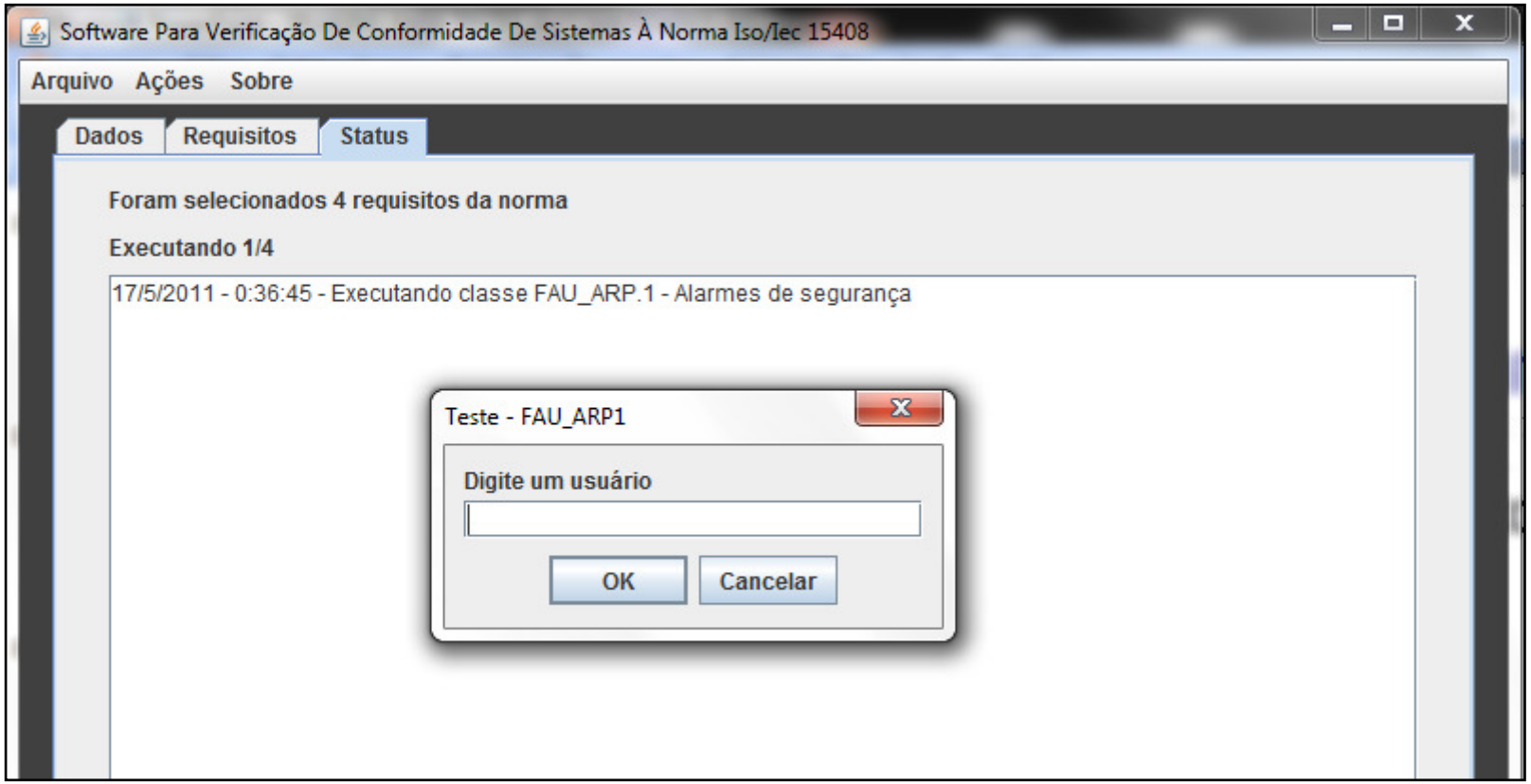
Selecionando os testes



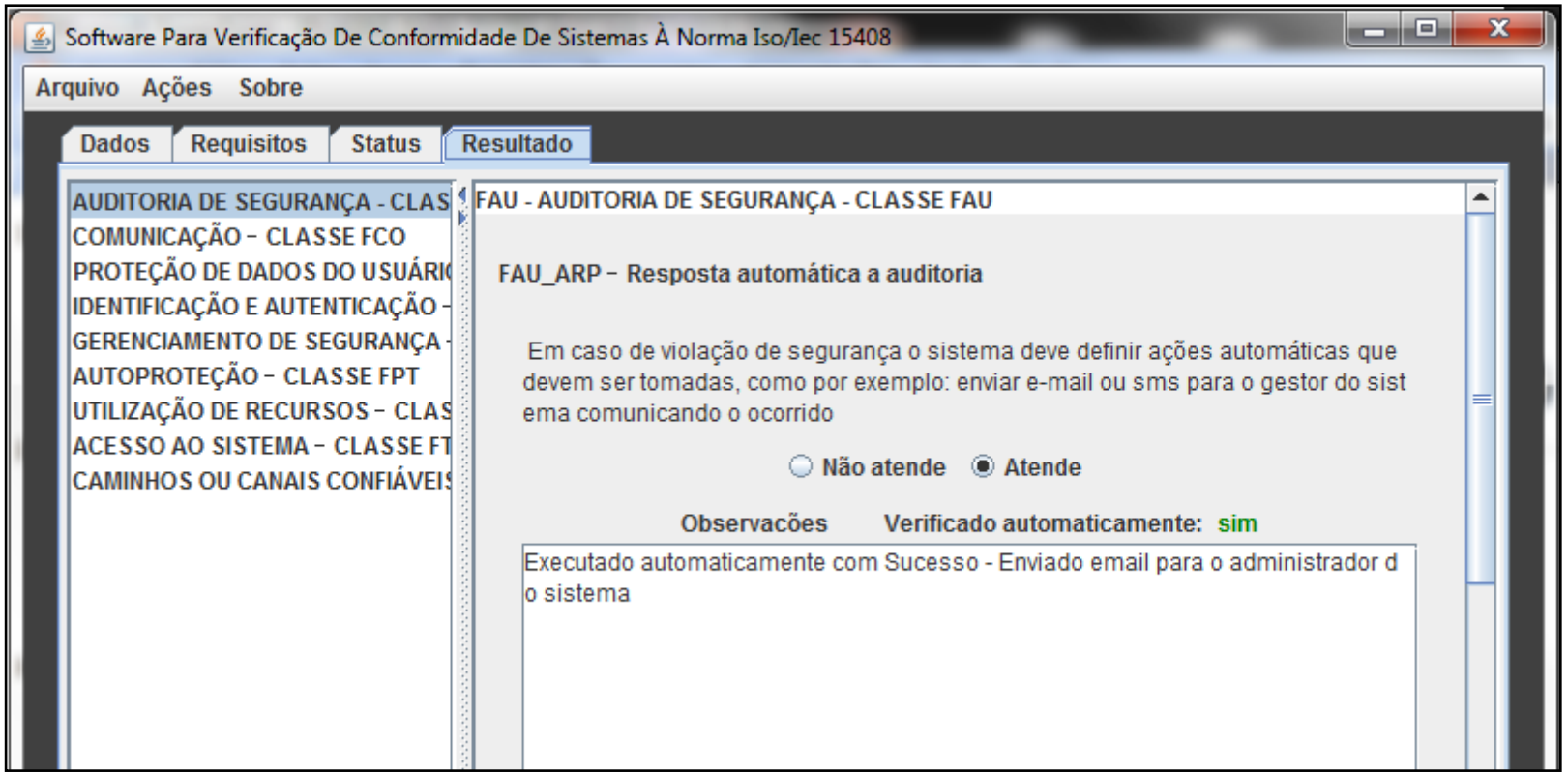
Solicitação do arquivo



Execução dos testes



Resultado dos testes



Relatório gerado

FAU - AUDITORIA DE SEGURANÇA - CLASSE FAU

FAU_ARP Resposta automática a auditoria

FAU_ARP.1 - Alarmes de segurança

O requisito está de acordo com a norma

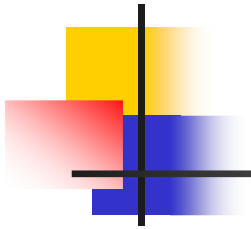
Executado automaticamente com Sucesso - Enviado email para o administrador do sistema

Resultados e discussão

- Dificuldade de realizar os testes de forma genérica
- Poucos softwares de testes verificam este padrão
- Verificação automatizada

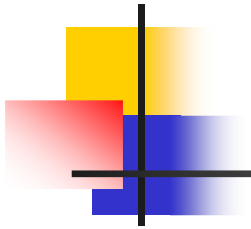
Resultados e discussão

Requisito	SCAN-CC anterior			SCAN-CC atual		
	Atende	Atende parcialmente	Não Atende	Atende	Atende parcialmente	Não Atende
FAU_ARP			X	X		
FAU_GEN		X		X		
FAU_SAA			X	X		
FAU_SAR			X	X		
FAU_SEL			X	X		
FIA_AFL			X	X		
FIA_SOS			X	X		
FIA_ATD				X		
FIA_UAU	X			X		



Conclusão

- Objetivos do trabalho
- Desenvolvimento de um software seguro
- Agilidade no desenvolvimento
- Validação da segurança
- Segurança



Extensões

- Implementar testes automatizados de outras classes
- adaptar a ferramenta para sua utilização via web;
- implementação de um WebService para dar possibilidade a ferramenta para verificação de sistemas não desenvolvidos em Java.