UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

MODELO DE AVALIAÇÃO AUTOMATIZADA DE REQUISITOS DE SEGURANÇA EM SISTEMAS CONFORME A NORMA ISO/IEC 15.408

EDUARDO PERSUHN

BLUMENAU
2015

2015/1-12

EDUARDO PERSUHN

MODELO DE AVALIAÇÃO AUTOMATIZADA DE REQUISITOS DE SEGURANÇA EM SISTEMAS CONFORME A NORMA ISO/IEC 15.408

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Paulo Fernando da Silva, Mestre - Orientador

BLUMENAU
2015

2015/1-12

MODELO DE AVALIAÇÃO AUTOMATIZADA DE REQUISITOS DE SEGURANÇA EM SISTEMAS CONFORME A NORMA ISO/IEC 15.408

Por

EDUARDO PERSUHN

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Paulo Fernando da Silva, Mestre – Orientador, FURB
Membro:	Prof. Francisco Adell Péricas, Mestre – FURB
Membro:	Prof. Antonio Carlos Tavares, Mestre – FURB

Blumenau, 09 de julho de 2015

Dedico este trabalho aos meus amigos, minha família e principalmente aos meus pais, que, me apoiaram durante o desenvolvimento deste trabalho.

AGRADECIMENTOS

A Deus pela saúde.

À minha família por sempre estar presente incentivando, motivando e apoiando.

Aos meus amigos, por estarem lá, trocando ideias, conversando e contribuindo desta forma para a minha evolução.

Ao meu orientador, professor Paulo Fernando da Silva por acreditar em mim e prestar o apoio necessário para a conclusão deste trabalho.

Se, a princípio, a ideia não é absurda, então não há esperança para ela.

Albert Einstein

RESUMO

Este trabalho apresenta a especificação de um modelo para a avaliação automatizada de requisitos de segurança da norma *International Organization for Standardization* / *International Electrotechnical Commission* (ISO/IEC) 15.408. O objetivo deste modelo é criar uma forma de avaliação de segurança envolvendo o sistema alvo e o software que fará a avaliação. A comunicação entre as partes do modelo foi feita utilizando *web services* para tornar o modelo independente da linguagem utilizada no desenvolvimento do sistema alvo. Além do desenvolvimento do modelo, foi implementado um software para avaliação automatizada de requisitos de segurança e alguns requisitos de segurança da norma para testar o modelo. Com o desenvolvimento deste trabalho, verificou-se que é possível automatizar alguns critérios estabelecidos pela norma.

Palavras-chave: ISO/IEC 15.408. *Web services*. Segurança da informação.

ABSTRACT

This work presents the specification of a model for the automated evaluation of safety requirements of the standard International Organization for Standardization / International Electrotechnical Commission (ISO/IEC) 15.408. The purpose of this model is to create a form of safety assessment involving the target system and the software that will make the evaluation. The communication between the parts of the model was done using web services to make the language independent model used to develop the target system. In addition to model development, was implemented a software for automated evaluation of safety requirements and some security requirements of the standard to test the model. With the development of this work, it was found that it is possible to automate some criteria set by the standard.

Key-words: ISO/IEC 15.408. Web services. Information security.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API – *Application Programming Interface*

CC – *Common Criteria*

DOM – *Document Object Model*

EAL – *Evaluation Assurance Level*

HTTP - *Hypertext Transfer Protocol*

IDE – *Integrated Development Environment*

IEC – *International Electrotechnical Commission*

ISO – *International Organization for Standardization*

PDF – *Portable Document Format*

SOAP – *Simple Object Access Protocol*

TI – Tecnologia da Informação

UML – *Unified Modeling Language*

W3C – *World Wide Web Consortium*

WSD – *Web Service Description*

WSDL – *Web Services Description Language*

XML – *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Na década de 60 surgiram os sistemas de computadores, com o objetivo de auxiliar ou até substituir processos manuais de gerência de informações e processamento de dados corporativos. Desde a época ocorreram avanços tecnológicos tanto em relação aos sistemas computadorizados quanto aos microcomputadores nos quais são executados. Diante desta evolução, as grandes organizações estão cada vez mais dependentes dos auxílios providos pelos sistemas computadorizados (DIAS, 2000, p.2). Segundo Howard e LeBlanc (2005, p.33), "à medida que cresce a importância da Internet, os aplicativos estão se tornando altamente interconectados." A partir deste momento a internet pôde ser explorada de forma maliciosa, ou seja, uma pessoa mal intencionada poderia explorar a interconexão dos sistemas em seu beneficio. Este ato é chamado de ataque. (HOWARD; LEBLANC, 2005, p.33).

A partir desta ideia tem-se a premissa de que todo dispositivo interconectado está exposto ao risco de sofrer ataques e que além da preocupação com a segurança visando o usuário do sistema, devem-se levar em conta outros meios que podem explorar a vulnerabilidade.

Diante desta necessidade de segurança para sistemas computadorizados, a adoção de critérios para realizar a avaliação de segurança é imprescindível. Existe uma norma homologada pela ISO chamada ISO/IEC 15.408, conhecida também com *Common Criteria* (CC), que é uma referência para a avaliação da segurança de Tecnologia da Informação (TI). Segundo Albuquerque e Ribeiro (2002, p. 7), "o objetivo da norma é fornecer um conjunto de critérios fixos que permitem especificar a segurança de uma aplicação de forma não ambígua a partir de características do ambiente da aplicação, e definir formas de garantir a segurança da aplicação para o cliente final".

No CC foram definidos sete níveis de garantia de segurança denominados *Evaluation Assurance Level* (EAL). Foi estabelecido que os níveis EAL-5 ao EAL-7 são inviáveis devido à rigidez de seus requisitos. Portanto os níveis reconhecidos pela ISO são os EAL-1 ao EAL-4 (ALBUQUERQUE; RIBEIRO, 2002). Mesmo com a adoção de critérios, a tarefa de auditar sistemas torna-se cada vez mais difícil devido aos avanços tecnológicos e ao ambiente heterogêneo e complexo criado pela área de informática (DIAS, 2000, p.2-3).

Diante do exposto, percebe-se a necessidade da criação de um modelo de avaliação automatizada de requisitos de segurança em sistemas, com base na norma ISO/IEC 15.408. Este modelo deve ser seguido durante a etapa de desenvolvimento do sistema alvo, ou seja, deve-se ter em mente que aquele sistema será construído de forma que se possa realizar a

avaliação dele em conformidade com a norma. O modelo utiliza uma tecnologia que disponibiliza funções através da internet que são usadas como serviços, chamada de *web services*, onde o sistema alvo deve chamar os serviços que implementam os requisitos da norma relevantes para o tipo de segurança exigida. Este modelo é necessário para tornar a avaliação da segurança homogênea e independente do sistema alvo e da implementação dos requisitos. Desta forma não é necessário refazer a implementação da lógica de avaliação dos requisitos da norma para cada sistema.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é propor um modelo para a avaliação automatizada de requisitos de segurança em sistemas utilizando uma interface baseada em *web services*, para a verificação da segurança em conformidade com a norma ISO/IEC 15.408.

Os objetivos específicos do trabalho são:

a) propor um modelo para que o software que fará a avaliação automatizada da segurança não seja dependente do sistema alvo;
b) criar um software que fará a avaliação automatizada da segurança em sistemas;
c) analisar a segurança de sistemas através do software de avaliação automatizada de segurança em sistemas desenvolvidos, para verificar o funcionamento dos requisitos automatizados.

1.2 ESTRUTURA DO TRABALHO

O trabalho está dividido em quatro capítulos. No capítulo um foi apresentado o contexto, o objetivo e alguns conceitos relacionados ao tema do trabalho. No capítulo dois é apresentada a fundamentação teórica, que apresenta os principais aspectos sobre segurança da informação, *web services* e da norma ISO/IEC 15.408. O capítulo três trata do desenvolvimento do modelo, do software para avaliação de segurança e da implementação de alguns requisitos da norma. Por fim, o capítulo quatro tem por objetivo expor as conclusões e resultados do trabalho assim como sugestões para extensões do mesmo.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo apresentar os principais conceitos relacionados ao trabalho desenvolvido. Primeiramente é apresentada uma introdução dos aspectos envolvidos na segurança da informação. São eles: segurança da informação, segurança no desenvolvimento de software, norma ISO/IEC 15.408 e auditoria de segurança da informação. Em um segundo momento é apresentado *web service* que é a principal tecnologia utilizada no desenvolvimento do trabalho. Por fim, são apresentados os trabalhos correlatos e suas características.

2.1 SEGURANÇA DA INFORMAÇÃO

Atualmente organizações utilizam softwares que trabalham com informações importantes como dados pessoais, senhas ou ainda informações confidenciais. Os sistemas para as mais variadas finalidades têm conectividade com a internet que acaba expondo ao risco de sofrerem ataques a fim de obter proveito destas informações. Exemplos de sistemas como comércios eletrônicos, bancos, pagamento de taxas, entre outros devem ter sua segurança garantida, visando reduzir a sua vulnerabilidade (ALLEN et al., 2008, p.2).

A partir desta ideia, pode-se afirmar que a segurança de sistemas não é algo trivial, porém existem diferentes domínios de aplicações às quais o nível de segurança pode variar de acordo com a importância dos dados manipulados pelos softwares.

Os três aspectos básicos para segurança de aplicações são (OLIVERA, 2001, p.9-10):

a) confidencialidade: garantir o ocultamente de dados importantes;
b) disponibilidade: garantir que estejam disponíveis informações e serviços quando solicitados;
c) integridade: garantir que as informações permaneçam intactas.

Além dos aspectos básicos existem diversos outros, como (ALBUQUERQUE; RIBEIRO, 2002, p.2-3):

a) autenticação: garantir que um usuário, sistema ou informação é mesmo o que alega ser;
b) não repúdio: provar que um usuário executou determinada ação no sistema;
c) legalidade: aderência de um sistema à legislação;
d) privacidade: capacidade do sistema de manter incógnito um usuário, impossibilitando a ligação direta da identidade do usuário com ações por ele realizadas;

e) auditoria: garantir que tudo que foi realizado pelo usuário possa ser analisado, detectando fraudes ou tentativas de ataque.

A segurança em softwares possui várias áreas de vulnerabilidade, como *bugs* ao codificar sistemas e dados inconsistentes, intrusos maliciosos ou ainda códigos maliciosos que obtém acesso não autorizado e realizam ataques se aproveitando dos erros de segurança do software. Com o uso da internet ocorre o aumento da complexidade de mecanismos de segurança (ALLEN et al., 2008, p.3).

2.2 SEGURANÇA NO DESENVOLVIMENTO DE SOFTWARE

Existem três preocupações quanto à segurança no desenvolvimento de software. São elas: segurança do ambiente de desenvolvimento, da aplicação desenvolvida e garantia de segurança (ALBUQUERQUE; RIBEIRO, 2002, p.5).

> Essas três preocupações são complementares. Não há como garantir a segurança sem uma aplicação segura. Também não há como obter uma aplicação segura sem um ambiente de desenvolvimento seguro. Estamos sempre lidando com essas três preocupações em conjunto. (ALBUQUERQUE; RIBEIRO, 2002, p.5).

2.2.1 Segurança do ambiente de desenvolvimento

Segurança no ambiente de desenvolvimento é uma das maiores preocupações das empresas, pois é impossível obter um software seguro em um ambiente inseguro. Dentre as características de ambiente seguro estão: espaço restrito, separação entre os ambientes de desenvolvimento, testes e construção, gerência de configuração dos fontes, processos de desenvolvimento bem estabelecidos e controlados e equipe de testes capacitada e bem equipada (ALBUQUERQUE; RIBEIRO, 2002, p.5).

Durante o desenvolvimento de software é comum a adoção de processos provindos da área da engenharia de software como o modelo em espiral, iterativo e incremental, entre outros modelos de processo de desenvolvimento de software. Durante o ciclo de desenvolvimento é possível conciliar às rotinas tradicionais dos processos, atividades como: revisão da equipe de segurança; testes do menor privilégio; ou ainda educação da equipe de desenvolvimento. Tornando assim o sistema em desenvolvimento mais seguro a cada etapa do processo (HOWARD; LEBLANC, 2005, p.52-53).

2.2.2 Segurança da aplicação desenvolvida

O ambiente de desenvolvimento é formado por pessoas incumbidas de desenvolver softwares. No entanto, são por meio das pessoas pertencentes às equipes de desenvolvimento

que falhas de segurança nas aplicações são cometidas. Howard e LeBlanc (2005, p.51, adaptado à ABNT) destacam algumas razões para os equívocos de segurança cometidos por programadores:

a) a segurança é entediante;
b) a segurança costuma ser vista como um desmotivador de funcionalidades, como algo que atrapalha;
c) a segurança é difícil de medir;
d) normalmente, a segurança não é a principal habilidade ou interesse dos projetistas e desenvolvedores que criam o produto;
e) a segurança não significa criar algo novo e animador.

Existe uma série de dicas de programação que automaticamente eliminam muitos dos erros e falhas de segurança em aplicações. Alguns exemplos incluem (ALBUQUERQUE; RIBEIRO, 2002, p.6):

a) funções intrinsecamente seguras;
b) sempre verificar os códigos de erro retornados por uma função;
c) atentar sempre o tamanho dos *buffers* e *arrays* do sistema;
d) documentar o código.

2.2.3 Garantia de segurança

Garantia de segurança tem um propósito diferente da segurança da aplicação desenvolvida. Para fornecer a garantia de segurança deve-se disponibilizar uma forma de comprovar ao cliente da aplicação que o software é seguro de fato. Uma forma são testes que evidenciam a segurança do sistema analisado. A forma como a ISO/IEC 15.408 trata a garantia é:

a) especificar a segurança de forma clara e objetiva;
b) construir conforme essa especificação;
c) testar para verificar o atendimento da especificação original.

São estas condições e um ambiente seguro que garantem a segurança. (ALBUQUERQUE; RIBEIRO, 2002, p.6).

2.3 NORMA ISO/IEC 15.408

De acordo com Wagner (2011, p.20), "a norma ISO/IEC 15408, também chamada de *Common Criteria*, é um conjunto de critérios que permite a especificação da segurança de uma aplicação, baseado nas características do ambiente de desenvolvimento".

O *Common Criteria* tem como base o *Trusted Computer Sucurity Evaluation Criteria* (TCSEC) que foi o primeiro padrão para avaliação de segurança desenvolvido em 1980. O TCSEC também era conhecido como *Orange Book* e era um padrão bem aceito, porém continha problemas devido aos níveis estáticos de avaliação de segurança. Além deste, outras tentativas de padrões surgiram, como o ITSEC (1991), CTCPEC (1993) e o *Federal Criteria* (FC) (1993) (ALBUQUERQUE; RIBEIRO, 2002 p.7-8).

A ISO/IEC 15.408 disponibiliza um conjunto de requisitos para a avaliação de segurança de TI que podem ser aplicados durante a avaliação, como: hardware, firmware ou software. A norma é útil não só para a avaliação do produto final, mas é um guia que disponibiliza regras que auxiliam no desenvolvimento de sistemas seguros (ISO, 2009).

> A idéia do Common Criteria é de que posso desenvolver o sistema, seguindo a norma, testá-lo em um laboratório credenciado e obter um selo de certificação: meu sistema atende aos requisitos de segurança. Com isso, eliminamos (ou minimizamos) o risco para o cliente na compra do sistema. (ALBUQUERQUE; RIBEIRO, 2002, p.8).

O CC define ainda alguns termos relacionados à segurança, como (ALBUQUERQUE; RIBEIRO, 2002, p.8):

a) *Target Of Evaluation* (TOE): traduzido como alvo da avaliação. É o sistema em desenvolvimento ou sob avaliação que contém funções que devem ser avaliadas;
b) *Security Target* (ST): traduzido como objetivo ou alvo de segurança. Tendo como base um TOE, indica quais os aspectos de segurança importantes e o motivo;
c) *Protection Profile* (PP): traduzido como perfil de proteção. É um documento semelhante ao ST, porém não está restrito a um TOE específico. O PP cria um perfil para toda uma classe de sistemas.

A norma define sete níveis de garantia de segurança. Os níveis são denominados como *Evaluation Assurance Level* (EAL), que varia do EAL-1 até EAL-7. Os reconhecidos pela ISO são apenas os do nível EAL-1 até EAL-4, pois do EAL-5 ao EAL-7 é considerado, na prática, inviável (ALBUQUERQUE; RIBEIRO, 2002, p.8).

Com base na norma uma família é identificada por seis caracteres alfanuméricos separados pelo caractere '_' (*underscore*). Tem por objetivo designar um conjunto de critérios que tem relação com uma determinada área de avaliação. Um requisito por sua vez é

identificado pelo nome da família seguido de sua numeração. Os requisitos são os critérios citados anteriormente. Eles definem o que é necessário para que a segurança seja avaliada. Analisando o sucesso ou falha dos requisitos é que o auditor dá seu veredito quanto à aprovação do sistema alvo.

Os requisitos de segurança são divididos em famílias que definem critérios baseados no tipo de informação utilizada pelo sistema. São elas (ALBUQUERQUE; RIBEIRO, 2002, p.114-238):

a) FAU: regras para registro e visualização de dados de auditoria:
   - FAU_ARP: resposta automática à auditoria,
   - FAU_GEN: geração de dados para auditoria,
   - FAU_FAA: análise da auditoria de segurança,
   - FAU_SAR: revisão de dados da auditoria,
   - FAU_SEL: auditoria seletiva,
   - FAU_STG: armazenamento da trilha de auditoria;
b) FIA: regras para identificação e autenticação do usuário:
   - FIA_AFL: falhas na autenticação,
   - FIA_ATD: atributos do usuário para autenticação,
   - FIA_SOS: especificação de senhas,
   - FIA_UAU: autenticação do usuário,
   - FIA_UID: identificação do usuário,
   - FIA_USB: ligação do usuário com o sistema;
c) FTA: regras para controle de acesso do usuário:
   - FTA_LSA: limitação do escopo ao sistema,
   - FTA_MCS: limitação de seções simultâneas,
   - FTA_SSL: travamento de seção,
   - FTA_TAB: mensagem de acesso,
   - FTA_TAH: histórico de acesso,
   - FTA_TSE: limitação de acesso ao sistema;
d) FCS: regras para especificação e uso de criptografia:
   - FCS_COP: operação de criptografia,
   - FCS_CKM: gerenciamento de chaves de criptografia;
e) FCO: regras para tratamento de ataques de repúdio:
   - FCO_NRO: não repúdio de origem,
   - FCO_NRR: não repúdio de recebimento;

f) FTP: regras para especificação e uso de canal seguro ou confiável de comunicação entre o sistema e o usuário ou entre sistemas e outros sistemas:
   - FTP_ITC: canal seguro entre funções de segurança,
   - FTP_TRP: caminho confiável;

g) FPT: regras para especificação da proteção de dados do sistema:
   - FPT_AMT: teste da camada subjacente,
   - FPT_ITA: disponibilidade de dados exportados pela aplicação,
   - FPT_ITC: confidencialidade dos dados exportados pela aplicação,
   - FPT_ITI: integridade dos dados exportados pela aplicação,
   - FPT_ITT: proteção na transferência interna de dados,
   - FPT_PHP: proteção física do sistema,
   - FPT_RPL: detecção de repetição,
   - FPT_RVM: monitor de referência,
   - FPT_SSP: protocolo de sincronismo de estado,
   - FPT_STM: registro de tempo,
   - FPT_TDC: consistência de dados entre funções de segurança,
   - FPT_TRC: consistência de dados replicados,
   - FPT_TST: autoteste;

h) FPR: regras para privacidade dos dados do usuário:
   - FPR_ANO: anonimato,
   - FPR_PSE: pseudônimo,
   - FPR_UNL: não-rastreamento,
   - FPR_UNO: invisibilidade;

i) FDP: regras para proteção de dados do usuário do sistema:
   - FDP_ACF: funções de segurança,
   - FDP_ACC: política de controle de acesso,
   - FDP_DAU: autenticação de dados,
   - FDP_ETC: exportação de dados para fora do controle do sistema,
   - FDP_IFC: funções de controle de fluxo de informações,
   - FDP_IFF: política de controle de fluxo de informações,
   - FDP_ITC: importação de fora do controle das funções de segurança da aplicação,
   - FDP_ITT: transferência interna,
   - FDP_RIP: proteção de informação residual,

- FDP_ROL: retorno (*rollback*),
- FDP_UCT: confidencialidade de transferência de dados básica,
- FDP_UIT: proteção de integridade de dados do usuário;

j) FMT: regras para gerenciamento de segurança do sistema:
- FMT_MOF: gerenciamento de funções de segurança,
- FMT_MSA: gerenciamento de atributos de segurança,
- FMT_ MTD: gerenciamento de dados de segurança,
- FMT_SMF: especificação do gerenciamento de funções,
- FMT_SMR: papéis de gerenciamento de segurança;

k) FRU: regras para segurança no uso de recursos do sistema:
- FRU_PRS: prioridade de serviços,
- FRU_RSA: alocação de recursos.

2.4 AUDITORIA DE SEGURANÇA DA INFORMAÇÃO

> A auditoria é uma atividade que engloba o exame das operações, processos, sistemas e responsabilidades gerenciais de uma determinada entidade, com o intuito de verificar sua conformidade com certos objetivos e políticas institucionais, orçamentos, regras, normas ou padrões. A atividade de auditoria pode ser dividida em três fases: planejamento, execução e relatório. (DIAS, 2000, p.8).

A auditoria na área de TI analisa a gestão de recursos, com o foco em aspectos de eficiência, eficácia, economia e efetividade. O auditor tem a liberdade de analisar áreas que julgar necessário, podendo assim julgar o ambiente de informática como um todo ou ainda a organização do departamento de informática (DIAS, 2000, p.12).

Dias (2000, p.13-14) divide a auditoria em tecnologia da informação em três subáreas:

a) auditoria da segurança de informações: geralmente envolve avaliação da política de segurança, controles de acesso lógico, controles de acesso físico, controles ambientais e planos de contingência e continuidade de serviços;
b) auditoria da tecnologia da informação: envolve controles organizacionais, de mudanças, de operação dos sistemas, sobre banco de dados, sobre microcomputadores e sobre ambientes cliente-servidor;
c) auditoria de aplicativos: abrange controles sobre o desenvolvimento de sistemas aplicativos, controles de entrada, processamento e saída de dados, controles sobre conteúdo e funcionamento do aplicativo, com relação à área por ele atendida.

2.5 *WEB SERVICE*

Um *web service* pode ser definido como um software projetado para suportar a interoperabilidade entre máquinas através de uma rede de computadores. Um *web service* tem uma interface descrita numa linguagem compreendida por máquinas (uma linguagem que possa ser computada), denominada *Web Services Description Language* (WSDL). O *Simple Object Access Protocol* (SOAP) é uma forma de descrição de mensagens, utilizada para a comunicação com outros sistemas. As mensagens descritas com base no protocolo SOAP são transmitidas por protocolos da Web como o *Hypertext Transfer Protocol* (HTTP) e usando a *Extensible Markup Language* (XML) (W3C, 2004a).

Para Graham et al. (2002, p.11, tradução nossa) do ponto de vista técnico "[...] um *Web service* nada mais é que uma coleção de uma ou mais operações que são acessíveis pela rede e são descritas como um serviço. [...]". O foco principal da tecnologia é a capacidade de trocar dados e funcionalidades entre sistemas diferentes, usando mensagens que seguem um padrão. Além desta característica a tecnologia permite ainda que novos sistemas possam utilizar os serviços criados (GRAHAM et al., 2002, p.500). Esclarecendo a ideia, pode-se dizer que quando se constrói um *web service*, o mesmo pode ser usado por qualquer aplicação existente que possa suporta-lo. E caso novas aplicações desenvolvidas tenham interesse em utiliza-lo, a utilização pode ser feita sem afetar as aplicações envolvidas e sem a necessidade de reimplementar o serviço. Estas aplicações podem estar sendo executadas em diversas plataformas como: browsers, PCs, dispositivos moveis, entre outros (GRAHAM et al., 2002, p.11).

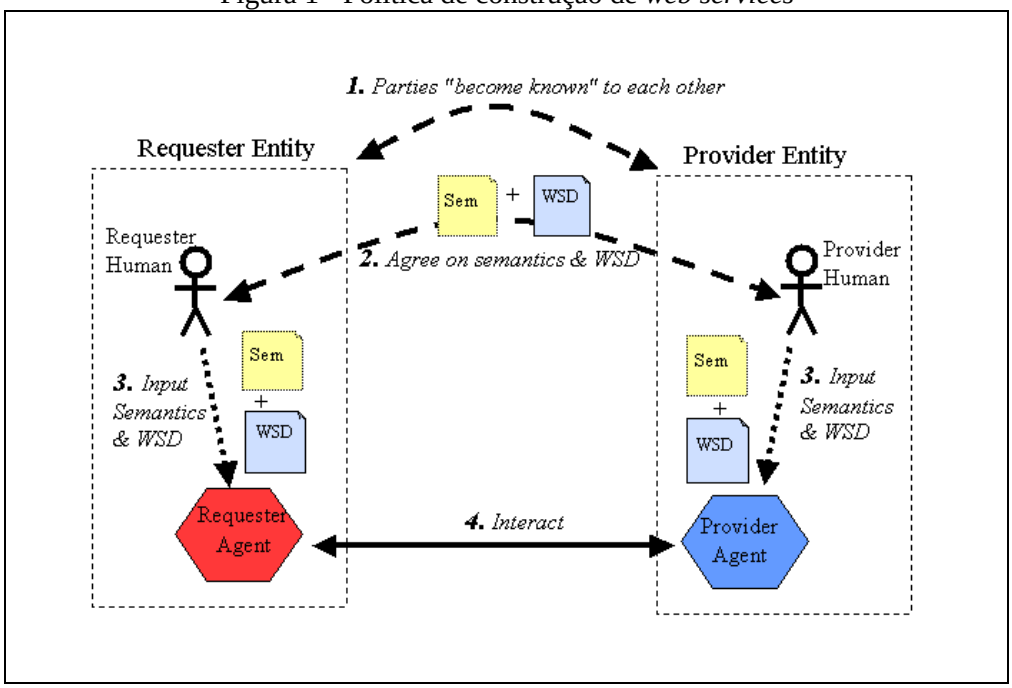
A arquitetura dos *web services* apresenta os seguintes conceitos (W3C, 2004):

a) *web service:* conceito que representa um serviço de forma abstrata;
b) agente: um elemento concreto (não-abstrato) de software ou hardware que recebe e envia mensagens e realiza a função proposta pelo *web service*;
c) provedor: entidade que disponibiliza um serviço e implementa um agente, o provedor pode ser uma pessoa ou uma organização;
d) solicitante: entidade que busca um serviço e implementa um agente que interage com o agente disponibilizado pelo provedor, o solicitante pode ser uma pessoa ou uma organização;
e) descrição do serviço: conhecido como *Web Service Description* (WSD), descreve a forma como será feita a interação entre os dois agentes (Provedor e Solicitante), descrita através da WSDL;

f) semântica: especifica o comportamento do serviço, em outras palavras como o agente solicitante deve utilizar o *web service*. É referido no documento de especificação da arquitetura dos *web services*, disponibilizado pela *World Wide Web Consortium* (W3C) como, o "contrato" entre a entidade provedora e a entidade solicitante.

A Figura 1 demonstra a política de construção de *web services*.

Figura 1 - Política de construção de *web services*



Fonte: W3C (2014b).

As etapas demonstradas na Figura 1 são (W3C, 2004b):

a) etapa 1: a entidade solicitante e entidade provedora tomam conhecimento entre si;
b) etapa 2: as entidades aceitam o "contrato" (semântica) por meio do WSD;
c) etapa 3: são criados os dois agentes respeitando a semântica;
d) etapa 4: é realizada a interação entre agente solicitante e agente provedor por meio da troca de mensagens.

Ainda no contexto de *web services*, o trabalho desenvolvido utiliza *web services* baseados no protocolo SOAP, que é um dos protocolos utilizados para a troca de dados entre os dois agentes envolvidos no modelo.

> SOAP Versão 1.2 (SOAP) é um protocolo leve destinado à troca de informações estruturadas em um ambiente descentralizado, distribuído. Usa a tecnologia XML para definir um framework de mensagens extensível provendo uma construção para mensagens que podem ser trocadas através de uma variedade de protocolos subjacentes. O framework foi desenvolvido para ser independente de qualquer modelo de programação em particular e outras semânticas especificas. (W3C, 2004a, tradução nossa).

Este protocolo baseia-se em mensagens descritas usando XML e pode ser dividido em quatro componentes que formam o SOAP. Formato, que especifica a forma da mensagem descrita na linguagem XML. Em outras palavras é o que torna um simples documento XML em uma mensagem SOAP. Envelope delimita por *tags* o conteúdo do documento XML, também contém atributos como o *namespace* e *schema*. Cabeçalho, espaço usado para inserir diretivas para o processador SOAP. Ligação (*binding*) contém informações usadas para realizar a ligação com o protocolo HTTP, que é o protocolo utilizado para o transporte da mensagem (CHAPPELL; JEWELL, 2002, p.28-32).

2.6 TRABALHOS CORRELATOS

As próximas seções apresentam três trabalhos que tem relação com o trabalho desenvolvido. O primeiro é um software comercial e os outros dois são trabalhos de conclusão de curso.

2.6.1 IBM Rational AppScan

O IBM Rational AppScan é uma ferramenta comercial que permite a realização de testes caixa branca em tempo de execução. É possível também utilizar um híbrido de testes caixa preta e análise dinâmica para simulação de ataques de segurança com um agente interno que monitora o comportamento do aplicativo durante o ataque. Portanto, o IBM Rational AppScan analisa a estrutura dinâmica (caixa preta) e o código fonte (caixa branca) e fornece integração com a gestão do ciclo de vida da aplicação (IBM, 2014). Em sua versão mais recente, o IBM Rational AppScan versão 8.0.0, foi incluída a funcionalidade que realiza testes caixa branca com análise em tempo de execução. Além disto, foi reestruturado para permitir um acoplamento maior ao sistema alvo, permitindo uma análise mais profunda.

2.6.2 Software para verificação de conformidade de sistemas à norma ISO/IEC 15408

O trabalho de conclusão de curso desenvolvido por Trapp (2010) teve como objetivo desenvolver um software para auxiliar na tarefa de auditoria de segurança com base na norma ISO/IEC 15.408. A análise de segurança é feita em sistemas alvo escritos na linguagem Java. A comunicação entre software de avaliação e sistema alvo é feita através de uma interface, que deve ser implementada pelo sistema alvo. Portanto, o software de avaliação baseia-se na interface para realizar a análise de segurança. Para cada requisito a ser avaliado, o auditor deve desenvolver um *plugin*.

Dentre as funcionalidades disponibilizadas estão (TRAPP, 2010, p.15):

a) analisar a conformidade do sistema avaliado à norma;
b) disponibilizar uma *Application Programming Interface* (API) para criação de *plugins* de testes;
c) disponibilizar um *check-list* para os requisitos que não serão analisados e disponibilizar um relatório assinalando os requisitos atendidos pelo sistema avaliado.

2.6.3 Plugins para testes semi-automatizados de conformidade com a norma ISO/IEC 15408

O trabalho de conclusão de curso desenvolvido por Herkenhoff (2011) foi uma extensão do trabalho de Trapp (2010) e teve como objetivo o desenvolvimento de *plugins* de testes semi-automatizados para o software de avaliação de segurança resultante do trabalho estendido.

Os testes semi-automatizados desenvolvidos foram:

a) verificação das regras de geração de dados para auditoria e auditoria selecionável;
b) verificação das regras de análise de violação em potencial, revisão de trilha de auditoria e auditoria de resposta automática;
c) verificação das regras de tratamento de falhas de autenticação, definição do usuário e especificação de senhas.

Os requisitos automatizados no trabalho de Herkenhoff (2011) são os de duas classes da norma, a *Security Audit* (FAU) e a *Identification and Authentication* (FIA).

3 DESENVOLVIMENTO

Este capítulo descreve a definição do modelo desenvolvido, os requisitos principais do problema a ser trabalhado, a especificação, a implementação e, por fim, os resultados e discussões.

3.1 DEFINIÇÃO DO MODELO

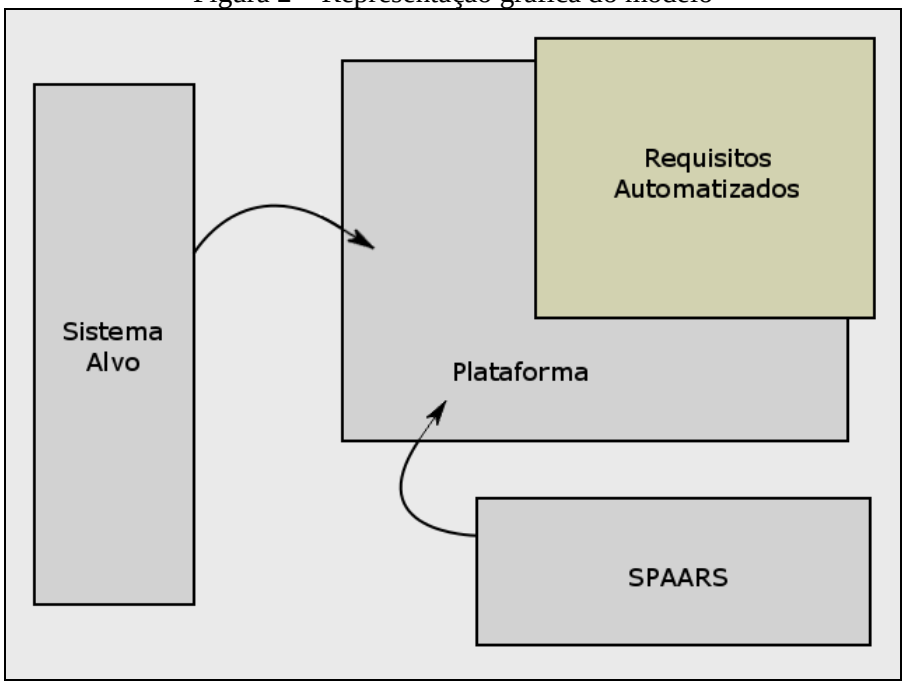
O desenvolvimento do modelo para avaliação automatizada de segurança – chamado no restante do trabalho apenas como modelo – tornou-se necessário devido à necessidade da comunicação entre um sistema em desenvolvimento – que neste trabalho foi chamado de sistema alvo – e o software que faz a avaliação automatizada de requisitos de segurança. Este modelo torna possível que um sistema escrito em qualquer linguagem que suporte a tecnologia *web service*, possa ter sua segurança avaliada segundo a norma ISO/IEC 15.408.

O modelo possui quatro elementos, são eles:

a) sistema alvo: é o sistema sob avaliação, o TOE do CC;
b) plataforma de avaliação de segurança: é a parte do modelo que contém os requisitos automatizados e os disponibiliza. Foi adicionada ao modelo para garantir a independência dos requisitos automatizados e permitir que sejam implementados novos requisitos sem a necessidade de alterações no software para avaliação automatizada de requisitos de segurança. A plataforma de avaliação de segurança no restante do texto será chamada apenas de plataforma;
c) requisitos automatizados: na norma existe uma série de requisitos, alguns possíveis de se automatizar e outros com grande complexidade que exige a avaliação de forma manual. Os requisitos automatizados são aqueles que puderam ser descritos na forma de programa;
d) software para avaliação automatizada de requisitos de segurança: é o software que fará a avaliação da segurança usando informações disponibilizadas pela plataforma. O software para avaliação automatizada de requisitos de segurança desenvolvido neste trabalho foi chamado de SPAARS (Software Para Avaliação Automatizada de Requisitos de Segurança).

A Figura 2 apresenta a representação gráfica do modelo.

Figura 2 – Representação gráfica do modelo



Na Figura 2 são mostrados os elementos do modelo. As setas mostram suas dependências. O sistema alvo e o SPAARS dependem da plataforma e a plataforma é independente deles. Isto é importante para permitir o reaproveitamento dos requisitos automatizados. Com o modelo, a implementação dos requisitos automatizados ficam encapsulados na plataforma. Permitindo desta forma que novos requisitos sejam criados sem a necessidade de alterações no SPAARS.

Para tornar o sistema alvo avaliável existe um custo, que é a utilização de um protocolo no qual eventos são acionados. Os eventos no contexto do modelo são chamadas à *web services*, que devem ser acionados em determinadas ocasiões da execução do sistema alvo. A utilização do protocolo foi necessária, pois, por mais independente que sejam os elementos do modelo, é inevitável que em tempo de execução sejam trocadas informações. A especificação do protocolo é apresentada através de diagramas na seção 3.3.

3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos do problema estão divididos entre os requisitos do modelo e os do software para avaliação automatizada de requisitos de segurança.

Os requisitos do modelo são:

a) permitir a comunicação entre o software de avaliação de segurança e o sistema alvo (Requisito Não Funcional - RNF);
b) prover uma conexão homogênea com o sistema alvo (RNF);

c) prover interface de Java com outras linguagens de programação (RNF);
d) permitir a criação de softwares de avaliação de segurança usando o modelo especificado (RNF);
e) permitir o desenvolvimento de novos requisitos automatizados (RNF).

Os requisitos do software para avaliação automatizada de requisitos de segurança são:

a) fornecer uma lista dos requisitos da norma ISO/IEC 15.408 que foram automatizados, para que possam ser selecionados pelo auditor (Requisito Funcional - RF);
b) permitir a auditoria automática de acordo com os itens da norma selecionados pelo auditor (RF);
c) gerar um relatório com as informações obtidas pela auditoria (RF);
d) analisar de forma automatizada os requisitos de segurança do sistema alvo (RF);
e) ser desenvolvido usando a linguagem Java (RNF);
f) utilizar o modelo desenvolvido (RNF).

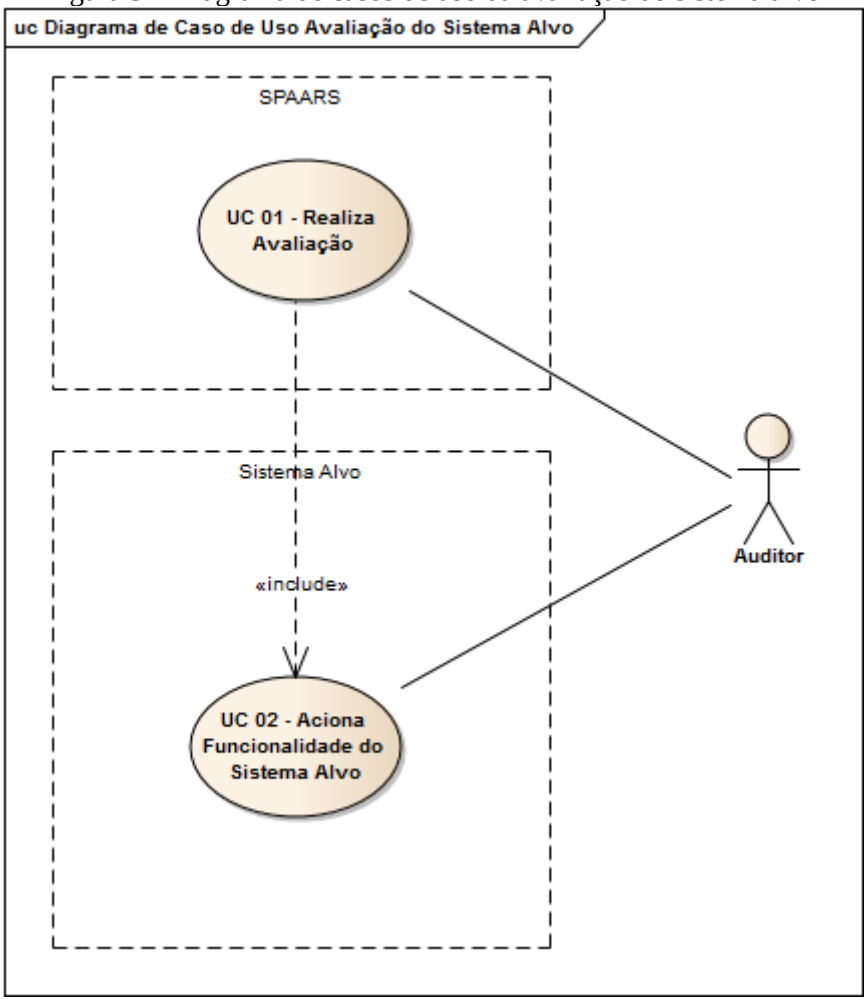
3.3 ESPECIFICAÇÃO

Na especificação são apresentados alguns diagramas da *Unified Modeling Language* (UML) para representar de forma gráfica e textual o trabalho desenvolvido. Para a construção dos diagramas foi utilizada a ferramenta Enterprise Architect da Sparx Systems. Foram desenvolvidos os diagramas de componentes, pacotes, classe, sequência, estados e casos de uso.

3.3.1 Diagrama de casos de uso

A principal interação com o modelo é feita pelo auditor. Ele é responsável por utilizar o SPAARS e analisar os resultados gerados por ele. A Figura 3 apresenta o diagrama de casos de uso que envolve a interação do auditor com o SPAARS.

Figura 3 – Diagrama de casos de uso da avaliação do sistema alvo



O Quadro 1 mostra de forma detalhada os passos executados no caso de uso `Realiza Avaliação`.

Quadro 1 – Descrição do caso de uso `Realizar Avaliação`

UC 01 – Realiza Avaliação	
Pré-condições	Selecionar no mínimo um requisito de segurança e selecionar o sistema alvo
Cenário principal	1) o auditor clica no botão iniciar 2) o software se conecta à plataforma 3) o software inicia a verificação dos estados dos requisitos 4) o auditor interage com o sistema alvo 5) o software aguarda o encerramento 6) o auditor clica no botão encerrar 7) o software encerra a verificação dos estados dos requisitos 8) o auditor fecha o software
Cenário alternativo	No passo 4, o auditor irá interagir com o sistema alvo: 1) é executado o UC 02 2) é executado o passo 1 até: o auditor julgar necessário ou todos os requisitos alcançarem o estado "OK" ou todos os requisitos alcançarem o estado "FALHA" 3) volta ao cenário principal
Cenário alternativo	No passo 8, o auditor pode iniciar uma nova avaliação de segurança: 1) volta ao passo 1 do cenário principal
Cenário alternativo	No passo 8, o auditor pode gerar o relatório resultante da última avaliação: 1) o auditor clica no botão relatório 2) o sistema apresenta mensagem do diretório onde o relatório foi gerado 3) volta ao cenário principal

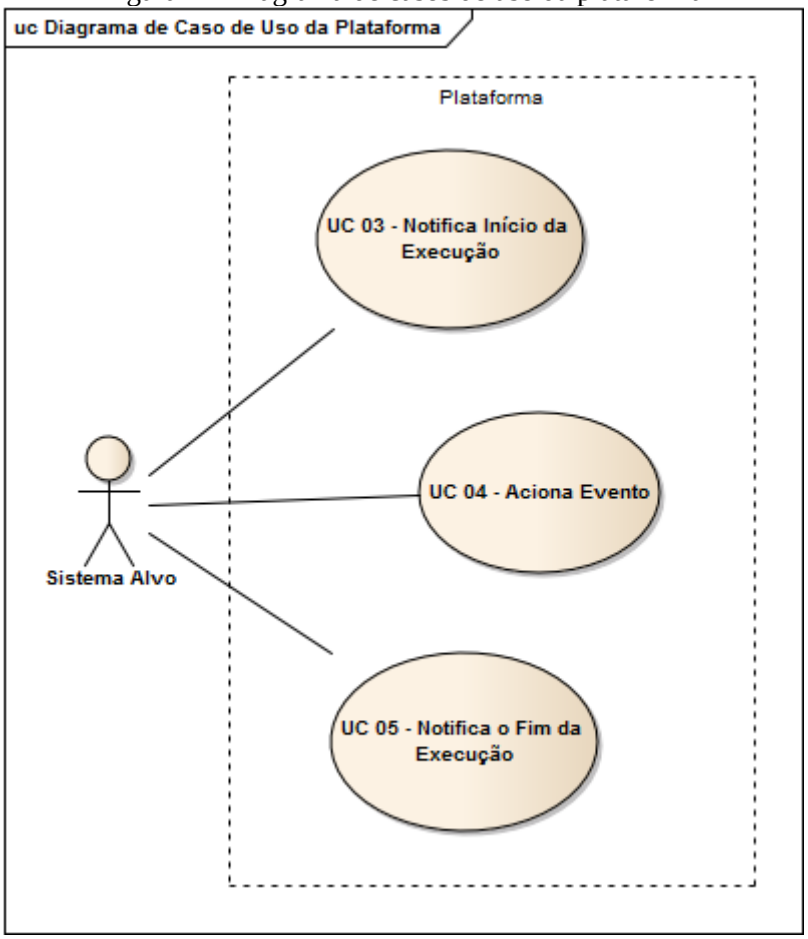
O Quadro 2 mostra de forma detalhada os passos envolvidos no caso de uso `Aciona Funcionalidade do Sistema Alvo`. É importante destacar que este caso de uso pode ser executado várias vezes durante a avaliação do sistema.

Quadro 2 – Descrição do caso de uso `Aciona Funcionalidade do Sistema Alvo`

UC 02 – Aciona Funcionalidade do Sistema Alvo	
Cenário principal	1) o auditor aciona função do sistema 2) o sistema aciona evento para a avaliação de segurança 3) o auditor conclui a função 4) o caso de uso é encerrado
Cenário alternativo	No passo 1, dependendo da função são solicitados dado de entrada: 1) o sistema solicita dados de entrada do auditor 2) o auditor fornece dados de entrada 3) volta ao cenário principal
Pós-condições	Estado do requisito de segurança é atualizado

A Figura 4 exibe o diagrama de casos de uso da plataforma, que são executados pelo sistema alvo.

Figura 4 – Diagrama de casos de uso da plataforma



O Quadro 3 mostra de forma detalhada os passos envolvidos no caso de uso `Notifica Início da Execução`.

Quadro 3 – Descrição do caso de uso `Notifica Início da Execução`

UC 03 – Notifica Início da Execução	
Cenário principal	1) o sistema alvo envia a notificação 2) a plataforma cria um arquivo XML para o sistema 3) a plataforma retorna uma identificação 4) o caso de uso é encerrado
Pós-condições	O sistema alvo guarda em memoria a identificação

O Quadro 4 mostra de forma detalhada os passos envolvidos no caso de uso `Aciona Evento`.

Quadro 4 - Descrição do caso de uso `Aciona Evento`

UC 04 – Aciona Evento	
Cenário principal	1) o sistema alvo chama o *web service* disponibilizado pela plataforma 2) a plataforma verifica a chamada de acordo com o requisito correspondente 3) o caso de uso é encerrado
Pós-condições	O estado de um ou mais requisitos pode ser alterado

O Quadro 5 mostra de forma detalhada os passos envolvidos no caso de uso `Notifica Fim da Execução`.

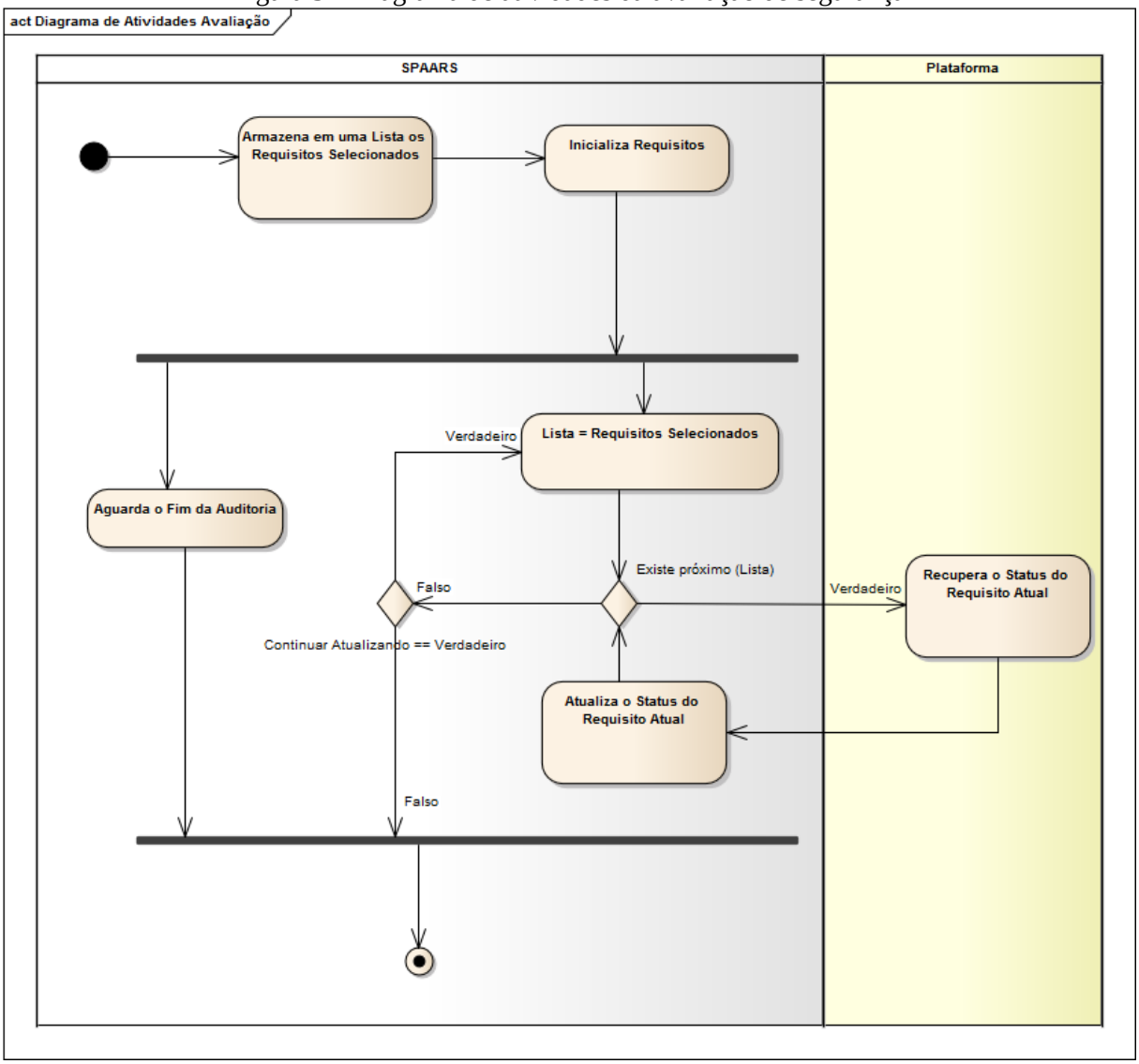
Quadro 5 - Descrição do caso de uso `Notifica o Fim da Execução`

UC 05 – Notifica o Fim da Execução	
Pré-condições	A notificação de início deve ter sido enviada
Cenário principal	1) o sistema alvo envia a notificação 2) a plataforma elimina o arquivo XML correspondente ao sistema 3) o caso de uso é encerrado

3.3.2 Diagrama de atividades

O diagrama de atividades da Figura 5 complementa o diagrama de casos de uso da Figura 3. O diagrama de atividades demonstra como é feita a avaliação da segurança por parte do software, ou seja, o SPAARS.

Figura 5 – Diagrama de atividades da avaliação de segurança



Os passos do diagrama são:

a) `Armazena em uma Lista os Requisitos Selecionados`: os requisitos selecionados na interface gráfica pelo auditor são armazenados em uma lista;

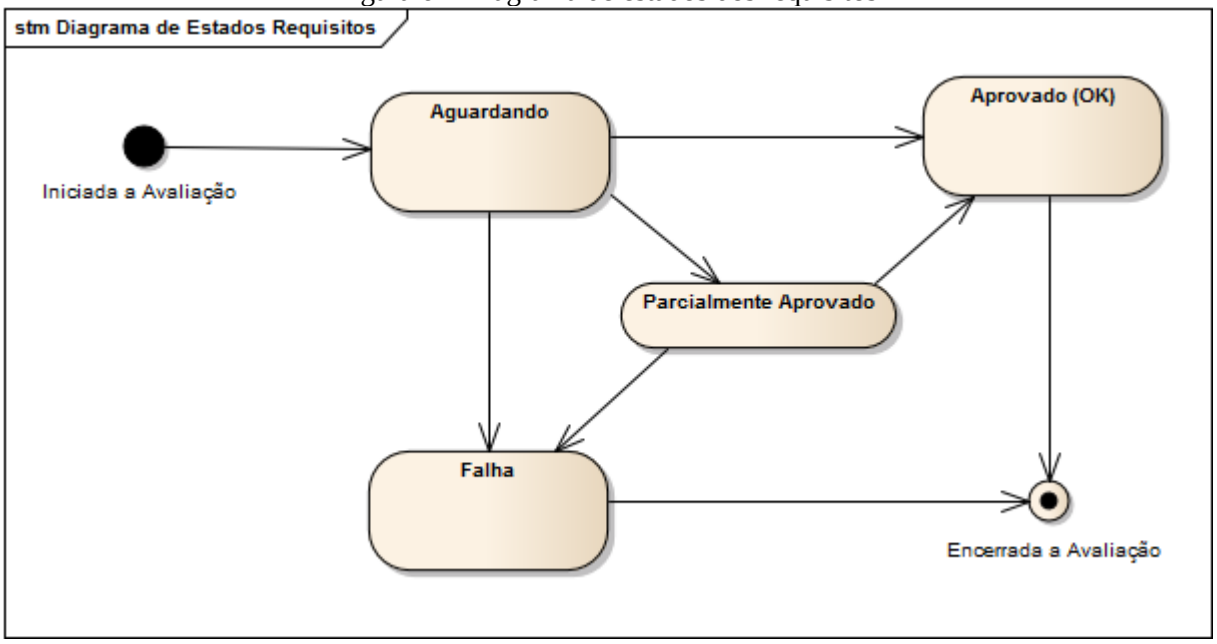
b) `Inicializa Requisitos`: os requisitos são inicializados com o estado aguardando, após a execução deste passo a execução é dividida em duas *threads*:
   - *thread* 01/ `Aguarda o Fim da Auditoria`: aguarda o auditor encerrar a avaliação. A avaliação é encerrada ao clicar no botão encerrar,
   - *thread* 02/ `Lista = Requisitos Selecionados`: atribui os itens da lista de requisitos selecionados para a lista que será iterada,
   - *thread* 02/ `Existe próximo (Lista)`: testa se existe um próximo requisito em `Lista`,
   - *thread* 02/ `Recupera Status do Requisito Atual`: o requisito atual é o requisito da iteração atual sobre `Lista`, o estado deste requisito é recuperado da plataforma,
   - *thread* 02/ `Atualiza o Status do Requisito Atual`: atualiza o estado do requisito recuperado,
   - *thread* 02/ `Continuar Atualizando == Verdadeiro`: testa se ainda é necessário continuar atualizando os estados dos requisitos. O valor de `Continuar Atualizando` é alterado para `Falso` quando o botão encerrado é pressionado;

c) Junção (*join*) das *threads* 01 e 02. É encerrada a execução.

3.3.3 Diagrama de estados

Durante a avaliação de segurança os estados dos requisitos podem mudar dependendo dos eventos acionados. Os estados e transições são mostrados no diagrama da Figura 6.

Figura 6 – Diagrama de estados dos requisitos



Os estados do diagrama são:

a) `Aguardando`: primeiro estado assumido pelo requisito. Pode transitar para cada um dos estados seguintes. Se a avaliação for encerrada e o requisito estiver neste estado ele é alterado para `Falha`, pois não foi acionado nenhum evento que comprove a condição de aprovação do requisito;

b) `Parcialmente Aprovado`: um requisito transita para este estado quando algum evento foi acionado, e, a partir do contexto atual da avaliação não foi possível comprovar sua condição de sucesso ou falha. Este estado pode transitar tanto para `Aprovado` quanto para `Falha`. Existem requisitos, que, durante a avaliação não podem ser considerados como aprovados ou falhos. Portanto cabe ao auditor julgar se os testes feitos são suficientes para comprovar a condição de aprovação do requisito;

c) `Aprovado` ou `OK`: um requisito transita para este estado quando a partir do contexto da avaliação é possível comprovar totalmente a condição de aprovado. Quando este estado é atingido não pode ser alterado para nenhum outro estado;

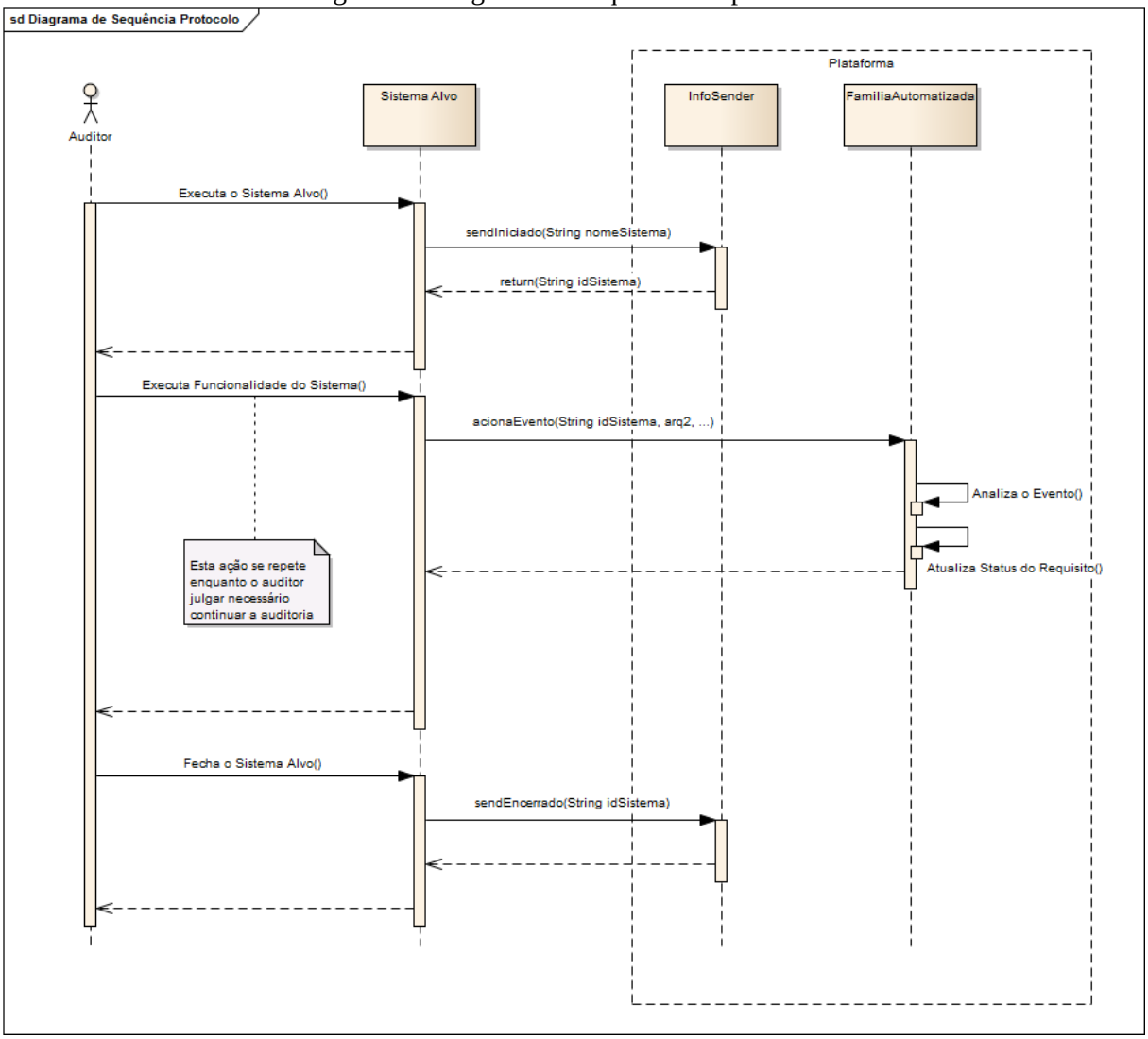
d) `Falha`: um requisito transita para este estado quando a partir do contexto da avaliação é possível comprovar totalmente a condição de falha. Quando este estado é atingido não pode ser alterado para nenhum outro estado.

Na seção 3.4.7 onde é descrita a implementação dos requisitos automatizados para o teste da plataforma, são explicadas as condições para a mudança de estado de todos os requisitos automatizados.

3.3.4 Diagrama de sequência

Para ser possível a avaliação do sistema alvo foi necessária a especificação de um protocolo, que estipula a ordem da troca de mensagens entre sistema alvo e plataforma. O protocolo é especificado através do diagrama de sequência da Figura 7.

Figura 7 – Diagrama de sequência do protocolo



Os passos do diagrama são:

a) o `Auditor` inicia a execução do sistema alvo;

b) o `Sistema Alvo` por sua vez notifica através do `InfoSender` o início de sua execução passando o nome do sistema;

c) o `InfoSender` retorna ao `Sistema Alvo` uma identificação – usada nos eventos subsequentes;

d) o `Auditor` executa alguma funcionalidade no `Sistema Alvo`. Uma funcionalidade, por exemplo, seria o *login* no sistema ou então o cadastro de um produto. Este passo se repete até o auditor julgar necessário;

e) o `Sistema Alvo` aciona um evento na `FamiliaAutomatizada` correspondente. A família e o evento acionado dependem do contexto da execução. Deve ser passada a identificação do sistema e os parâmetros quando necessário;

f) a `FamiliaAutomatizada` analisa o evento acionado e dependendo da análise altera o estado do requisito que está sendo avaliado;

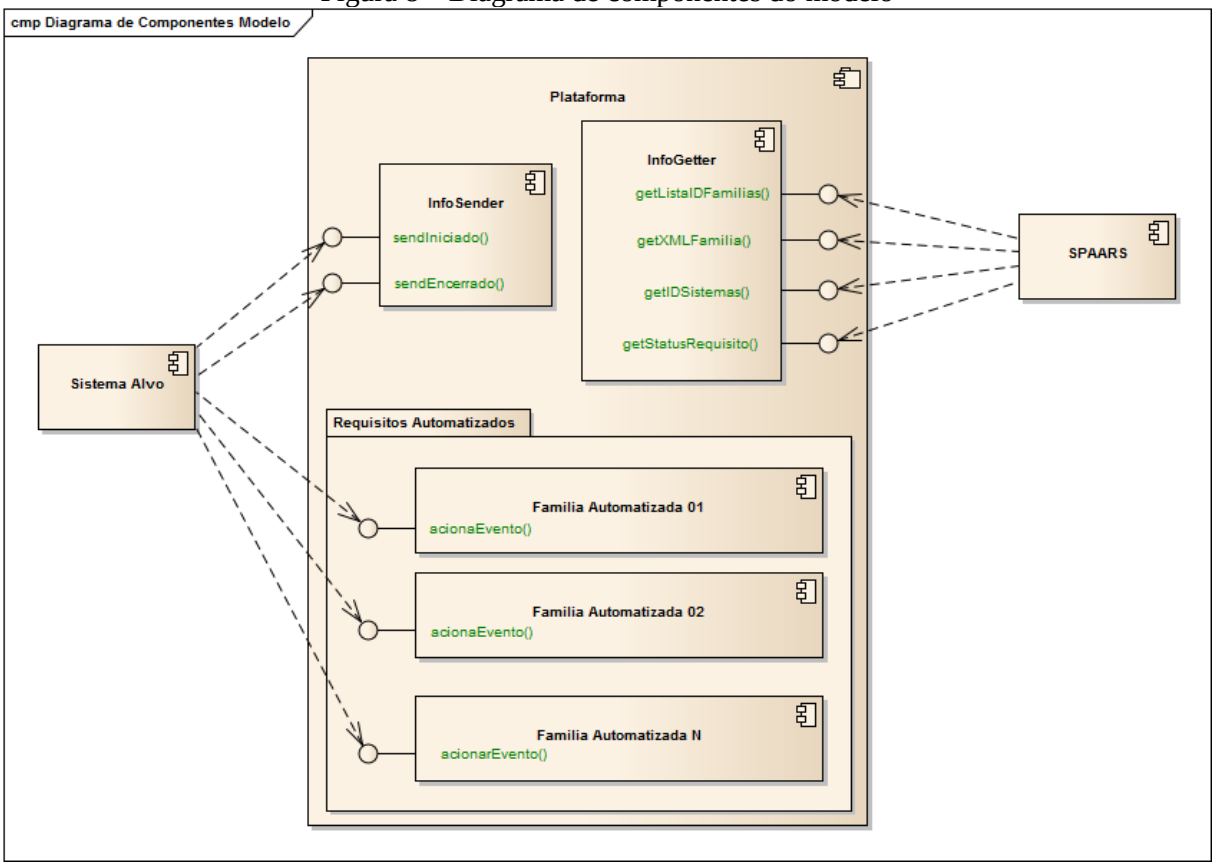
g) o `Auditor` fecha o `Sistema Alvo`;

h) o `Sistema Alvo` notifica seu encerramento através do `InfoSender` informando sua identificação.

3.3.5 Diagrama de componentes

A plataforma é o elemento central que media a comunicação entre sistema alvo e o SPAARS e contém a implementação dos requisitos. Para a avaliação de segurança são disponibilizados eventos que são acionados pelo sistema alvo. A Figura 8 ilustra como é a estrutura dos *web services* no modelo.

Figura 8 – Diagrama de componentes do modelo



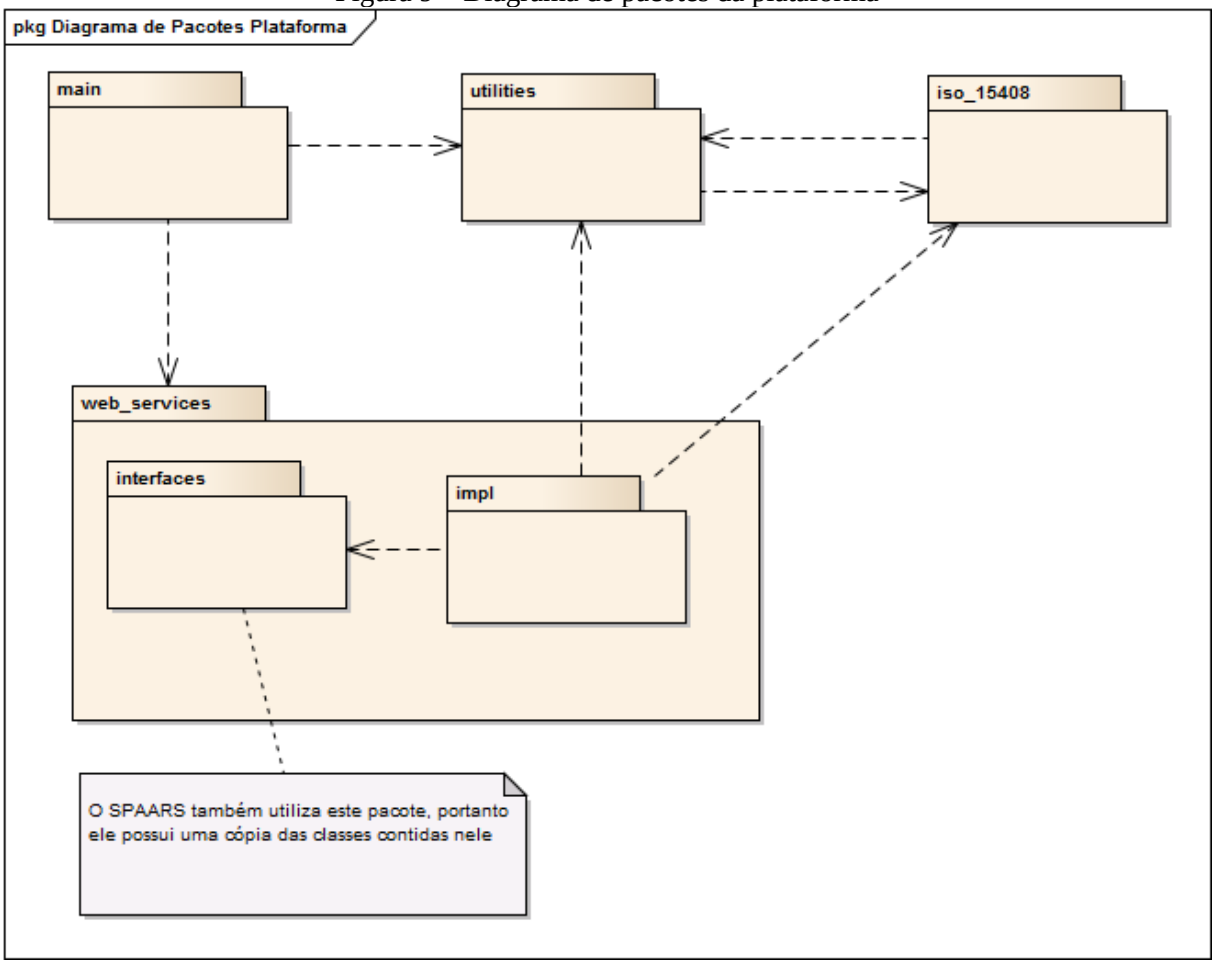
Os componentes `InfoSender`, `InfoGetter` e os componentes do pacote `Requisitos Automatizados` são *web services*. O `InfoSender` é usado pelo sistema alvo e contém métodos para enviar informações sobre o início e término de sua execução. `InfoGetter`

usado pelo SPAARS é responsável por disponibilizar informações sobre as famílias de requisitos automatizados além do estado atual dos requisitos que estão sendo avaliados. Os *web services* do pacote `Requisitos Automatizados` presentes no diagrama foram incluídos apenas para ilustrar a relação entre as famílias de requisitos automatizados e o sistema alvo. Cada família automatizada possui um conjunto de eventos. Os eventos são acionados em ocasiões especificas da execução do sistema alvo. É através dos eventos que ocorre a mudança de estado de um requisito.

3.3.6 Diagrama de pacotes

A Figura 9 apresenta o diagrama de pacotes da plataforma. O diagrama mostra a organização dos pacotes presentes e as dependências entre eles.

Figura 9 – Diagrama de pacotes da plataforma



Os pacotes do diagrama são:

a) `main`: contém apenas a classe que possui o método `main()`, responsável pelo inicio da execução da plataforma;

b) `utilities`: contém classes utilitárias. Usadas para simplificar tarefas como leitura

e escrita de arquivos, entre outras tarefas que auxiliam na avaliação dos requisitos;

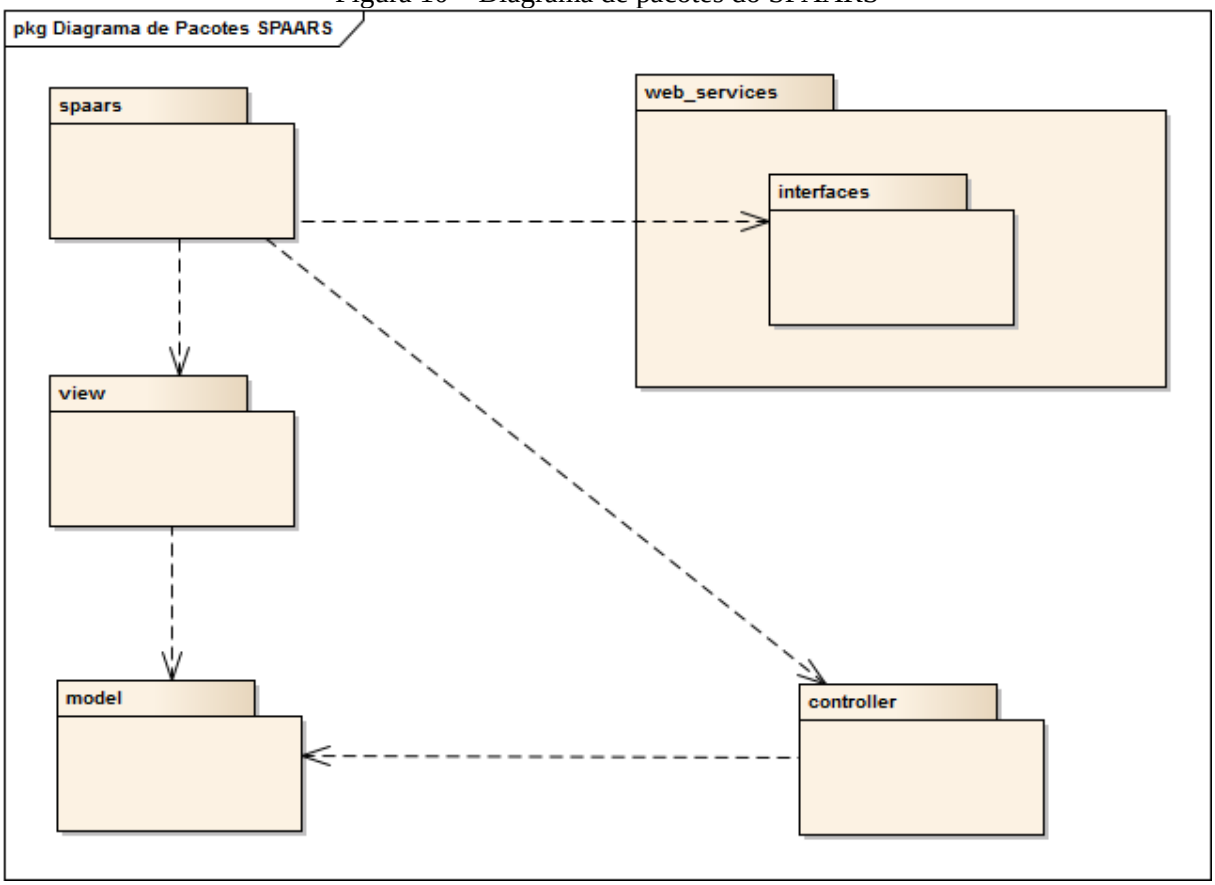
c) `iso_15408`: contém as classes que representam famílias e requisitos da norma ISO/IEC 15.408. Neste pacote existe ainda uma classe de constantes utilizadas pela plataforma;

d) `web services`: está dividido em dois subpacotes:

- `interfaces`: contém as interfaces de todos os *web services*. As classes deste pacote são utilizadas também pelo SPAARS,
- `impl`: contém as implementações das interfaces do pacote `interfaces`.

A Figura 10 apresenta o diagrama de pacotes do SPAARS.

Figura 10 – Diagrama de pacotes do SPAARS



Os pacotes do diagrama são:

a) `spaars`: contém a classe que inicia o programa;

b) `web_services` / `interfaces`: contém as interfaces dos *web services* disponibilizados pela plataforma. Através destas interfaces é possível obter instâncias dos *web services* necessários;

c) `view`: contém as classes com o código da interface gráfica com o usuário;

d) `model`: contém as classes para a representação das informações para o relatório e

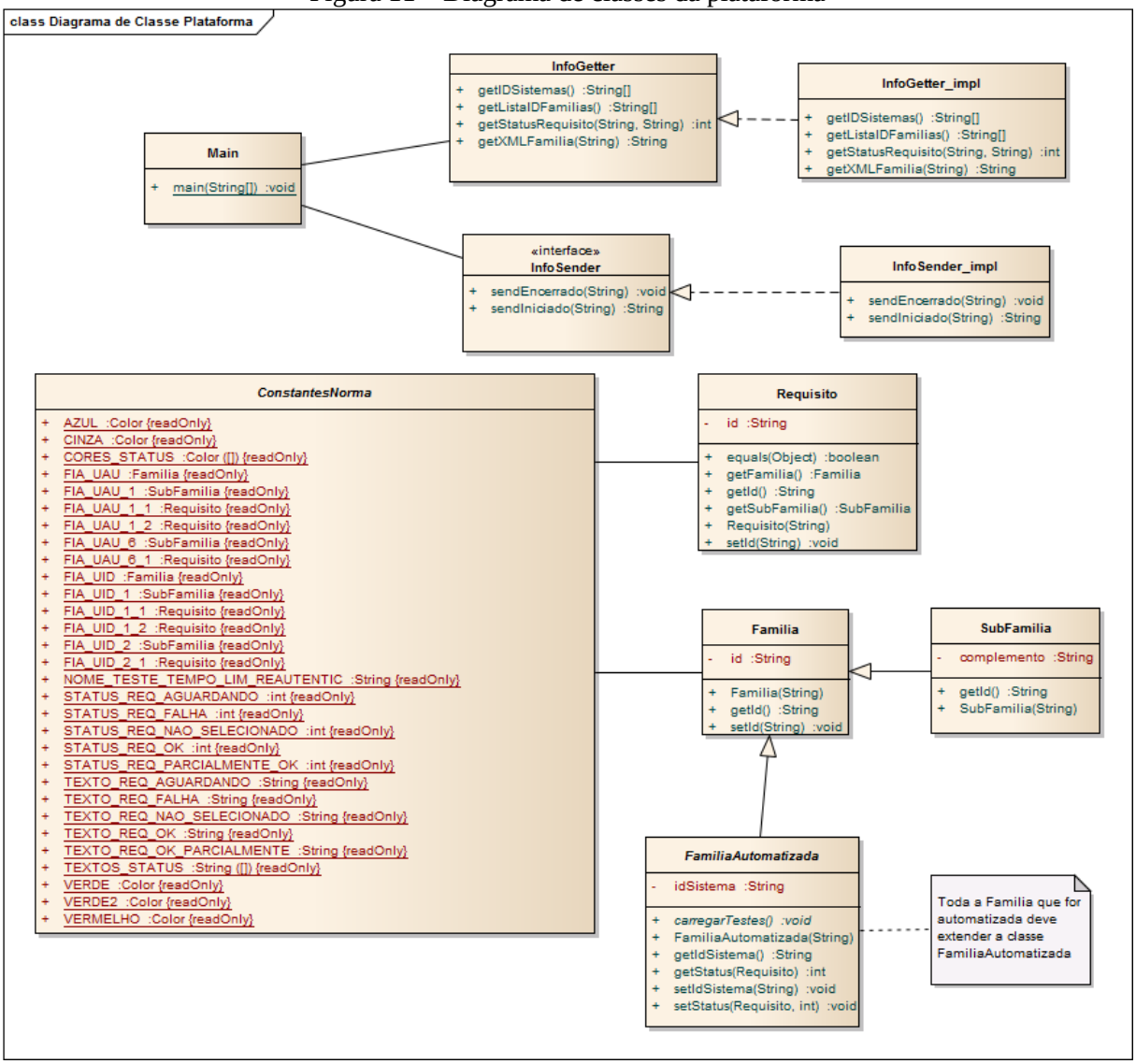
classes utilizadas para manipulação das famílias e requisitos da norma;

e) `controller`: contém classes para a manipulação de arquivos XML e geração de relatório.

3.3.7 Diagrama de classes

Na Figura 11 é apresentado o diagrama de classes da plataforma. O diagrama não contém a especificação dos requisitos automatizados, pois é apresentada uma seção específica.

Figura 11 – Diagrama de classes da plataforma



As classes do diagrama são:

a) `Main`: classe que contém o método `main()` responsável por iniciar o programa;

b) `InfoGetter`: interface *web service*. Contém métodos que são usados pelo SPAAR. Usando esta interface é possível desenvolver outros softwares para avaliação

automatizada de requisitos de segurança;

c) `InfoGetter_impl`: classe que implementa `InfoGetter`;

d) `InfoSender`: interface *web service*. Contém métodos que são usados pelo sistema alvo;

e) `InfoSender_impl`: classe que implementa `InfoSender`;

f) `ConstantesNorma`: classe que contém constantes de `Familia`, `SubFamilia` e `Requisitos`, contém ainda constantes referentes aos estados dos requisitos avaliados;

g) `Requisito`: classe que representa um requisito de segurança da norma ISO/IEC 15.408;

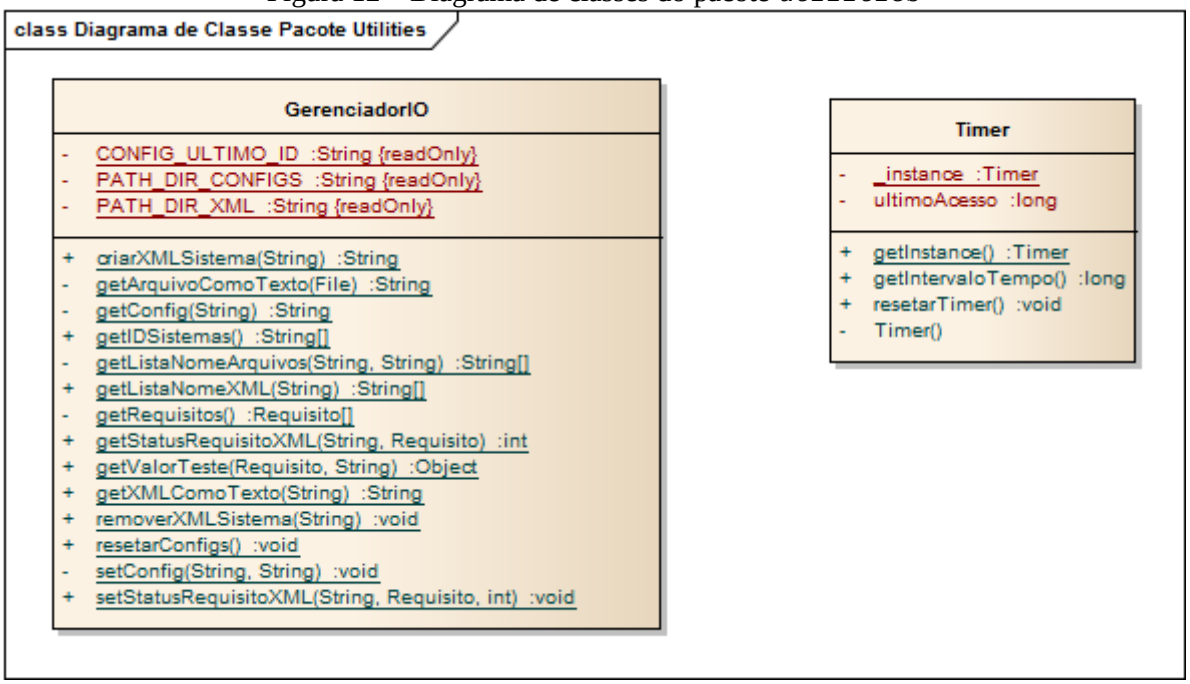
h) `Familia`: classe que representa uma família de requisitos da norma;

i) `SubFamilia`: classe que representa uma subfamília. Por exemplo, FIA_UAU identifica uma família e FIA_UAU.1 é a identificação de uma subfamília;

j) `FamiliaAutomatizada`: classe que representa uma família de requisitos que será automatizada. É estendida pelas classes que implementam a lógica de avaliação de segurança dos requisitos.

Na Figura 12 é apresentado o diagrama de classes do pacote `utilities` que também pertence à plataforma.

Figura 12 – Diagrama de classes do pacote `utilities`



As classes do diagrama são:

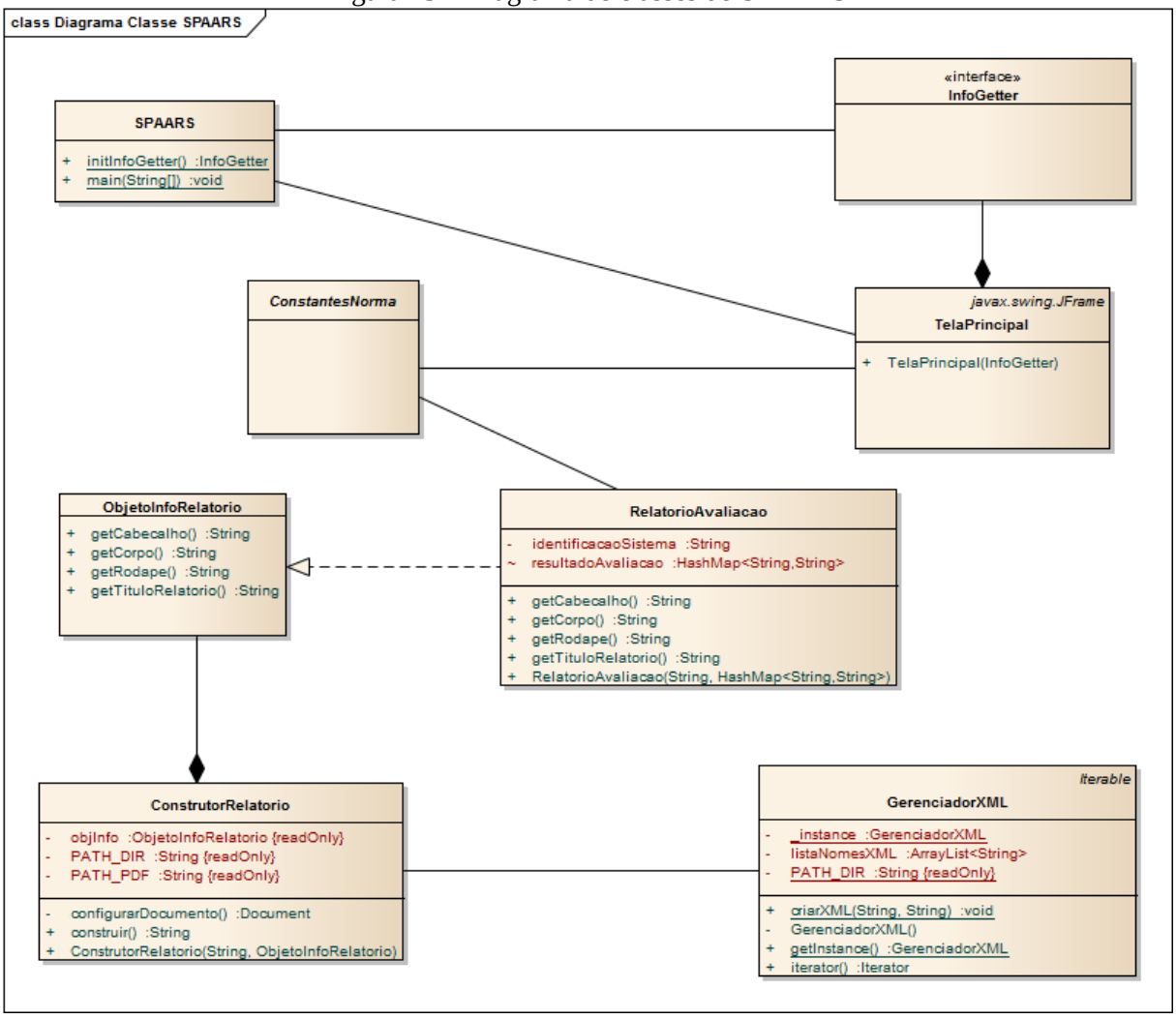
a) `GerenciadorIO`: classe que controla as operações que envolvem leitura e escrita

em arquivos. Esta classe manipula os arquivos XML e um arquivo de configurações;

b) `Timer`: classe criada para marcar intervalos de tempo. O método `resetarTimer()` inicia a contagem do tempo. O método `getIntervaloTempo()` retorna o tempo passado desde o inicio da contagem – em milissegundos.

Na Figura 13 é apresentado o diagrama de classes do SPAARS.

Figura 13 – Diagrama de classes do SPAARS



As classes do diagrama são:

a) `SPAARS`: classe que contém o método `main()`, responsável por iniciar a execução;

b) `InfoGetter`: interface usada para obter uma referência ao *web service* correspondente. É disponibilizado pela plataforma e contém métodos necessários para a obtenção de informações sobre as famílias automatizadas e sobre o sistema alvo;

c) `ConstantesNorma`: classe que contém as constantes utilizadas para a verificação

dos valores vindos da plataforma;

d) `TelaPrincipal`: classe responsável por disponibilizar a interface gráfica com o usuário. Estende a classe `JFrame`. Seu construtor recebe uma referência a um objeto `InfoGetter` utilizado para obter informações durante a avaliação de segurança;
e) `ObjetoInfoRelatorio`: classe que representa um objeto a partir do qual é possível obter informações para a geração de relatório;
f) `RelatorioAvaliação`: estende a classe `ObjetoInfoRelatorio`. Representa um objeto para a geração de relatórios de avaliação de segurança;
g) `ConstrutorRelatorio`: classe que com a finalidade de gerar relatórios utilizando objetos da classe `ObjetoInfoRelatorio`;
h) `GerenciadorXML`: classe responsável por gerenciar os arquivos de famílias automatizadas obtidas da plataforma.

3.4 IMPLEMENTAÇÃO

Nesta seção são descritas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.4.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento da plataforma, dos requisitos automatizados para teste e do SPAARS foi utilizada a linguagem Java com o *Integrated Development Environment* (IDE) NetBeans versão 8.0.2. Foram utilizados ainda conceitos de orientação a objeto.

3.4.2 DOM

A API *Document Object Model* (DOM) foi utilizada para a manipulação de documentos XML. Dentre as bibliotecas disponíveis a DOM foi escolhida pela praticidade de sua utilização e por estar presente na API padrão da Java não necessitando adicionar uma biblioteca separada para este fim.

> DOM é uma interface de programação, baseada no modelo de objetos, que permite a manipulação e transformação de documentos em XML. A interface DOM manipula documentos XML na forma de uma estrutura em árvore. Quando um documento é carregado na memória do computador, suas estruturas podem ser lidas e manipuladas através do objeto DOM. (GRONER, 2009).

Documentos XML foram utilizados para armazenar as informações sobre as famílias de requisitos automatizados, portanto cada arquivo XML corresponde a uma família. Sendo

que uma família possui um ou mais requisitos de segurança. O Quadro 6 apresenta o formato dos documentos XML das famílias.

Quadro 6 – Formato do XML das famílias

```xml
<?xml version="1.0" encoding="UTF-8"?>

<familia id="FFF_FFF">

    <requisito id="FFF_FFF.1.1">
        <teste nome="teste1" valor="0"></teste>
        <teste nome="testeN" valor="0"></teste>
    </requisito>

    <requisito id="FFF_FFF.N.M">
        <teste nome="teste1" valor="0"></teste>
        <teste nome="testeN" valor="0"></teste>
    </requisito>

</familia>
```

O elemento `familia` contém os elementos `requisito`, estes contêm os elementos `teste`, que por sua vez descrevem os parâmetros para a avaliação do requisito. Pode-se destacar que, existem requisitos que não necessitam de parâmetros, neste caso, o elemento `teste` é inexistente. Basicamente este XML serve para descrever os requisitos automatizados e suas configurações. É importante destacar que todos os arquivos XML de famílias são mantidos na plataforma.

Quando o SPAARS conecta-se à plataforma, ele adquire uma cópia dos arquivos XML de famílias para que possa exibir os requisitos selecionáveis para o auditor. O Quadro 7 apresenta um trecho de código do SPAARS que analisa o arquivo XML de uma família para realizar a construção de uma árvore que exibe os requisitos na interface gráfica.

Quadro 7 – Trecho de código da API DOM

```
DocumentBuilder docBuilder;
Document doc;
NodeList nodes;
NodeList nodesRequisitos;
NodeList nodesTestes;
String[] idSeparado;
Element elemento;

docBuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
doc = docBuilder.parse(arqXML);
nodes = doc.getElementsByTagName("familia");

for (int i = 0; i < nodes.getLength(); i++) {
    if (nodes.item(i).getNodeType() == Node.ELEMENT_NODE) {
        elemento = (Element) nodes.item(i);
        raiz = new DefaultMutableTreeNode(elemento.getAttribute("id"));
        nodesRequisitos = elemento.getElementsByTagName("requisito");

        for (int j = 0; j < nodesRequisitos.getLength(); j++) {
            if (nodesRequisitos.item(j).getNodeType() == Node.ELEMENT_NODE) {
                elemento = (Element) nodesRequisitos.item(j);

                idRequisito = elemento.getAttribute("id");
                listaIDReq.add(idRequisito);
```

O objeto da classe `Document` é a representação do documento. Como documentos XML tem a estrutura de árvore, pode-se dizer que `Document` é a raiz dela e a partir dele é possível iterar sobre os níveis superiores da árvore. No código do Quadro 7 em um primeiro momento são percorridos os nós do elemento `familia` e para cada nó `familia` são percorridos os nós `requisito` e por fim para cada nó `requisito` são percorridos os nós `teste`. Desta forma o último nível da árvore será o dos elementos `teste`. O Quadro 7 apresenta apenas a iteração dos níveis `familia` e `requisito`. Devido à extensão código foi colocado apenas um trecho para facilitar a leitura.

3.4.3 Web services

Os *web services* são disponibilizados pela plataforma e são acessados tanto pelo sistema alvo quanto pelo SPAARS. Foi utilizado o protocolo SOAP e para a implementação dos *web services* foi utilizada a API da biblioteca padrão Java.

A implementação dos *web services* contém duas partes sendo uma interface, que é visível para o cliente do *web service* e uma classe que implementa os *web services* especificados na interface. No Quadro 8 e Quadro 9 são mostrados exemplos de interfaces para o desenvolvimento dos *web services*.

Quadro 8 – Trecho de código da interface *web service* `InfoGetter`

```
@WebService
@SOAPBinding(style = SOAPBinding.Style.RPC)
public interface InfoSender {

    @WebMethod public String sendIniciado(String nomeSistema);
    @WebMethod public void sendEncerrado(String identificacaoSistema);

}
```

Esta interface é utilizada pelo sistema alvo e tem como objetivo a notificação de início e o encerramento da execução, respectivamente enviadas pelos métodos `sendIniciado()` e `sendEncerrado()`.

Quadro 9 – Trecho de código da interface *web service* `InfoSender`

```
@WebService
@SOAPBinding(style = SOAPBinding.Style.RPC)
public interface InfoGetter {

    @WebMethod public String[] getListaIDFamilias();
    @WebMethod public String getXMLFamilia(String id);
    @WebMethod public String[] getIDSistemas();
    @WebMethod public int getStatusRequisito(String identificacaoSistema,
                                             String idReq);
}
```

Esta interface está disponível para o SPAARS e disponibiliza os métodos para adquirir informações da plataforma. O método `getListaIDFamilias()` retorna uma lista dos identificadores das famílias de requisitos automatizados. Os identificadores são usados posteriormente no método `getXMLFamilia()` para recuperar os documentos XML correspondente à uma determinada família. O método `getIDSistemas()` é usado para recuperar os identificadores dos sistemas para que se possa selecionar o sistema que será avaliado. Por fim o método `getStatusRequisito()` retorna o estado do requisito solicitado para um determinado sistema sob avaliação.

Cada família automatizada tem um *web service* correspondente e os métodos disponibilizados são os eventos. Todos os *web services* são disponibilizados pela plataforma. No Quadro 10 é mostrado o trecho de código onde os *web services* são disponibilizados.

Quadro 10 – Trecho de código da disponibilização dos *web services*

```
public static void main(String[] args) {
    InfoGetter ig;
    InfoSender is;

    FIA_UAU fia_uau;

    ig = new InfoGetter_impl();
    is = new InfoSender_impl();

    fia_uau = new FIA_UAU_impl();

    GerenciadorIO.resetarConfigs();

    Endpoint.publish("http://127.0.0.1:9876/info_getter", ig);
    Endpoint.publish("http://127.0.0.1:9876/info_sender", is);

    Endpoint.publish("http://127.0.0.1:9876/fia_uau", fia_uau);


}
```

Neste trabalho foi adotado um padrão para o nome das classes que implementam os *web services*. São nomeadas pelo nome da interface seguido por `_impl`. No trecho de código são instanciadas as classe que implementam os *web services* e através do método estático `publish()` da classe `Endpoint` são disponibilizados. Os serviços são disponibilizados na maquina local e acessados pela porta 9876.

3.4.4 iText

O iText é uma biblioteca Java para a manipulação de documentos em *Portable Document Format* (PDF) e pode ser usada para criar relatórios (MEDEIROS, 2013). Esta biblioteca foi utilizada para a geração dos relatórios resultantes da avaliação de segurança. O Quadro 11 apresenta um fragmento do código presente na classe `ConstrutorRelatorio`, responsável pela configuração do documento PDF e por sua geração.

Quadro 11 – Trecho de código do construtor de relatório

```
public ConstrutorRelatorio(String nomeArquivo, ObjetoInfoRelatorio objInfo)

private Document configurarDocumento() {...11 linhas }

public String construir() {
    Paragraph paragrafo;
    Phrase frase;
    Document doc;
    Font fonte;

    String textoAux;

    doc = configurarDocumento();

    try {
        PdfWriter.getInstance(doc, new FileOutputStream(PATH_PDF));
        doc.open();

        // Inserindo o titulo do relatório.
        paragrafo = new Paragraph();
        textoAux = objInfo.getTituloRelatorio();
        fonte = FontFactory.getFont("Times-Roman", 14, Font.BOLD);
        fonte.setColor(70, 130, 180);
        frase = new Phrase(textoAux, fonte);
        paragrafo.add(frase);
        paragrafo.setAlignment(Element.ALIGN_CENTER);
        paragrafo.setSpacingAfter(Utilities.millimetersToPoints(20));

        doc.add(paragrafo);
```

O construtor da classe `ConstrutorRelatorio` recebe um nome para o arquivo que será gerado e uma referência para um objeto `ObjetoInfoRelatorio` que contém as informações relevantes para a geração do relatório. Este objeto possui métodos que retornam o texto do título, cabeçalho, corpo e rodapé do relatório. No iText, elementos do texto são representados por classe. Parágrafos e frases são controlados pelas classes `Paragraph` e `Phrase` respectivamente. É possível inserir objetos `Phrase` em objetos `Paragraph` separadamente o que torna simples a aplicação de formatos específicos para trechos do texto. Os outros elementos do código presente no Quadro 11 são configurações no formato do texto do relatório gerado.

3.4.5 Desenvolvimento da plataforma

Conforme especificado a plataforma é responsável por disponibilizar os *web services* e manter sua implementação além de trocar informações com o sistema alvo e o SPAARS. Para a comunicação com o sistema alvo é disponibilizada a interface `InfoSender` e para a

comunicação com o SPAARS é disponibilizada a interface `InfoGetter`. O Quadro 12 e Quadro 13 contêm a implementação das interfaces. É importante destacar que as tarefas que incluem leitura e escrita de arquivos são delegadas aos métodos estáticos da classe `GerenciadorIO`.

Quadro 12 – Trecho de código da implementação da interface `InfoSender`

```
@WebService(endpointInterface = "web_services.interfaces.InfoSender")
public class InfoSender_impl implements InfoSender {

    @Override
    public String sendIniciado(String nomeSistema) {
        return GerenciadorIO.criarXMLSistema(nomeSistema);
    }

    @Override
    public void sendEncerrado(String identificacaoSistema) {
        GerenciadorIO.removerXMLSistema(identificacaoSistema);
    }

}
```

Quando o sistema alvo inicia a execução, ele aciona o método `sendIniciado()` passando o nome do sistema. Que por sua vez solicita ao `GerenciadorIO` a criação de um documento XML para o sistema iniciado. O documento segue o formato descrito no Quadro 6 que contém o estado dos requisitos em avaliação. Inicialmente o estado de todos os requisitos é inserido como aguardando e conforme ocorre a avaliação seu estado é alterado. Após a criação do documento XML o método retorna uma identificação para o sistema, que é usada para identificar o sistema a cada evento acionado. A identificação foi necessária, pois, caso dois sistemas com nome semelhante sejam avaliados, não ocorra ambiguidade na avaliação dos requisitos. O método `sendEncerrado()` solicita ao `GerenciadorIO` que remova o documento referente ao sistema encerrado.

Quadro 13 – Trecho de código da implementação da interface `InfoGetter`

```
@WebService(endpointInterface = "web_services.interfaces.InfoGetter")
public class InfoGetter_impl implements InfoGetter {

    @Override
    public String[] getListaIDFamilias() {
        return GerenciadorIO.getListaNomeXML("xml_iso_15408");
    }

    @Override
    public String getXMLFamilia(String idFamilia) {
        return GerenciadorIO.getXMLComoTexto("xml_iso_15408\\" +
                                             idFamilia + ".xml");
    }

    @Override
    public String[] getIDSistemas() {
        return GerenciadorIO.getIDSistemas();
    }

    @Override
    public int getStatusRequisito(String identificacaoSistema, String idReq) {
        return GerenciadorIO.getStatusRequisitoXML(identificacaoSistema,
                                                   new Requisito(idReq));
    }

}
```

O método `getListaIDFamilias()` é usado em conjunto com o método `getXMLFamilia()`. Primeiramente o SPAARS solicita as identificações das famílias para que com elas possa recuperar o XML correspondente. Este procedimento é usado para atualizar os documentos mantidos pelo SPAARS caso novos requisitos sejam implementados.

O método `getListaIDSistemas()` retorna uma lista com a identificação dos sistemas que estão sob avaliação atualmente. É utilizado pelo SPAARS para selecionar o sistema que será avaliado. O método `getStatusRequisito()` é usado para consultar o estado atual de um requisitos. A identificação do sistema deve ser informa para que se possa acessar o documento correspondente ao sistema alvo.

O Quadro 14 mostra um dos métodos da classe `GerenciadorIO`.

Quadro 14 – Trecho de código da classe `GerenciadorIO`

```
public static String criarXMLSistema(String nomeSistema) {
    File xmlSistema;
    FileWriter fw;
    int id;
    Requisito[] listaReq;
    String identificacao;

    id = Integer.parseInt(getConfig(CONFIG_ULTIMO_ID));
    setConfig(CONFIG_ULTIMO_ID, Integer.toString(id + 1));
    identificacao = nomeSistema + "_" + id;
    xmlSistema = new File(PATH_DIR_CONFIGS + "\\" + identificacao + ".xml");

    try {
        fw = new FileWriter(xmlSistema);

        fw.write("<?xml version=\"1.0\" encoding=\"UTF-8\""
                + " standalone=\"no\"?>\n");
        fw.write("<sistema id=\"" + id + "\" nome=\"" + nomeSistema + "\">\n");
        listaReq = getRequisitos();
        for (Requisito r : listaReq) {
            fw.write("<requisito id=\"" + r.getId() + "\" status=\"" +
                    STATUS_REQ_AGUARDANDO + "\"></requisito>\n");
        }
        fw.write("</sistema>\n");
        fw.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    return identificacao;
}
```

No método `criarXMLSistema()` criado um arquivo XML com a identificação dos sistema que é representada pela variável `identificacao` e é formada pelo nome do sistema concatenado com o caractere '_' (*underscore*) e seguido de um número sequencial. O número inicia em zero e é incrementado a cada novo sistema que se conecta à plataforma. O arquivo é escrito usando um objeto `FileWriter` e escrito como um texto normal, porém seguindo o formato da linguagem XML.

3.4.6 Desenvolvimento do SPAARS

Para conectar-se à plataforma, o SPAARS necessita de uma instância de `InfoGetter`. O trecho de código correspondente a esta tarefa é apresentado no Quadro 15.

Quadro 15 – Trecho de código da inicialização da instância de `InfoGetter`

```
public static InfoGetter initInfoGetter() {
    URL url;
    QName qName;
    Service servico;
    InfoGetter ig;

    try {
        url = new URL("http://127.0.0.1:9876/info_getter?wsdl");
        qName = new QName("http://impl.web_services/",
                          "InfoGetter_implService");

        servico = Service.create(url, qName);

        ig = servico.getPort(InfoGetter.class);
    } catch (MalformedURLException ex) {
        ex.printStackTrace();
        ig = null;
    } catch (WebServiceException wsEx) {
        wsEx.printStackTrace();
        ig = null;
    }

    return ig;
}
```

Para criar uma instância de `InfoGetter` é necessário informar a URL do documento WSDS disponibilizado e o `QName` que é resultante do *target namespace* e do *name*, atributos descritos no documento WSDL. No final é retornada a referência do objeto instanciado.

Foi necessária a criação de um objeto `Thread` para tratar de forma concorrente a atualização dos estados dos requisitos na interface gráfica. A cada período de tempo são consultados os requisitos na plataforma. O Quadro 16 apresenta o trecho de código onde é criada a instância da `Thread`.

Quadro 16 – Trecho de código da instanciação da `Thread` para atualização dos estados

```
Thread atualizacao;
atualizacao = new Thread(new AtualizadorRequisitos(infoGetter));
atualizacao.start();
```

O construtor de `Thread` recebe um objeto `AtualizadorRequisitos` que implementa a interface `Runnable`, o código referente ao método `run()` é mostrado no Quadro 17. O construtor do objeto `AtualizadorRequisitos` recebe uma referência à um objeto `InfoGetter` usado para obter informações sobre os requisitos.

Quadro 17 – Trecho de código do método `run()` da classe `AtualizadorRequisito`

```
public void run() {
    int status;

    try {
        while (iniciado) {

            if (sistemaSelecionado != null) {

                for (int i = 0; i < listaIDReq.size(); i++) {
                    status = infoGetter.getStatusRequisito(sistemaSelecionado,
                                                           listaIDReq.get(i));
                    if (status != -1) {
                        if (status == -2) {
                            i = listaIDReq.size();
                            JOptionPane.showMessageDialog(TelaPrincipal.this,
                                                          "O sistema alvo disconectou!");
                            setIniciadoEncerrado(false);
                        }
                        else {
                            setStatus(listaIDReq.get(i), status);
                        }
                    }
                }
                Thread.sleep(1000);
            }
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
        setSistemaAlvo(null);
}
```

O método `run()` continua executado até que a variável `iniciado` seja alterada para `false`. O valor é alterado quanto o botão encerrar é pressionado na interface gráfica. Para cada identificador de requisito da `listaIDReq` é consultado seu estado através do método `getStatusRequisito()` do `InfoGetter`. Se o valor atribuído à variável `status` for `-1`, é sinal que as configurações na plataforma ainda não foram concluídas e nesta iteração deve ser ignorado. Se seu valor for `-2`, indica que o sistema alvo foi encerrado prematuramente, uma mensagem é exibida e a avaliação encerrada. Se o valor não for `-1` nem `-2`, o estado do requisito é atualizado. A cada iteração existe um *delay* de 1 segundo. O *delay* foi incluído para não sobrecarregar a máquina que hospeda a plataforma nem a máquina que executa o SPAARS. Caso o *delay* não fosse incluído, as iterações seriam executadas no menor tempo disponível, o que não é relevante, visto que para um usuário humano atualizações em intervalos de 1 segundo são consideravelmente rápidas.

3.4.7 Desenvolvimento dos requisitos para teste

Esta seção contém o desenvolvimento dos requisitos automatizados para o teste do modelo. Os requisitos automatizados são FIA_UAU.1.2, FIA_UAU.6.1, FDP_ACF.1.2, FTA_TAH.1.1 e FTA_TAH.1.2. Para cada requisito foi descrita sua definição, os eventos envolvidos e condições para a mudança de estados.

3.4.7.1 Requisito FIA_UAU.1.2 e FIA_UAU.6.1

O requisito FIA_UAU.1.2 define que, antes do usuário realizar qualquer outra função no sistema deve estar corretamente autenticado. (ALBUQUERQUE; RIBEIRO, 2002, p.136).

O requisito FIA_UAU.6.1 define que, em determinadas condições deve ser exigida a reautenticação do usuário. Um exemplo de condição seria após um tempo de inatividade a reautenticação é exigida. (ALBUQUERQUE; RIBEIRO, 2002, p.138).

Para avaliar estes dois requisitos foi criado um *web service* chamado `FIA_UAU`. Sua interface é apresentada no Quadro 18.

Quadro 18 – Interface web service `FIA_UAU`

```
@WebService
@SOAPBinding(style = Style.RPC)
public interface FIA_UAU {

    @WebMethod public void usuarioAutenticado(String idSistema);

    @WebMethod
    public void funcaoPosAutenticacaoIniciada(String idSistema,
                                              String nomeFuncao);

    @WebMethod public void autenticacaoIniciada(String idSistema);

}
```

Os métodos `usuarioAutenticado()`, `funcaoPosAutenticacaoIniciada()` e `autenticacaoIniciada()` são os eventos utilizados para avaliar os requisitos FIA_UAU.1.2 e FIA_UAU.6.1. Todos os métodos necessitam da identificação do sistema.

Para tornar a avaliação dos requisitos mencionados possível, é necessário que:

a) quando o usuário for autenticado no sistema alvo, o método `usuarioAutenticado()` seja chamado;

b) imediatamente após uma funcionalidade do sistema alvo ser iniciada, deve-se chamar o método `funcaoPosAutenticacaoIniciada()`. Por exemplo, caso o sistema possua um botão para cadastrar um produto qualquer. Após o clique neste

botão – que irá abrir, por exemplo, a tela de cadastro de produto – deve ser chamado o método;

c) quando a autenticação ou reautenticação seja iniciada o sistema alvo deve chamar o método `autenticacaoIniciada()`.

Condições para as transições de estado do requisito FIA_UAU.1.2, sendo que o estado inicial é `Aguardando`:

a) `Aguardando` para `Falha`: quando a avaliação é encerrada no estado `Aguardando`. Ou o evento `funcaoPosAutenticacao()` é acionado antes do evento `usuarioAutenticado()`;

b) `Aguardando` para `Parcialmente Aprovado`: quando o evento `funcaoPosAutenticacao()` é acionado após o evento `usuarioAutenticado()`;

c) `Parcialmente Aprovado` para `Falha`: quando, por exemplo, a reautenticação é exigida e antes da autenticação ser novamente feita é acionado o evento `funcaoPosAutenticacao()`;

d) `Parcialmente Aprovado` para `Aprovado`: a avaliação foi encerrada no estado `Parcialmente Aprovado`, portanto, até o fim da avaliação não ocorreu a condição que reprova o requisito.

No Quadro 19 é mostrado um trecho de código do método `funcaoPosAutenticacao()` onde é feita a mudança de estado do requisito FIA_UAU.1.2.

Quadro 19 – Trecho de código da analise do requisito FIA_UAU.1.2

```
// Analisando o requisito FIA_UAU.1.2
if (getStatus(idSistema, FIA_UAU_1_2) == STATUS_REQ_AGUARDANDO ||
    getStatus(idSistema, FIA_UAU_1_2) == STATUS_REQ_PARCIALMENTE_OK) {
    if (usuarioAutenticado) {
        setStatus(idSistema, FIA_UAU_1_2, STATUS_REQ_PARCIALMENTE_OK);
    }
    else {
        if (!reautenticacao) {
            setStatus(idSistema, FIA_UAU_1_2, STATUS_REQ_FALHA);
        }
    }
}
```

A variável `usuarioAutenticado` é do tipo `boolean` e indica que o usuário está devidamente autenticado. A variável `reautenticacao` também do tipo `boolean`, indica que a reautenticação já está sendo exigida.

Condições para as transições de estado do requisito FIA_UAU.6.1, sendo que o estado inicial é `Aguardando`:

a) `Aguardando` para `Falha`: quando a avaliação é encerrada no estado `Aguardando`. Ou o tempo de inatividade ultrapassa o limite de tempo para a reautenticação. O evento `funçãoPosAutenticacao()` é acionado e ainda não foi acionado o evento `autenticacaoIniciada()`. Isto caracteriza que, após o tempo de inatividade uma funcionalidade do sistema foi acionada e a reautenticação não foi exigida;

b) `Aguardando` para `Parcialmente Aprovado`: quando o evento `autenticacaoIniciada()` é acionado após o tempo de inatividade ultrapassar o limite estipulado;

c) `Parcialmente Aprovado` para `Falha`: quando a reautenticação não é exigida após o limite de tempo;

d) `Parcialmente Aprovado` para `Aprovado`: quando a avaliação é encerrada no estado `Parcialmente Aprovado`.

No Quadro 20 é mostrado um trecho de código do método `funcaoPosAutenticacao()` que mostra como é feita a mudança de estados do requisito FIA_UAU.6.1.

Quadro 20 – Trecho de código da analise do requisito FIA_UAU.6.1

```
// Analisando o requisito FIA_UAU.6.1
if ((getStatus(idSistema, FIA_UAU_6_1) == STATUS_REQ_AGUARDANDO ||
     getStatus(idSistema, FIA_UAU_6_1) == STATUS_REQ_PARCIALMENTE_OK)) {

    long intervaloTempo;

    intervaloTempo = Timer.getInstance().getIntervaloTempo();
    if (intervaloTempo > limiteTempo) {
        if (reautenticacao) {
            setStatus(idSistema, FIA_UAU_6_1, STATUS_REQ_PARCIALMENTE_OK);
        }
        else {
            setStatus(idSistema, FIA_UAU_6_1, STATUS_REQ_FALHA);
        }
    }
    else {
         if (reautenticacao) {
            setStatus(idSistema, FIA_UAU_6_1, STATUS_REQ_PARCIALMENTE_OK);
        }
    }

}
```

A classe `Timer` é usada para marcar o tempo passado, obtendo assim o intervalo de tempo entre a execução das funcionalidades do sistema alvo. O valor da variável `limiteTempo` é obtida através do documento XML da família FIA_UAU.

3.4.7.2 Requisito FDP_ACF.1.2

O requisito FDP_ACF.1.2 define que, deve-se garantir a política de controle de acesso com base nos níveis de confidencialidade da informação e de acesso do usuário. (ALBUQUERQUE; RIBEIRO, 2002, p.71).

Para a avaliação deste requisito foi definido que para cada perfil de usuário existe uma política de controle de acesso. Um perfil pode ter permissão para escrita, leitura ou ambos. O Quadro 21 apresenta a interface *web service* da família FDP_ACF utilizada para avaliar o requisito.

Quadro 21 – Interface *web service* `FDP_ACF`

```
@WebService
@SOAPBinding(style = Style.RPC)
public interface FDP_ACF {

    /* FDP_ACF.1.2 */
    @WebMethod public void acessoEfetuado(String idSistema,
                                          String nomeUsuario,
                                          String perfil,
                                          char tipoAcesso);

}
```

O método `acessoEfetuado()` é utilizado para avaliar o requisito FDP_ACF.1.2, ele recebe como parâmetro a identificação do sistema, nome do usuário, perfil do usuário e o tipo de acesso realizado. Para tornar a avaliação do requisito possível é necessário que, ao ser efetuado um acesso a um dado do sistema, seja chamado o método `acessoEfetuado()` informando o tipo de acesso.

Condições para a mudança de estados do requisito FDP_ACF.1.2, sendo que o estado inicial é `Aguardando`:

a) `Aguardando` para `Falha`: quando a avaliação é encerrada no estado `Aguardando`. Ou um tipo de acesso foi efetuado e o usuário não tem permissão;

b) `Aguardando` para `Parcialmente Aprovado`: um acesso é efetuado e o usuário tem permissão para o tipo de acesso;

c) `Aguardando` para `Aprovado`: todos as combinações entre tipo de perfil e tipo de acesso forem efetuados, sendo que somente os acesso permitidos foram efetuados;

d) `Parcialmente Aprovado` para `Falha`: um tipo de acesso foi efetuado e o usuário não tem permissão;

e) `Parcialmente Aprovado` para `Aprovado`: mudança de estado idêntica à do item c).

No Quadro 22 é exibido um trecho de código do método `acessoEfetuado()`, nele é feita a análise do requisito FDP_ACF.1.2.

Quadro 22 – Trecho de código da análise do requisito FDP_ACF.1.2

```
@Override
public void acessoEfetuado(String idSistema, String nomeUsuario,
                           String perfil, char tipoAcesso) {

    boolean permitido;

    // Analisa o requisito FDP_ACF.1.2
    if (getStatus(idSistema, FDP_ACF_1_2) == STATUS_REQ_AGUARDANDO ||
        getStatus(idSistema, FDP_ACF_1_2) == STATUS_REQ_PARCIALMENTE_OK) {

        permitido = permissaoCorreta(tipoAcesso, perfil);

        if (permitido) {
            setStatus(idSistema, FDP_ACF_1_2, STATUS_REQ_PARCIALMENTE_OK);
        }
        else {
            setStatus(idSistema, FDP_ACF_1_2, STATUS_REQ_FALHA);
        }

        if (getStatus(idSistema, FDP_ACF_1_2) == STATUS_REQ_AGUARDANDO ||
            getStatus(idSistema, FDP_ACF_1_2) == STATUS_REQ_PARCIALMENTE_OK) {
            if (todosForamTestados()) {
                setStatus(idSistema, FDP_ACF_1_2, STATUS_REQ_OK);
            }
        }

    }
}
```

A variável `permitido` recebe o valor resultante da evocação do método `permissaoCorreta()`. Este método compara o tipo de acesso – parâmetro do tipo `char` – que contém o valor correspondente à leitura ou gravação.

3.4.7.3 Requisito FTA_TAH.1.1 e FTA_TAH.1.2

O requisito FTA_TAH.1.1 define que, ao estabelecer uma seção deve ser armazenado um histórico, que deve conter um conjunto de dados do último estabelecimento de seção para

aquele usuário. Um exemplo de dados seria a data e a hora do estabelecimento da seção. (ALBUQUERQUE; RIBEIRO, 2002, p.151).

O requisito FTA_TAH.1.2 define que, ao estabelecer uma seção deve ser armazenado um histórico, que deve conter um conjunto de dados da última falha no estabelecimento de seção para aquele usuário. Além dos dados presentes no requisito FTA_TAH.1.1 o histórico deve conter uma conta da quantidade de falhas ocorridas. (ALBUQUERQUE; RIBEIRO, 2002, p.151).

Para a avaliação destes requisitos foi criada a interface web service FTA_TAH. O Quadro 23 apresenta o código do a interface.

Quadro 23 – Interface *web service* `FTA_TAH`

```
@WebService
@SOAPBinding(style = Style.RPC)
public interface FTA_TAH {

    /* FTA_TAH.1.1 */
    @WebMethod public void historicoPersistido(String idSistema,
            String nomeUsuario, String regHistorico);

    /* FTA_TAH.1.2 */
    @WebMethod public void historicoPersistidoFalha(String idSistema,
            String nomeUsuario, String regHistorico);

}
```

O método `historicoPersistido()` é utilizado para avaliar o requisito FTA_TAH.1.1 e recebe como parâmetro a identificação do sistema, nome do usuário e o texto que foi persistido no histórico. O método `historicoPersistidoFalha()` é utilizado para avaliar o requisito FTA_TAH.1.2, os parâmetros são idênticos ao método `historicoPersistido()`, porém a forma de avaliação é distinta.

Para a verificação dos campos forma utilizadas expressões regulares. Através do método `matches()` da classe `String` é possível verificar se uma instância de `String` corresponde à expressão regular. Foram definidas expressões regulares para a data, hora e o contador de falhas. As expressões regulares são exibidas no Quadro 24.

Quadro 24 – Expressões regulares para o formato do histórico

Expressões regulares	
Data	([0-2][1-9]|[1-3][0]|31) ('/'|'-'|'.') (0[1-9]|1[0-2]) ('/'|'-'|'.') ([1-9][0-9])$^{?}$[0-9][0-9]
Hora	([0-2][0-3]|[0-1][0-9]) (':') ([0-5][0-9]) (':'[0-5][0-9])$^{?}$

Contador	$0^{*}[0\text{-}9]^{(1,6)}$

Condições para mudança de estados do requisito FTA_TAH.1.1, sendo que o estado inicial é `Aguardando`:

a) `Aguardando` para `Falha`: quando texto que corresponde ao histórico não contém todos os dados exigidos. Ou a avaliação é encerrada no estado `Aguardando`;

b) `Aguardando` para `Parcialmente Aprovado`: quando todos os dados exigidos estão inclusos no texto correspondente ao histórico;

c) `Parcialmente Aprovado` para `Falha`: quando algum dado não está incluso;

d) `Parcialmente Aprovado` para `Aprovado`: quando a avaliação é encerrada no estado `Parcialmente Aprovado`.

No Quadro 25 é apresentado o trecho de código que analisa o requisito FTA_TAH.1.1.

Quadro 25 – Trecho de código da análise do requisito FTA_TAH.1.1

```
@Override
public void historicoPersistido(String idSistema, String nomeUsuario,
                                String regHistorico) {
    if (getStatus(idSistema, FTA_TAH_1_1) == STATUS_REQ_AGUARDANDO ||
        getStatus(idSistema, FTA_TAH_1_1) == STATUS_REQ_PARCIALMENTE_OK) {
        String erHistorico;

        erHistorico = "(.)*" + ER_DATA + "(.)*" + ER_HORA + "(.)*|" +
                      "(.)*" + ER_HORA + "(.)*" + ER_DATA + "(.)*";

        if (regHistorico.matches(erHistorico)) {
            setStatus(idSistema, FTA_TAH_1_1, STATUS_REQ_PARCIALMENTE_OK);
        }
        else {
            setStatus(idSistema, FTA_TAH_1_1, STATUS_REQ_FALHA);
        }
    }

}
```

Inicialmente é construída uma nova expressão regular, resultante da combinação das expressões regulares. A expressão regular é atribuída à variável `erHistorico` e define que, a data pode ser precedida ou sucedida de qualquer sequência de caracteres seguido pela hora sucedida por qualquer sequência de caracteres. Ou ainda é válido o sentido inverso, sendo a hora por primeiro e a data na sequência. Com base no valor booleano resultante do método `matches()` é feita a mudança de estado.

As condições para a mudança de estados do requisito FTA_TAH.1.2 são idênticas ao do requisito FTA_TAH.1.1. O trecho de código que faz a análise do requisito é apresentado no Quadro 26.

Quadro 26 – Trecho de código da análise do requisito FTA_TAH.1.2

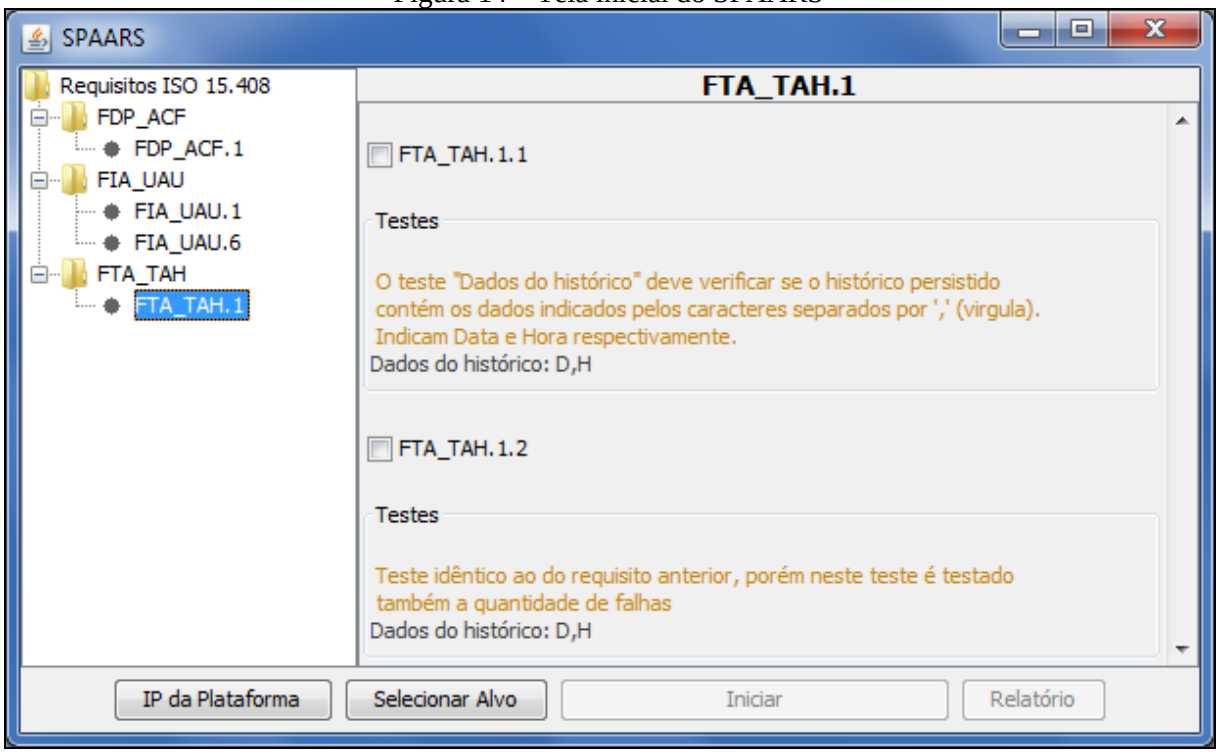
```
// Verifica se contém o contador (da quantidade falhas)
for (String s : tokensResultantes) {
    if (s.matches("(.)*" + ER_CONTADOR + "(.)*")) {
        contadorOK = true;
    }
}

if (dataOK && horaOK && contadorOK) {
    setStatus(idSistema, FTA_TAH_1_2, STATUS_REQ_PARCIALMENTE_OK);
}
else {
    setStatus(idSistema, FTA_TAH_1_2, STATUS_REQ_FALHA);
}
```

Inicialmente é feita uma busca para encontrar os fragmentos do texto do histórico que corresponde à data e hora. Estes fragmentos são removidos e as `String`s restantes são inseridas na lista `tokensResultantes`. Este procedimento foi necessário, pois devido à expressão regular definida para a data e hora, é possível que o contador seja confundido com, por exemplo, o dia da data ou os segundos da hora.

3.4.8 Operacionalidade da implementação

Nesta seção é apresentada a operacionalidade do SPAARS. O sistema utilizado para demonstrar a avaliação foi um sistema básico desenvolvido apenas para fins de teste. Este sistema contém algumas funções de segurança relevantes para a avaliação. Inicialmente executa-se o SPAARS, sua tela principal é mostrada na Figura 14.

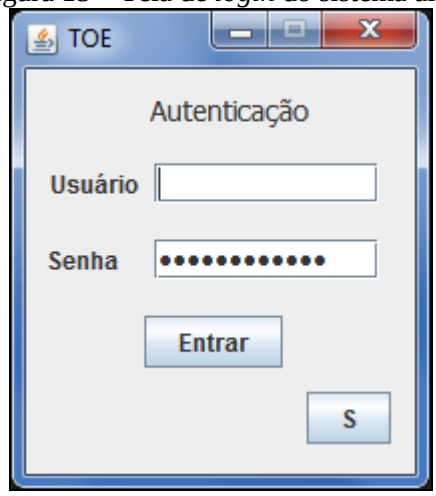
Figura 14 – Tela inicial do SPAARS



A tela contém uma árvore à esquerda. Nesta árvore é possível selecionar uma subfamília. À direita existe um painel onde são exibidos os requisitos da subfamília selecionada. O painel possui *checkboxes* para a seleção dos requisitos e um subpainel que exibe os parâmetros dos testes realizados para a avaliação do requisito. Os parâmetros podem ser alterados no documento XML correspondente à família do requisito. A parte inferior da tela contém quatro botões o botão `IP da Plataforma`, `Selecionar Alvo`, `Iniciar` e `Relatório`.

Na sequência é iniciado o sistema alvo. Sua tela inicial é a tela de *login* apresentada na Figura 15.

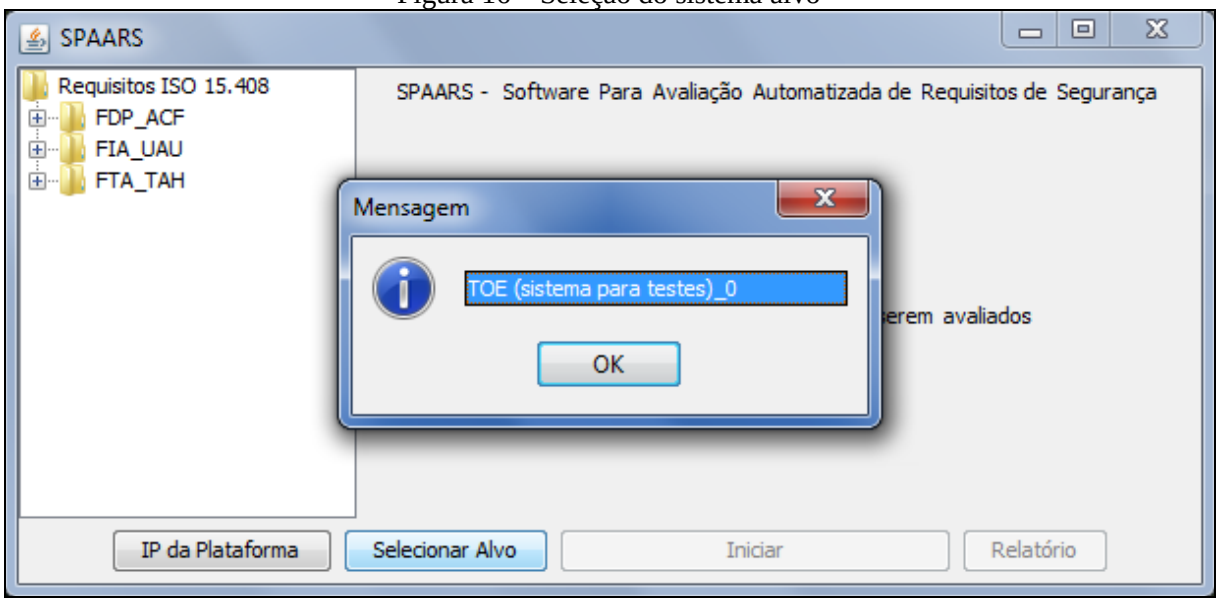
Figura 15 – Tela de *login* do sistema alvo



Nas principais telas do sistema alvo existe um botão `S` que é referente à segurança. Ao clica-lo as funções de segurança do sistema alvo são desabilitadas. Isto foi criado para mostrar as condições de falha durante a avaliação.

Em seguida é selecionado o sistema alvo ao clicar no botão `Selecionar Alvo`. A Figura 16 mostra a tela de seleção.

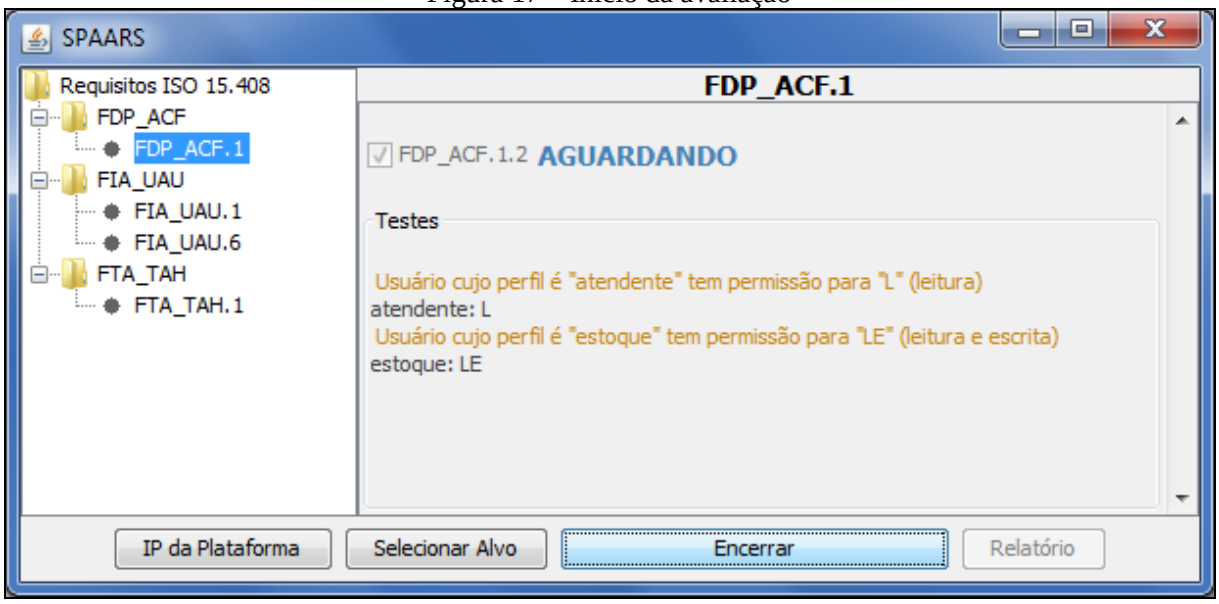
Figura 16 – Seleção do sistema alvo



Nesta janela é possível selecionar o sistema alvo. Ao clicar no botão `OK` o sistema alvo é selecionado e o botão `Iniciar` é habilitado.

O próximo passo é a seleção dos requisitos que é feita através dos *checkboxes*. Em um primeiro momento é mostrada a avaliação do requisito FDP_ACF.1.2 com sucesso. Em um segundo momento é mostrada a avaliação com falha, que é provocada através da desativação da segurança do sistema alvo. A Figura 17 mostra o início da avaliação.

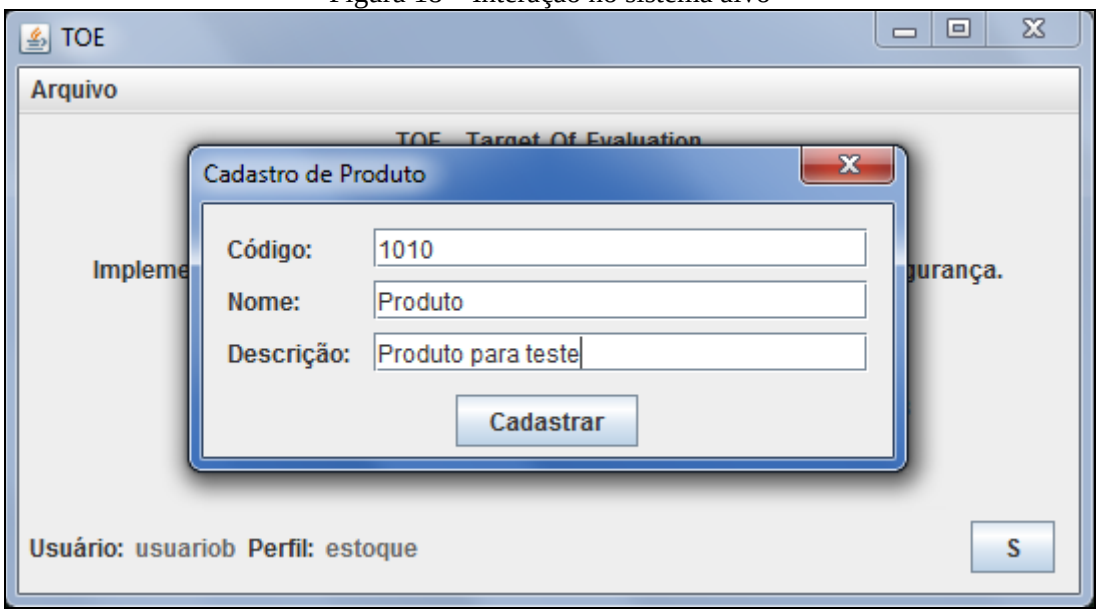
Figura 17 – Início da avaliação



Ao clicar no botão `Iniciar` é iniciada a avaliação e o estado atual do requisito é exibido.

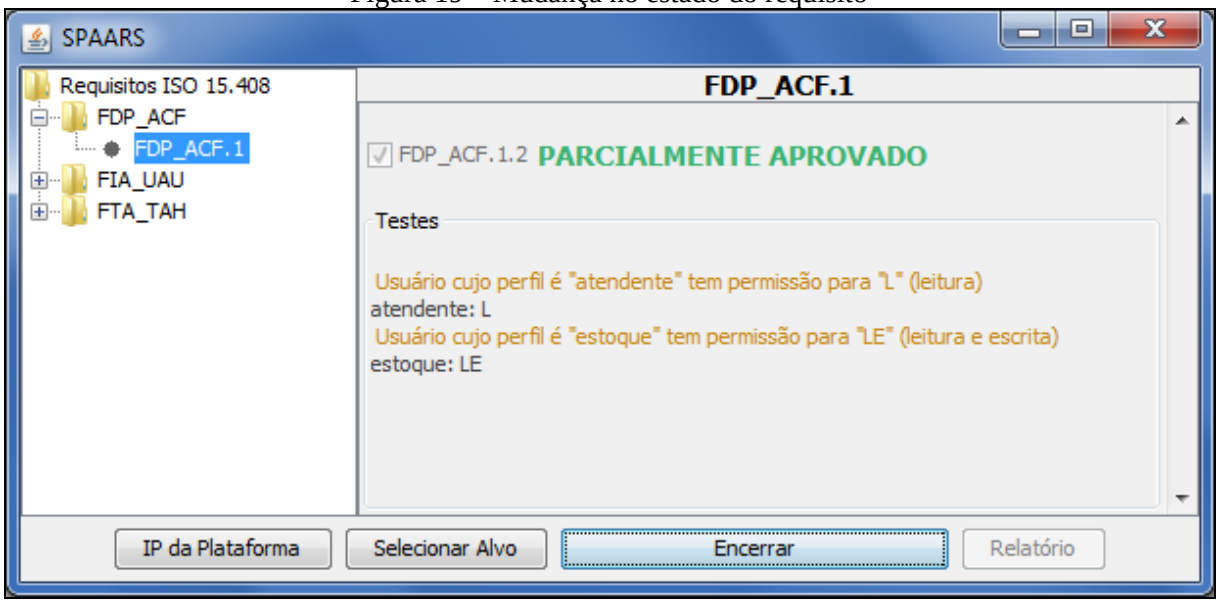
São feitas então interações no sistema alvo. É feita uma tentativa de cadastro de produto por um usuário com o perfil de `estoque`, sendo que para este perfil esta operação é permitida. A Figura 18 mostra a tela de cadastro de produto.

Figura 18 – Interação no sistema alvo



Ao efetuar a operação o estado do requisito é alterado para `Parcialmente Aprovado`. A Figura 19 mostra o novo estado do requisito.

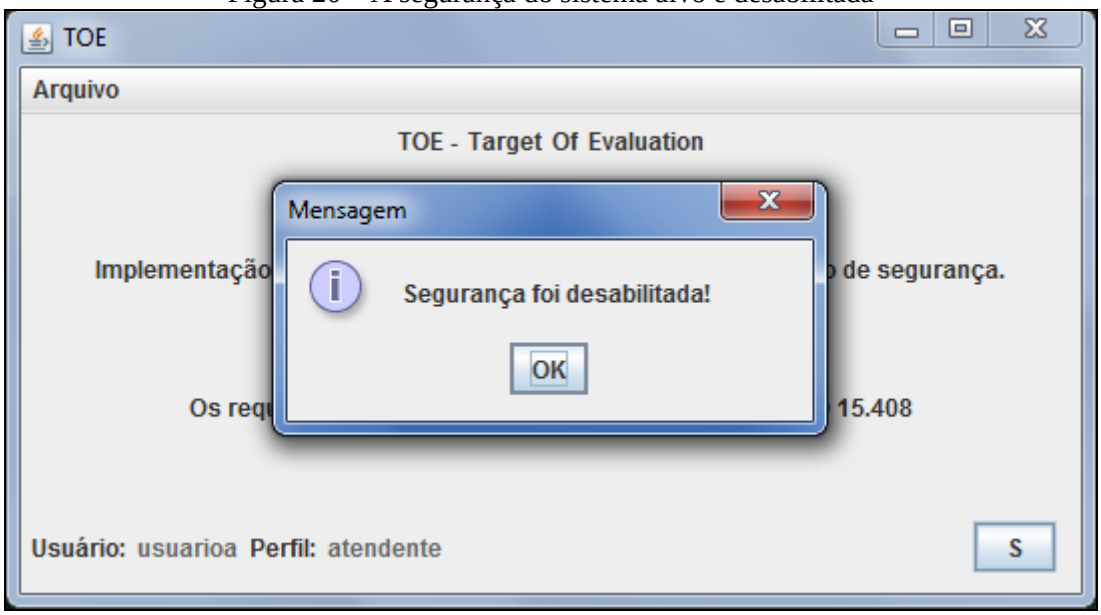
Figura 19 – Mudança no estado do requisito



Esta alteração ocorreu porque foi executado um tipo de acesso permitido a aquele perfil de usuário, ou seja, foi feita uma operação de escrita. Os estados são alterados conforme a especificação na seção 3.4.7.

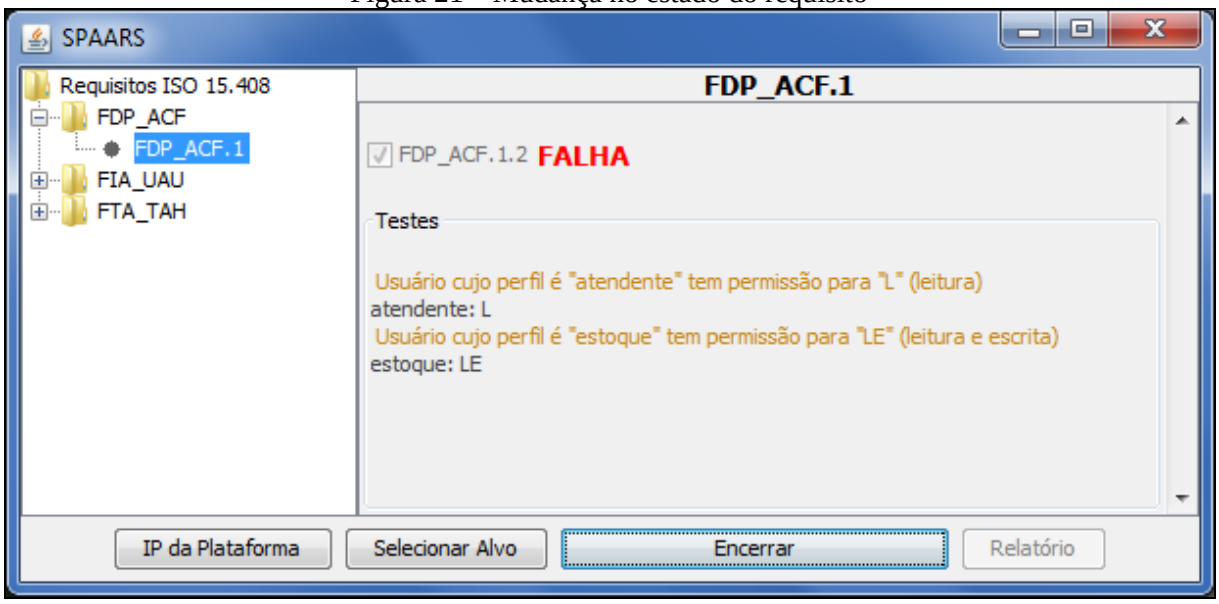
Ao clicar no botão `S` do sistema alvo, pode ser simulada uma ocasião de falha de segurança. A Figura 20 apresenta a mensagem após o clique no botão `S`.

Figura 20 – A segurança do sistema alvo é desabilitada



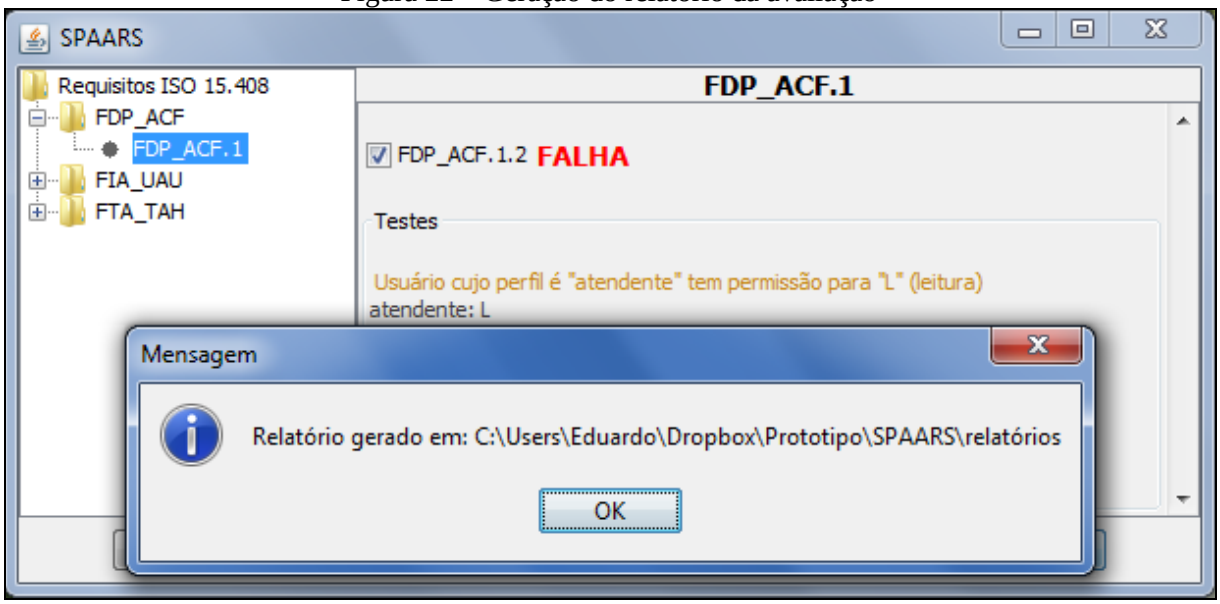
Neste momento é feita uma tentativa de cadastro de produto com o perfil de atendente. O estado é então alterado para `Falha`, pois o atendente não tem permissão para escrita. O estado alterado é mostrado na Figura 21.

Figura 21 – Mudança no estado do requisito



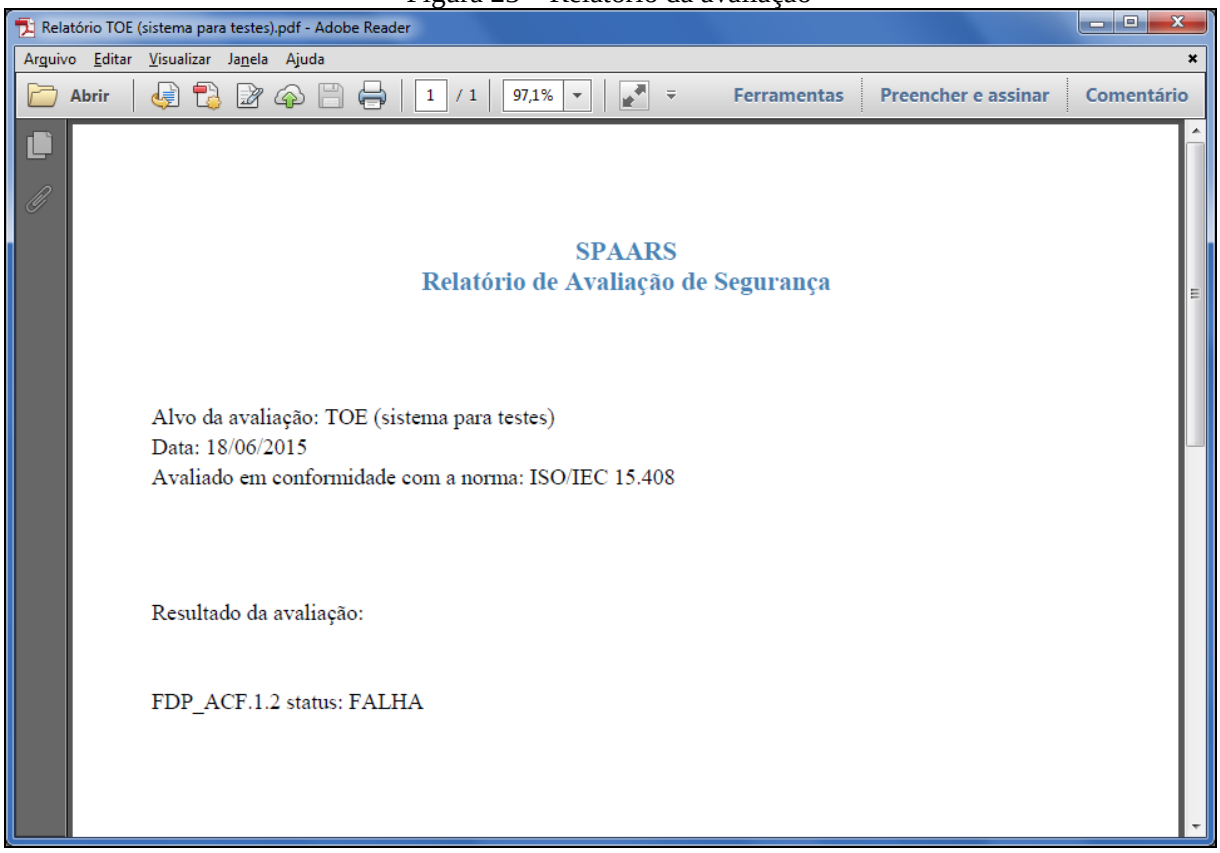
O botão `Encerrar` pode ser clicado a qualquer momento da avaliação. Ao clica-lo é habilitado o botão `Relatório`. Este botão é responsável por gerar o relatório resultante da última avaliação. A Figura 22 mostra a janela após o clique no botão `Relatório`.

Figura 22 – Geração do relatório da avaliação



Ao clicar no botão `Relatório` é exibida uma janela que contém o diretório no qual foi criado o arquivo do relatório. O relatório é criado como um arquivo PDF e é nomeado com o nome do sistema alvo. Portanto, caso a avaliação seja refeita e um relatório for gerado, o primeiro relatório será sobrescrito pelo novo relatório. Cabe ao auditor gerenciar os arquivos gerados. A Figura 23 mostra o arquivo PDF do relatório gerado.

Figura 23 – Relatório da avaliação



O relatório contém no cabeçalho o nome do sistema alvo, a data da avaliação e a norma usada como base para a avaliação. A norma não é selecionável, portanto este campo é fixo. O corpo contém os requisitos que foram selecionados pelo auditor e seus estados.

3.5 RESULTADOS E DISCUSSÕES

Com o término deste trabalho foi possível especificar um modelo para a avaliação automatizada de segurança em sistemas com base na norma ISO/IEC 15.408 e utilizando *Web Services*. Verificou-se que, utilizando o modelo foi possível testar seu funcionamento através da implementação de requisitos automatizados e desenvolvendo um software para avaliação automatizada de requisitos de segurança.

Com o desenvolvimento da plataforma, tornou-se possível a criação de novos requisitos automatizados além dos desenvolvidos neste trabalho. Outro ponto importante foi a reutilização dos requisitos automatizados, que independem do sistema alvo e do SPAARS. Desta forma é possível que qualquer tipo de sistema que necessite passar pela avaliação de segurança utilize os requisitos já desenvolvidos, não necessitando de adaptações. É possível ainda que, se necessário, sejam feitas mudanças na implementação dos requisitos automatizados. Para as mudanças não é necessário que se altere o sistema alvo e nem o SPAARS, contanto que se mantenha a assinatura dos eventos.

A principal dificuldade encontrada foi como criar uma forma para implementar de forma automatizada a avaliação dos critérios estabelecidos pela norma, visto que a norma foi projetada para ser utilizada por um auditor humano. A saída para este problema foi a criação dos eventos, que são chamados em ocasiões durante a execução do sistema alvo. Portanto, para que se torne possível a avaliação, é necessário que o sistema alvo siga o modelo, acionando os eventos nas ocasiões corretas. Outra dificuldade encontrada foi a escalabilidade da plataforma. Como pode ser usada por vários softwares de avaliação de segurança simultaneamente, foi necessário prover uma forma de comunicação entre a plataforma e vários sistemas alvo. Isto foi resolvido com a atribuição de uma identificação para os sistemas alvo. Desta forma, ao acionar um evento, a identificação do sistema deve ser passada como parâmetro. Pela plataforma, a identificação é usada para manipular o arquivo correspondente ao sistema que acionou o evento.

Também foram identificadas algumas limitações no modelo proposto. A principal delas é a possibilidade de uma sequência de acionamento de eventos maliciosa, a fim de ludibriar a avaliação dos requisitos. Outra limitação é a necessidade de que sejam feitas interações no sistema alvo, para tornar a avaliação possível.

Em relação aos trabalhos correlatos, o IBM Rational AppScan realiza testes em caixa branca e caixa preta. O trabalho desenvolvido realiza os testes com base nas ações ocorridas durante a execução do sistema alvo. Pode-se destacar ainda que o IBM Rational AppScan não realiza os testes com base em normas de segurança.

O “software para verificação de conformidade de sistemas à norma ISO/IEC 15408” foi uma ferramenta desenvolvida para a avaliação de sistemas escritos na linguagem Java. Para evitar a restrição de linguagens, o trabalho desenvolvido realiza a avaliação de sistemas usando *web services* para a comunicação, portanto sistemas escritos em qualquer linguagem que suporte *web services* pode ser avaliado. Outra diferença é quanto à forma de avaliação de segurança, que é feita através da análise do código fonte Java do sistema alvo, enquanto a forma como é avaliada a segurança neste trabalho é por meio de eventos.

Os “*plugins* para teste semi-automatizados de conformidade com a norma ISO/IEC 15408” foi uma extensão do “software para verificação de conformidade de sistemas à norma ISO/IEC 15408”. A principal diferença em comparação com o trabalho desenvolvido são os testes automatizados. Pela forma como é feita a avaliação de segurança neste trabalho, tornou-se possível a avaliação de requisitos de segurança como o FIA_UAU.6.1. A avaliação deste requisito é feita em tempo de execução, pois é necessário verificar o tempo limite para a reautenticação.

4 CONCLUSÕES

O crescimento na utilização e criação de programas é inevitável, devido à grande demanda nesta área. Como existem programas para diversas finalidades, é natural que existam áreas onde a segurança é crucial. É importante que, um sistema que deve ser considerado seguro, seja seguro de fato. Para garantir a segurança é aplicada a auditoria de TI, juntamente de critérios para a avaliação da segurança.

O processo de auditoria de TI é complexo. Existem tarefas que devem ser realizadas através da interação com o sistema alvo e desta forma é avaliado pelo auditor. Outro ponto que aumenta a complexidade da avaliação é a heterogeneidade dos sistemas, que, atualmente são criados para as mais diversas áreas. Para contornar estes problemas é que são usadas normas, como o CC.

Com a especificação do modelo de avaliação automatizada de requisitos de segurança em sistemas conforme a norma ISO/IEC 15.408, foi possível criar uma interface entre sistema alvo e o software que realiza a avaliação automatizada de segurança. Esta interface torna o SPAARS independente do sistema alvo e foi implementada através da chamada de métodos *web service* em momentos específicos da execução do sistema alvo. Foi desenvolvido um software que faz a avaliação automatizada de segurança em sistemas. Foi realizada a analise do modelo especificado automatizando alguns requisitos da norma e implementando um sistema alvo simples.

No requisito FIA_UAU.1.1 foi desenvolvida uma forma de verificar se um usuário acessa as funções do sistema após estar autenticado. No requisito FIA_UAU.6.1 foi analisado o tempo limite em que a reautenticação é exigida. No requisito FDP_ACF.1.2 foi implementada uma verificação, com base nas permissões de cada perfil de usuário para o acesso ao sistema alvo. No requisito FTA_TAH.1.1 foi realizada a verificação dos dados que são armazenados no histórico da seção estabelecida para os usuários, que deve conter a data e a hora. No requisito FTA_TAH.1.2 foi realizada a verificação do histórico de falhas no estabelecimento da seção para os usuários, no qual deve conter além dos campos data e hora, um campo que conta a quantidade de falhas.

Com este trabalho foi concluído que é possível automatizar requisitos usando o modelo especificado e que os requisitos implementados podem ser reaproveitados para diversos sistemas alvo. É possível automatizar requisitos que exigem que testes sejam feitos em tempo de execução. Concluiu-se ainda que pode-se implementar novos requisitos além dos desenvolvidos neste trabalho.

4.1 EXTENSÕES

As extensões para este trabalho são:

a) implementar novos requisitos automatizados;
b) portar o SPAARS para a web;
c) incluir mais funcionalidades ao SPAARS, como o cadastro de planos de auditoria ou cadastro de perfis de segurança.

REFERÊNCIAS

ALBUQUERQUE, Ricardo; RIBEIRO, Bruno. **Segurança no desenvolvimento de software**. Rio de Janeiro: Campus, 2002. 310 p.

ALLEN, Julia H. et al. **Software security engineering**: a guide for project managers. Upper Saddle River, NJ: Addison-Wesley, 2008. 334 p.

CHAPPELL, David A.; JEWELL, Tyler. **Java Web Services**. Sebastopol: O'Reilly, 2002.

DIAS, Claudia. **Segurança e auditoria da tecnologia da informação**. Rio de Janeiro: Axcel Books, 2000. 218 p.

GRAHAM, Steve et al. **Building Web Services with Java**: Making Sense of XML, SOAP, WSDL, and UDDI. Indianapolis: Sams, 2002.

GRONER, Loiane. **Manipulando arquivos XML em Java com a API DOM**. 2009. Disponível em: < http://www.loiane.com/2009/04/manipulando-arquivos-xml-em-java-com-a-api-dom-parte-i/>. Acesso em: 19 maio 2015.

HERKENHOFF, Dionei. **Plugins para testes semi – automatizados de conformidade com a norma ISO/IEC 15408**. 2011. 60 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

HOWARD, Michael; LEBLANC, David. **Escrevendo código seguro**: estratégias e técnicas práticas para codificação segura de aplicativos em um mundo de rede. Porto Alegre: Bookman, 2005. 701 p.

IBM. 2010. Disponível em: < http://www.ibm.com>. Acesso em: 15 set. 2014.

ISO. **Internetional Organization for Standarization**. 2009. Disponível em: <http://www.iso.org>. Acesso em: 14 set. 2014.

MEDEIROS, Higor. **Introdução ao iText**. 2013. Disponível em: <http://www.devmedia.com.br/introducao-ao-itext/29864>. Acesso em: 20 maio 2015.

OLIVEIRA, Wilson J. **Segurança da informação:** técnicas e soluções. Florianópolis: Visual Books, 2001. 182 p.

TRAPP, Daiana Fernanda. **Software para verificação de conformidade de sistemas à norma ISO/IEC 15408**. 2010. 66 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

WAGNER, Rosana. **Processos de desenvolvimento de software confiáveis baseados em padrões de segurança**. 2011. 107 f. Dissertação de mestrado em informática (Mestre em Informatica) – Centro de Tecnologia, Universidade Federal de Santa Maria, Santa Maria.

W3C. **SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)**. 2004a. Disponível em: <http://www.w3.org/TR/ws-arch/>. Acesso em: 30 mar. 2015.

W3C. **Web Services Architecture**. 2004b. Disponível em: <http://www.w3.org/TR/ws-arch/>. Acesso em: 30 mar. 2015.