UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GUILHERME SELAU RIBEIRO

Delphos – Uma Ferramenta de Análise de Segurança

Trabalho de Graduação.

Prof. Dr. Raul Fernando Weber
Orientador

Porto Alegre, novembro de 2011.

AGRADECIMENTOS

Agradeço à minha mãe por me ensinar o valor da educação.

Agradeço à minha mãe e à minha irmã pela paciência e compreensão durante toda a graduação.

Agradeço ao professor Weber pela paciência e dedicação durante o curso e durante a realização desse trabalho.

Agradeço a todos os professores do Instituto de Informática da UFRGS que estimulam a busca pelo conhecimento.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

SAVE	Software Architecture Visualization and Evaluation
SOA	Service Oriented Architecture
SCA	Service Component Architecture
GMF	Graphical Modeling Framework
EMF	Eclipse Modeling Framework
Java EE	Java Enterprise Edition
SiSOA	Sicherheit in Service-Orientierten Architekturen
JDT	Java Development Tools
SGIT	Security Goal Indicator Tree
ECC	Error Correction Code

LISTA DE FIGURAS

RESUMO

Delphos é um método de engenharia reversa que visa analisar aspectos de segurança de sistemas a partir do seu código fonte. Ele é baseado no método SiSOA desenvolvido no Fraunhofer Institute na Alemanha. A diferença entre as duas ferramentas é que enquanto a segunda analisa sistemas desenvolvidos com arquitetura SCA, a primeira tem como alvo sistemas desenvolvidos com arquitetura Java EE.

Esse método possui três fases principais: fase de extração, fase de identificação e fase de análise. Na primeira fase é extraído um modelo global do sistema com base na análise automática do seu código fonte e de arquivos de configuração. Na fase de identificação é feito uso de uma base de conhecimento em segurança, previamente criada por um especialista humano usando a arquitetura SGIT. Essa base é automaticamente avaliada sobre o modelo atribuindo tags de identificação aos artefatos do modelo conforme o algoritmo de identificação encontra os padrões. Na última fase são exibidos ao analista o modelo com as tags identificadas e relatórios de severidade e credibilidade desses artefatos. A partir dessa saída o analista pode atribuir manualmente tags ou alterar a base de conhecimento para refletir as características desejadas, logo após é possível executar novamente o algoritmo de identificação para que o modelo reflita as alterações.

O processo de análise de segurança através do Delphos é interativo, ou seja, o analista de segurança pode reutilizar e atualizar bases de conhecimento. Da mesma forma é possível executar o algoritmo de identificação repetidas vezes sobre o mesmo modelo até que o grau de detalhamento e cobertura desejados sejam atingidos.

Palavras-Chave: engenharia reversa, análise de segurança, base de conhecimento.

Delphos: A Tool for Security Analysis

ABSTRACT

Delphos is a method of reverse engineering which aims to analyze the security aspects of systems from their source code. It is based on SiSOA method developed at the Fraunhofer Institute in Germany. The difference between the two tools is that while the second examines systems developed with SCA, the first targets systems developed with Java EE architecture.

This method has three main stages: extraction phase, identification phase and analysis phase. In the first phase is extracted a global model of the system based on automatic analysis of its source code and configuration files. In the identification phase is used a security knowledge base previously created by a human expert using the SGIT architecture. This knowledge base is automatically evaluated over the model by assigning identification tags to artifacts of the model as the identification algorithm meets the standards. In the last phase are displayed to the analyst the model identified with tags and reports of severity and credibility of these artifacts. From this output the analyst can manually assign tags or change the knowledge base to reflect the desired characteristics, it is possible after this process re-run the identification algorithm to update the model to reflect the changes.

The security analysis process is interactive through Delphos, a security analyst can reuse and update knowledge bases. Similarly it is possible to run the identification algorithm repeatedly on the same model until the level of detail and coverage desired are achieved.

Keywords: reverse engineering, security analysis, knowledge base.

1 INTRODUÇÃO

Este trabalho tem como objetivo descrever uma ferramenta de engenharia de software voltada para a análise estática de aspectos de segurança em grandes sistemas corporativos.

Segurança é um conceito amplo que envolve diversos aspectos relacionados a pessoas, a comportamentos e a sistemas de computação. Esse trabalho visa analisar características técnicas de sistemas de grande porte que sofrem mudanças contínuas, relacionadas à manutenção, à expansão ou à mudança de requisitos. Manter o registro de todas essas alterações é uma tarefa dispendiosa e complicada. Efeitos colaterais de mudanças em sistemas computacionais são muitas vezes difíceis de prever.

Delphos é uma ferramenta de engenharia reversa que se baseia nas informações mais consistentes possíveis, as extraídas do código fonte e dos arquivos de configuração. Baseado nessas informações são feitas buscas por padrões pré-definidos por um especialista em segurança, de maneira automática e interativa.

Delphos é baseado na metodologia SiSOA (Antonino, 2010) de análise estática de software utilizando uma base de conhecimento, criada por um especialista em segurança. Essa metodologia foi desenvolvida no Fraunhofer Institute for Experimental Software Engineering em Kaiserslautern, na Alemanha, durante um intercâmbio que participei com a Universidade de Kaiserslautern durante o ano de 2010. Nesse estágio no Fraunhofer participei do desenvolvimento da ferramenta SiSOA, uma implementação da metodologia de mesmo nome. Essa ferramenta foi criada para analisar sistemas construídos com a arquitetura SCA. Baseado na experiência obtida nesse processo surgiu a ferramenta Delphos, outra instância do método SiSOA porém visando a análise de sistemas desenvolvidos com a arquitetura Java EE e com outro projeto e com outro desenvolvimento, descritos nesse trabalho.

Essa ferramenta está dividida em três fases: extração, identificação e análise. A primeira visa extrair informações do código fonte e dos arquivos de configuração do sistema alvo e criar um modelo. Na segunda fase a base de conhecimento em segurança, criada pelo especialista humano, é avaliada sobre esse modelo e tags de identificação de padrões encontrados são adicionadas ao modelo. Na última fase o analista recebe o modelo com as tags e métricas associadas e pode tomar as ações necessárias para correção do sistema ou para a mudança de determinadas características não previstas.

O processo de análise com o Delphos é interativo, ou seja, é possível realizar alterações na base de conhecimento ou diretamente no modelo do sistema e executar novamente o processo de identificação para que essas alterações sejam reavaliadas.

2 ANÁLISE ARQUITETURAL DE SEGURANÇA

Segundo Avizienis (2004) segurança é uma composição de atributos de confidencialidade (confidentiality), integridade (integrity) e disponibilidade (availability) que requer a existência concorrente de 1) disponibilidade somente para ações autorizadas, 2) confidencialidade e 3) integridade contra acesso não autorizado.

Já em Stoneburner (2002) é apresentado um conceito mais amplo: "Segurança em sistemas de informação é uma característica de sistema e um conjunto de mecanismos que abrangem o sistema tanto lógica como fisicamente." Mais adiante são apresentados 5 objetivos de segurança (security goals) que fornecem uma noção mais restrita: integridade (integrity), disponibilidade (availability), confidencialidade (confidentiality), responsabilidade (accountability) e garantia (assurance).

Esse capítulo tem como objetivo descrever esses e outros conceitos e técnicas fundamentais de segurança envolvidos nesse projeto.

2.1 Segurança de Dados

Existem diversos conceitos que caracterizam segurança de dados. Abaixo é apresentada uma lista de conceitos abordados direta ou indiretamente pela ferramenta.

2.1.1 Confidencialidade (Confidentiality)

"Ausência de acesso não autorizado a informações." Avizienis (2004).

Em Goodrich (2010) confidencialidade é definido como proteção de dados, ou seja, prover acesso às informações somente a quem possui permissão. Para atingir esse objetivo diversos conceitos são envolvidos: criptografia, controle de acesso, autenticação, autorização e segurança física.

2.1.2 Integridade (Integrity)

"Ausência de alterações impróprias do sistema." Avizienis (2004).

Além dos conceitos envolvidos em confidencialidade, Goodrich (2010) cita mais três conceitos envolvidos em integridade: *backups*, *checksums* e códigos de correção de erros. Esses conceitos adicionais envolvem uma característica fundamental para se evitar alterações de dados: redundância, a replicação de dados ou funções para que possamos detectar violações de integridade.

Além de proteger os dados em sí devemos manter a integridade dos metadados de cada arquivo, ou seja, as informações de controle de acesso, de modificações, de proprietários, etc.

2.1.3 Não repúdio (Nonrepudiability)

"Disponibilidade e integridade da identidade do emissor de uma mensagem (não repúdio de origem) ou do receptor (não repúdio de recepção)." Avizienis (2004).

Em Goodrich (2010) não repúdio é definido como a propriedade que declarações autênticas emitidas por um sistema ou por uma pessoa possuem de que não podem ser negadas.

2.1.4 Autorização (Authorization)

"Especificação e controle de ações autorizadas de determinados grupos de usuários em um sistema." Stoneburner (2002).

"Determinação se um sistema ou se uma pessoa possui permissão de acesso a determinado recurso baseado na política de controle de acesso. Tais autorizações devem evitar que um atacante engane o sistema a permitir o acesso não autorizado a recursos protegidos." Goodrich (2010).

2.1.5 Autenticação (Authentication)

"Integridade do conteúdo de uma mensagem, de sua origem e possivelmente de outras informações, como hora da emissão." Avizienis (2004).

Em Goodrich (2010) autenticação é definido como a determinação da identidade ou do papel de alguém. Isso pode ser realizado com uma combinação do que essa pessoa possui (smart card, chaves), com o que essa pessoa sabe (senha) e com o que essa pessoa é (impressões digitais).

2.1.6 Disponibilidade (Availability)

"Garantia de acesso ao serviço correto." Avizienis (2004).

Segundo Goodrich (2010) disponibilidade é a propriedade que a informação possui de ser acessada e de ser modificada constantemente por quem possui autorização. Dois conceitos fundamentais para que uma alta disponibilidade seja alcançada são: proteções físicas e redundância computacional.

2.1.7 Responsabilidade (Accountability)

"Disponibilidade e integridade da identidade da pessoa que realizou uma operação." Avizienis (2004).

2.1.8 Garantia (Assurance)

"Motivos para a confiança de que as outras quatro metas de segurança (integridade, disponibilidade, confidencialidade e responsabilidade) foram devidamente atingidas por uma implementação específica. 'Devidamente atingidas' inclui (1) funcionalidade que opera corretamente, (2) proteção suficiente contra erros não intencionais (por usuários ou por software) e (3) resistência suficiente a penetração intencional ou a desvio." Stoneburner (2002).

Garantia no contexto de segurança computacional, segundo Goodrich (2010), refere-se a como provemos e gerenciamos a confiança em sistemas computacionais. Confiança é uma característica difícil de se medir, no entanto sabemos que isso refere-se à percepção de que pessoas e de que sistemas estão se comportando como esperado.

Conceitos envolvidos em confiança são: políticas, permissões e proteções.

2.1.9 Criptografia (Cryptography)

Em Goodrich (2010) criptografia é definida como a transformação de informações usando um segredo, chamado chave de encriptação, para que as informações transformadas só possam ser lidas usando outro segredo, chamado chave de decriptação.

Quando as chaves de encriptação e decriptação são iguais as chamamos de chaves simétricas. Quando essas chaves são diferentes elas são conhecidas como chaves assimétricas.

2.1.10 Controle de acesso (Access control)

Controle de acesso é definido em Goodrich (2010) como regras e políticas que limitam o acesso a informações confidenciais às pessoas ou aos sistemas autorizados.

2.1.11 Segurança física (Physical security)

Segurança física é o estabelecimento de barreiras físicas que limitem o acesso a sistemas computacionais protegidos, segundo Goodrich (2010). Essas barreiras podem ser cadeados em gabinetes ou portas, ausência de janelas nas salas que armazenam dados sensíveis ou ainda a construção de *Faraday cages*, salas com paredes onde foram incorporadas malhas de cobre para evitar a entrada e saída de sinais eletromagnéticos.

2.1.12 *Backups*

Backup é a cópia e o arquivamento periódico de dados para que em caso de alteração indesejada ou não autorizada eles possam ser restaurados, segundo Goodrich (2010).

2.1.13 *Checksums*

Em Goodrich (2010) *checksums* são definidos como a computação de uma função que mapeia o conteúdo de um arquivo para um número. Uma função de checksum depende do conteúdo de todo o arquivo, de modo que qualquer alteração realizada sobre ele possui uma alta probabilidade de resultar num número diferente. Dessa maneira podemos detectar alterações não permitidas em arquivos.

2.1.14 Códigos de correção de erros (Data correcting codes)

Segundo Goodrich (2010) códigos de correção de erros são métodos para armazenar dados de uma maneira que pequenas mudanças possam ser facilmente detectadas e automaticamente corrigidas. Esses códigos são normalmente aplicados em pequenas unidades de armazenamento (a nível de byte ou palavra de memória), no entanto existem variantes que são aplicadas a arquivos inteiros.

2.1.15 Proteções físicas (Physical protections)

Segundo Goodrich (2010) proteções físicas são infraestruturas projetadas para manter informações disponíveis mesmo em condições físicas adversas. Essas proteções

incluem prédios construídos para armazenar os sistemas computacionais com capacidade para suportar tempestades, terremotos, bombardeios ou falta de energia.

2.1.16 Redundância computacional (Computational redundancies)

Redundância computacional são computadores e dispositivos de armazenamento redundantes que podem substituir os primários em caso de falhas, segundo Goodrich (2010). Como exemplo podemos citar RAIDs (arrays of inexpensive disks) utilizados para manter a disponibilidade de informações. Outro exemplo são fazendas de computadores utilizadas como servidores, uma falha em uma máquina individual pode ser contornada sem comprometer o serviço.

2.1.17 Políticas (Policies)

Segundo Goodrich (2010) políticas especificam o comportamento esperado que pessoas ou sistemas possuem para sí e para outros. Por exemplo, num sistema de músicas, designers devem descrever como usuários podem acessar e copiar músicas.

2.1.18 Permissões (Permissions)

Segundo Goodrich (2010) permissões descrevem que comportamentos são permitidos para agentes que interagem com uma pessoa ou com um sistema. Por exemplo, um sistema de músicas online deve permitir a um usuário acessar e copiar somente as músicas que comprou.

2.1.19 Proteções (Protections)

Segundo Goodrich (2010) proteções descrevem mecanismos que verificam permissões e políticas. No sistema de músicas online isso significa proteções contra o acesso e cópia não autorizada de músicas.

2.1.20 Autenticidade (Authenticity)

Autenticidade é a habilidade de determinar que declarações, políticas e permissões emitidas por pessoas ou sistemas são genuínos, segundo Goodrich (2010). Se esses documentos podem ser falsificados, contratos de compra e venda online não possuem validade, por exemplo. Esse conceito está intimamente ligado ao não repúdio e à assinatura digital.

2.1.21 Assinatura digital (Digital signature)

Segundo Goodrich (2010), assinatura digital é uma computação criptográfica que permite a uma pessoa ou a um sistema autenticar seus documentos de uma maneira única e não repudiável.

2.2 Engenharia Reversa

Segundo Chikofsky e Cross (2000) engenharia reversa é um processo de análise de um software para a criação de uma abstração de mais alto nível. Existem dois tipos de engenharia reversa, no primeiro tipo se possui o código fonte e deseja-se obter modelos que descrevem o sistema e no segundo tipo se possui o código binário do software e deseja-se obter o código fonte. Delphos enquadra-se no primeiro tipo.

Em Murphy (2001) é apresentada uma técnica de engenharia reversa chamada modelos de reflexão (software reflexion models). Essa técnica ajuda um engenheiro de software a comparar consistências de artefatos de níveis diferentes de abstração num sistema.

O objetivo dessa técnica é permitir a um engenheiro de software produzir um modelo estrutural de alto nível de um sistema para análise. Primeiramente o engenheiro define um modelo alvo, após essa definição ele utiliza uma ferramenta para extrair do código fonte do sistema analisado um modelo de artefatos, de chamadas ou de interações. Então ele define um mapeamento entre esses dois modelos. Um modelo de reflexão é então calculado automaticamente para analisar se o modelo extraído do código fonte corresponde ao modelo alvo definido.

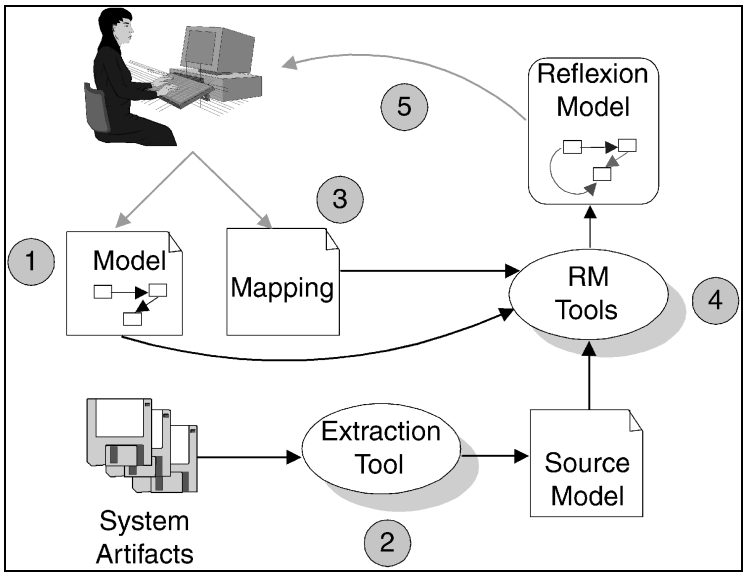

Figura 2.1: Modelos de reflexão. Murphy (2001).

Na Figura 2.1 podemos ver detalhadamente os passos propostos por essa técnica: (1) o especialista especifica um modelo que, pela sua experiência, caracteriza o sistema analisado, (2) utilizando uma ferramenta de *parser* de código fonte ele extrai um modelo real do sistema, (3) o especialista define um mapeamento entre os dois modelos, (4) uma ferramenta calcula o modelo de reflexão entre o modelo do analista e o modelo do código fonte e (5) com base no modelo de reflexão o especialista pode analisar o sistema alvo e tomar decisões quanto à sua arquitetura e quanto à sua implementação.

2.3 Princípios de segurança

Abaixo estão listados os 10 princípios de segurança criados por Saltzer e Schroeder em 1975, como apresentados em Goodrich (2010). Apesar de antigos eles caracterizam boas práticas de projeto de sistemas quanto à sua segurança.

2.3.1 Economia de mecanismos

Esse princípio atesta a simplicidade no design e implementação de mecanismos de segurança. É importante para usuários e engenheiros o fácil entendimento e a fácil

verificação de *frameworks* de segurança, portanto devemos adotar a abordagem minimalista na sua construção.

2.3.2 *Fail-safe defaults*

Esse princípio define que a configuração padrão de um sistema deve ter um esquema de proteção conservativo. Ao adicionarmos um usuário num sistema operacional por exemplo, ele deve possuir o menor conjunto de permissões de acesso possível. Dessa maneira, se não explicitamente permitido, o acesso a recursos deve ser proibido.

2.3.3 Completa mediação

Esse princípio atesta que todo o acesso a recursos deve ser verificado com o esquema de proteção. Embora essa política seja extenuante para o usuário e para o sistema pela constante verificação de permissões, ele visa garantir somente o acesso autorizado aos recursos.

2.3.4 Design aberto

De acordo com esse princípio a arquitetura e o design da segurança de um sistema deve ser público. Segurança deve ser construída com chaves criptográficas secretas e com algoritmos públicos. Ao tornar um sistema de segurança público ele será analisado, avaliado e testado por diversas pessoas e empresas, expondo falhas precocemente. Trocar a chave secreta é fácil, ao contrário de corrigir um sistema ameaçado por uma falha de design.

2.3.5 Separação de privilégios

Esse princípio afirma que diversos testes devem ser realizados antes que um programa execute alguma ação sobre recursos restritos. Da mesma forma, devemos limitar o dano potencial de um erro de um usuário ou de um componente.

2.3.6 Menor privilégio

Todo usuário e todo sistema deve operar com o menor conjunto de privilégios possível. Dessa maneira danos gerados por falhas serão os menores possíveis.

2.3.7 Menor mecanismo comum

Esse princípio atesta que mecanismos que permitem compartilhamento de recursos devem ser o menor possível. Assim, potenciais problemas gerados por um usuário ou por um sistema terão o menor efeito colateral possível.

2.3.8 Aceitação psicológica

Interfaces com o usuário devem ser intuitivas e bem projetadas. Todas as configurações relacionadas à segurança devem ser facilmente compreendidas. Ao não assimilar todos os efeitos gerados por um configuração de segurança, usuários podem comprometer todas as políticas de uma aplicação.

2.3.9 Fator trabalho

Ao projetar um mecanismo de segurança deve-se levar em conta os recursos que um atacante utilizará para comprometê-lo. Da mesma forma, o custo de comprometer um

mecanismo de segurança deve ser igual ou maior do que o valor obtido pelo acesso aos recursos protegidos.

2.3.10 Gravação de comprometimento

Esse princípio afirma que em alguns casos é desejável gravar os detalhes de uma intrusão a adotar medidas mais sofisticadas para prevení-lo. Instalar câmeras de segurança é desejável a reforçar todas as portas e janelas de um prédio. Em sistemas computacionais esse princípio se aplica na gravação de logs de acesso a todos os arquivos de um servidor, por exemplo.

3 DELPHOS

A ferramenta Delphos é um software de engenharia reversa que busca características sobre a segurança de sistemas. Serão descritas também ferramentas semelhantes que utilizam os mesmos conceitos.

3.1 Fases

Delphos é composto por três fases principais: extração, identificação e análise. As duas primeiras são automatizadas pela ferramenta e a terceira é realizada por um especialista em segurança que avalia os resultados encontrados através de relatórios e modelos.

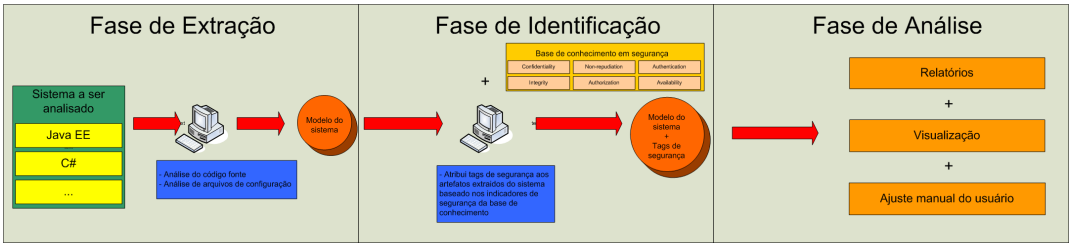

Figura 3.1: Fases de análise do Delphos

3.1.1 Extração

A fase de extração é responsável por criar o modelo do sistema analisado com base no seu código fonte e nos seus arquivos de configuração através de técnicas de engenharia reversa (Figura 3.2). Delphos foi projetado para analisar sistemas desenvolvidos com a arquitetura Java Enterprise Edition.

Através da biblioteca Java Development Tools (JDT) essa ferramenta cria um *parser* de classes Java do sistema analisado e cria um modelo desenvolvido com o Eclipse Modeling Framework (EMF) representado na Figura 3.3. Esse modelo basicamente armazena informações de relações entre classes (importação, exceção, herança, invocação e interface), anotações e a hierarquia de pacotes do sistema.

A hierarquia de pacotes e classes são representadas pelas classes DPackage e DCompilationUnit respectivamente. As relações de cada classe são armazenadas na classe DRelation, as anotações na classe DAnnotation e arquivos de configuração nas classes DConfig e DPolicy.

Com base nesse modelo Delphos irá procurar artefatos relevantes para a segurança do sistema como um todo.

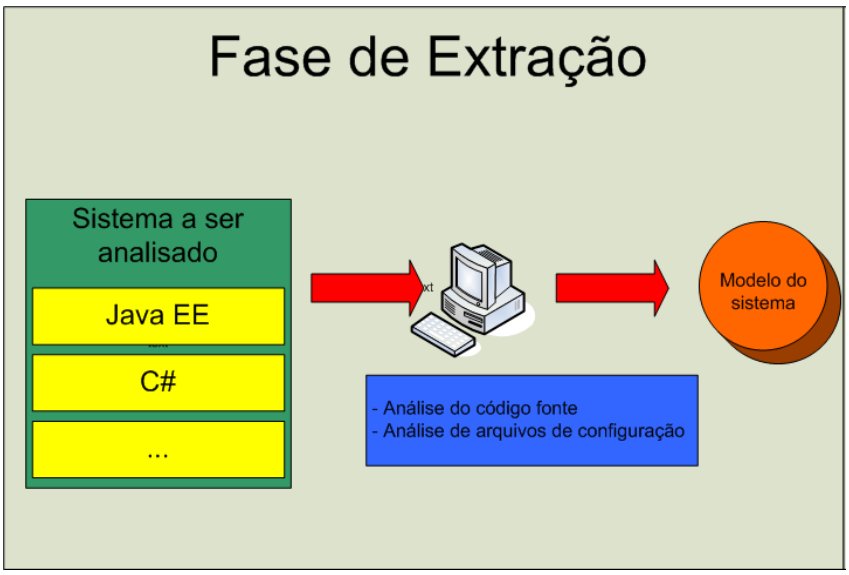

Figura 3.2: Fase de extração

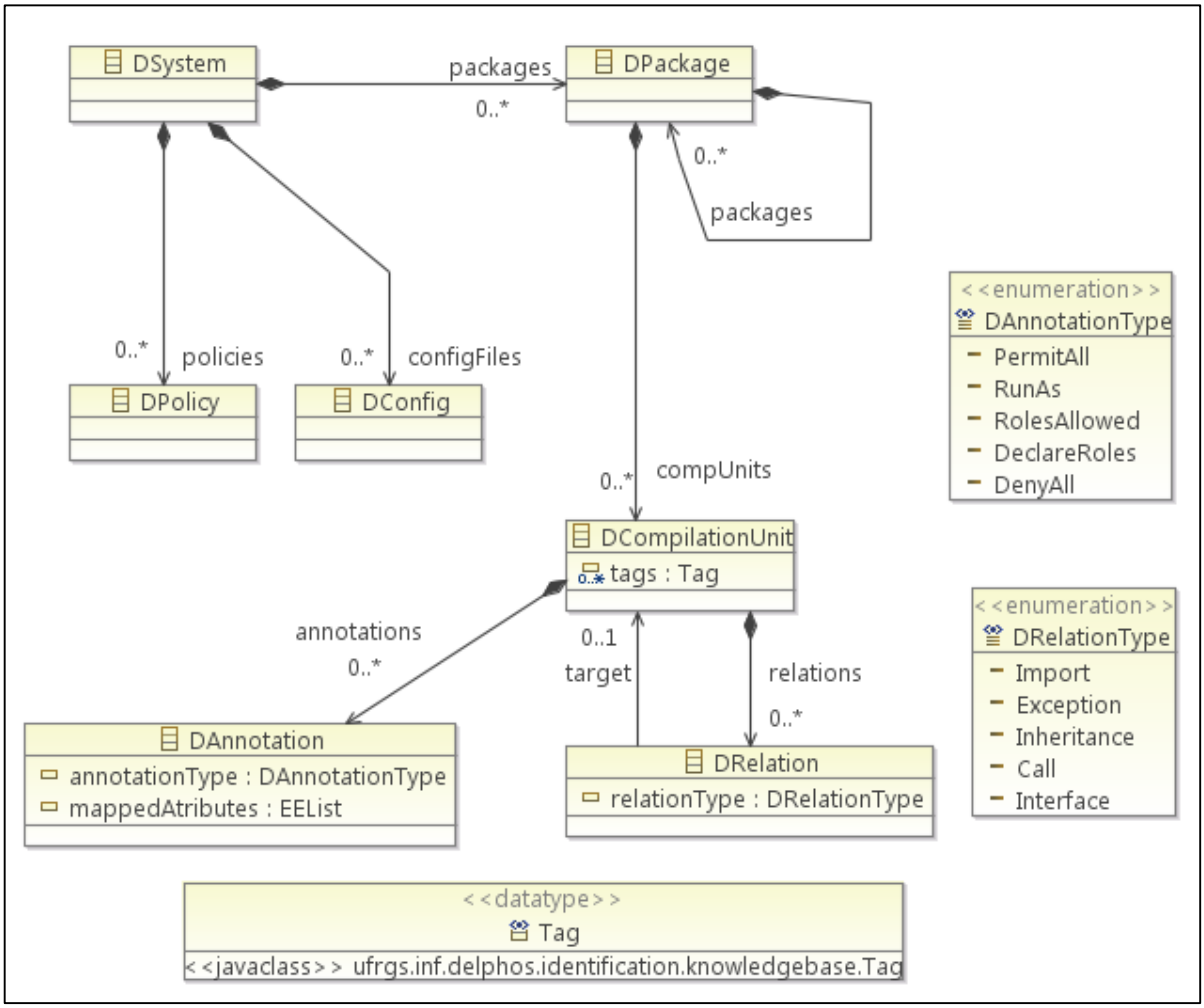

Figura 3.3: Modelo do sistema analisado no Delphos

O modelo do sistema analisado pelo Delphos (Figura 3.3) é armazenado numa classe raiz chamada DSystem que contém duas classes que representam políticas e configurações, DPolicy e DConfig respectivamente. Na classe DSystem também são armazenados os pacotes raízes do sistema analisado, que por sua vez podem conter outros pacotes ou compilation units, representadas pela classe DCompilationUnit. Nessa classe são armazenas as tags atribuídas pelo algoritmo de identificação, além de suas relações e de anotações.

3.1.2 Identificação

Nessa fase é executado um algoritmo interativo que busca artefatos da base de conhecimento em segurança no modelo de sistema e atribui as respectivas tags. Esse processo é detalhado no capítulo 4.

3.1.3 Análise

A fase de análise é onde o usuário recebe a avaliação do sistema através da visualização dos modelos e dos relatórios gerados. Nessa fase também é possível realizar alterações manuais na base de conhecimento em segurança e executar novamente o algoritmo de identificação para que essas alterações sejam propagadas na hierarquia de avaliação.

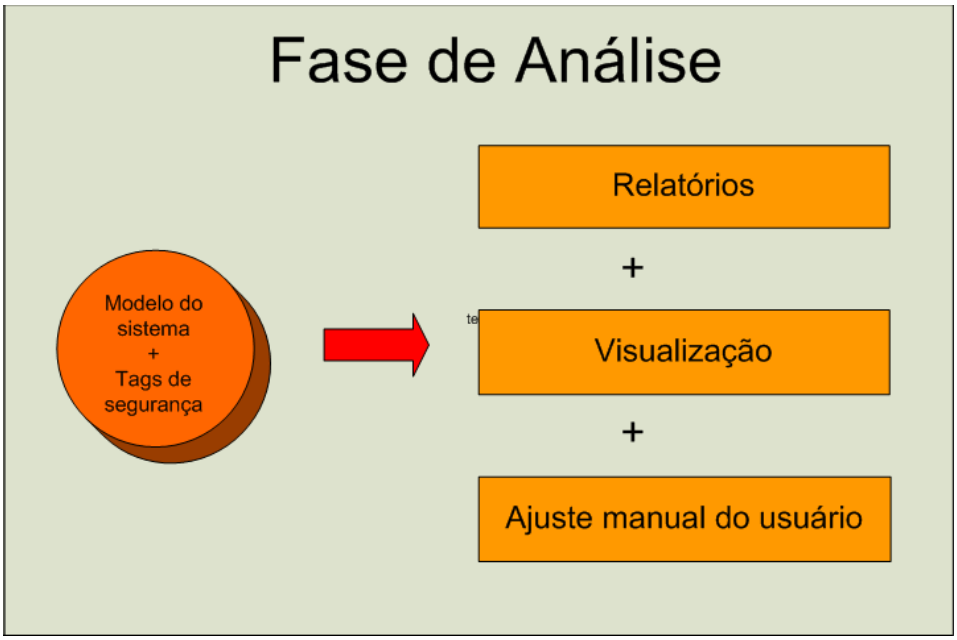

Figura 3.4: Fase de análise

O processo de análise do Delphos é interativo, ou seja, é possível fazer alterações na base de conhecimento ou diretamente no modelo, atribuindo tags a entidades ou criando novas árvores de avaliação na base de conhecimento, e executar novamente o processo de identificação. Nessa nova passagem o algoritmo levará em consideração as alterações realizadas e as propagará na árvore de avaliação do sistema. Dessa forma é possível ao analista interagir com a ferramenta para que ela seja capaz de se adequar às características particulares do sistema analisado.

3.2 Tecnologias

Delphos foi projetado e construído utilizando *model-driven architecture* com o suporte dos *frameworks* EMF, GMF e JDT sobre Eclipse.

3.2.1 Java EE

Java Enterprise Edition é uma plataforma baseada em componentes modulares que executam sobre um servidor de aplicação. Java EE é uma especificação que possui diversas implementações e é largamente utilizada no meio comercial.

Essa tecnologia foi escolhida como alvo para o Delphos pela grande quantidade de *frameworks* suportados. Dessa maneira Delphos atinge um público alvo amplo e diversificado.

3.2.2 Eclipse, EMF, GMF, JDT

Eclipse é um sistema de auxílio no desenvolvimento de software, uma IDE (Integrated Development Enviroment). Ele possui um extenso sistema de *plugins* que fornecem uma grande gama de ferramentas para auxiliar o desenvolvedor. Eclipse é a principal IDE utilizada para desenvolvimento de sistema com a linguagem Java. Por esse motivo Delphos foi desenvolvido como um *plugin* para o Eclipse.

EMF (Eclipse Modeling Framework) e GMF (Graphical Modeling Framework) são *frameworks* para o desenvolvimento de sistemas dirigidos a modelos. Eles foram utilizados pela facilidade de geração automática de código Java a partir da definição dos modelos do sistema, de geração automática de interfaces de visualização e edição de modelos.

Java Development Tools é um *framework* que auxilia no desenvolvimento de ferramentas que necessitam analisar código fonte Java. Ela foi utilizada pela facilidade de se estender um *parser* para montar o modelo de sistema do Delphos.

3.3 Ferramentas Similares

Essa seção tem como objetivo analisar *softwares* similares à ferramenta Delphos e explicar quais são as vantagens e desvantagens da nova abordagem proposta.

3.3.1 SAVE e SiSOA Tool

SAVE (Software Architecture Visualization and Evaluation) é uma ferramenta para analisar a arquitetura de *software*. Essa ferramenta extrai informações arquiteturais do código fonte do sistema que está analisando e cria um modelo desse software. Ao mesmo tempo o usuário descreve na ferramenta como seria o modelo teórico do sistema. Após os dois modelos estarem prontos é feita uma comparação entre eles para que a real implementação do sistema seja analisada.

A ferramenta SiSOA define a base do método de análise do Delphos, ela foi construída sobre a API definida no SAVE e possui as mesmas fases e processos. A principal diferença entre as duas abordagens é que a SiSOA analisa código fonte e arquivos de configuração de sistemas desenvolvidos com a arquitetura SCA (Service Component Architecture) enquanto o Delphos possui maior abrangência em sistemas corporativos ao analisar a arquitetura Java Enterprise Edition.

3.3.2 Bauhaus

Em Sohr e Berger é apresentada uma análise de segurança do ponto de vista arquitetural. Usando engenharia reversa de código fonte de sistemas, um modelo é formado. Com base nessa representação um especialista compara esse modelo com a especificação original do sistema e procura inconsistências. A principal diferença dessa abordagem com relação ao SiSOA e ao Delphos é que ela não armazena o conhecimento do especialista numa base de dados para uso posterior.

4 IDENTIFICAÇÃO DE ARTEFATOS DE SEGURANÇA

Artefatos de segurança são configurações, instruções ou relações que ao serem identificadas em um determinado contexto possuem um significado conhecido com relação à segurança do sistema analisado. Muitas vezes o conjunto de artefatos e de tags atribuídas a uma entidade indicam uma característica pertinente ao analista.

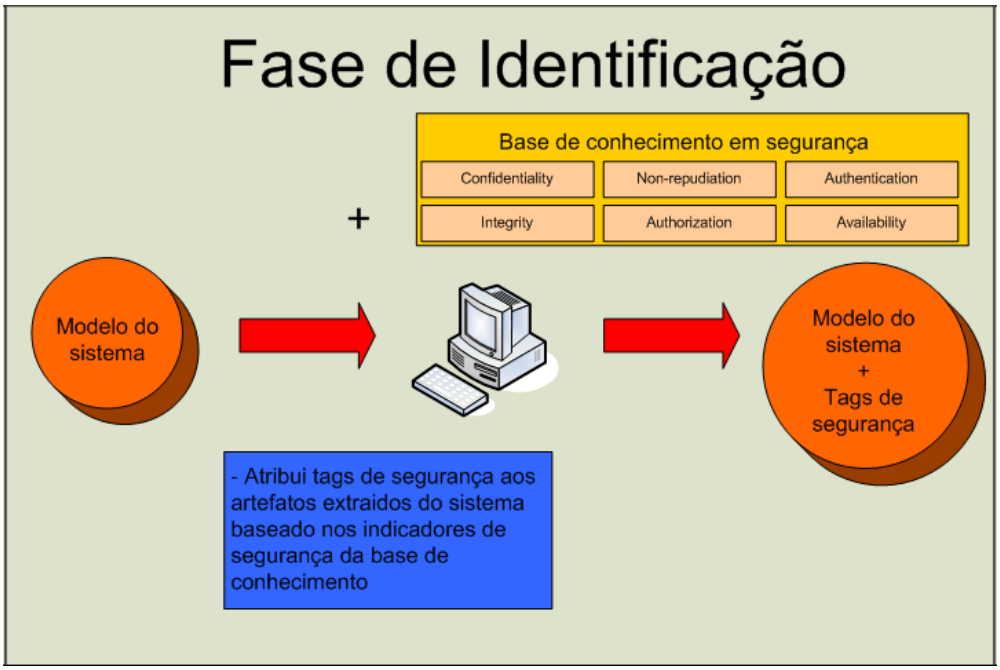

Figura 4.1: Fase de identificação

4.1 Base de Conhecimento em Segurança

Security Goal Indicator Tree (SGIT) foi desenvolvida para armazenar conhecimento sobre segurança em uma base de dados que pode ser reaproveitada, modificada e automaticamente avaliada sobre o modelo de um sistema.

A base de conhecimento em segurança do Delphos é formada por artefatos de segurança associados por conectores lógicos que indicam certas características, as tags. Por sua vez, um conjunto de tags pode levar à associação de um conceito mais abstrato relacionado à segurança do sistema, um indicador. Esses indicadores são agrupados em torno de 6 características chave de segurança analisadas pelo Delphos: confidencialidade, integridade, autenticação, autorização, disponibilidade e não repúdio.

Como apresentado em Peine (2008) características de segurança são enumeradas como características abstratas negativas, como não apresentar vazamento de informação, não negar os serviços disponibilizados, não permitir acesso não autorizado. SGIT foi desenvolvido para traduzir essas características em artefatos de código que podem ser identificados e classificados conforme a sua contribuição no sistema.

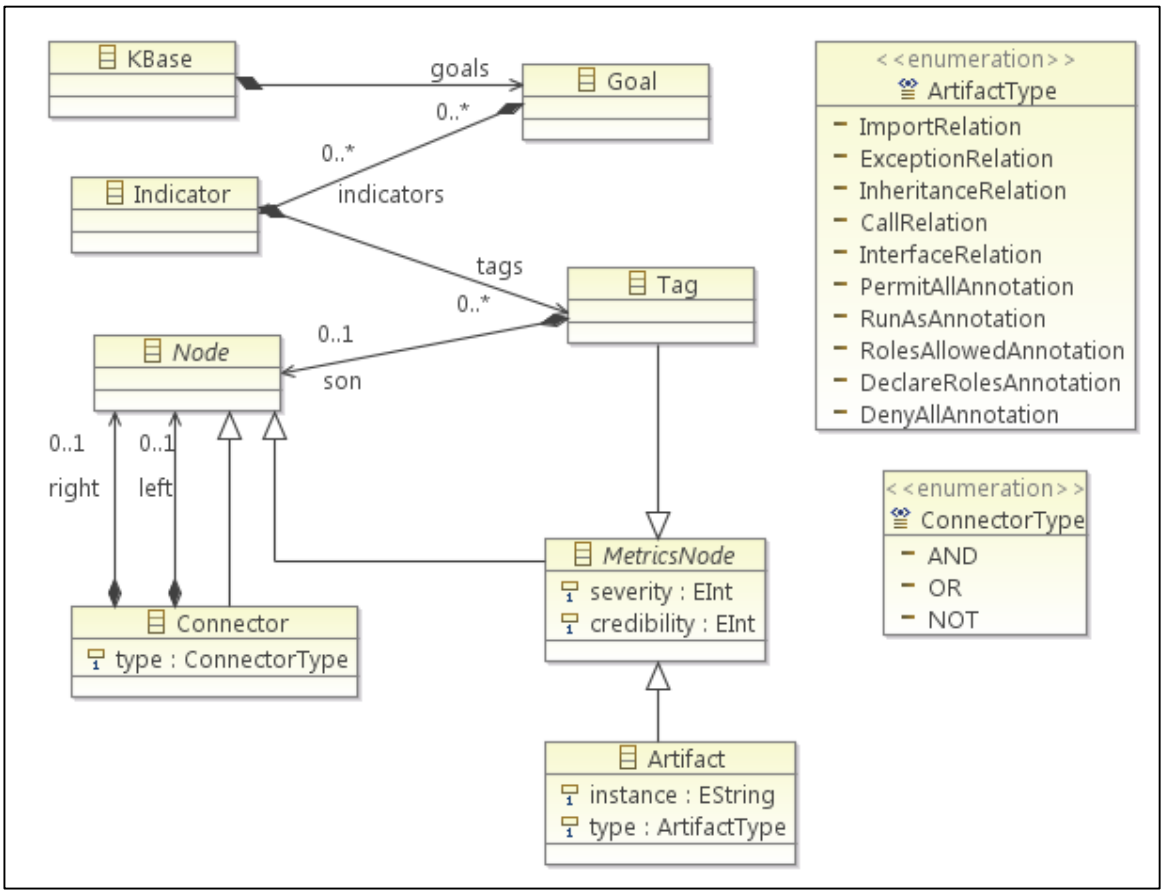

Figura 4.2: Modelo da base de conhecimento em segurança

A base de conhecimento em segurança do Delphos é armazenada numa classe chamada KBase onde existe uma lista com as 6 características analisadas pelo processo de identificação (classe Goal). Cada uma dessas características possui uma lista de indicadores (classe Indicator) que por usa vez possui uma lista de tags (classe Tag). A classe Tag é a raiz de uma árvore de outras tags, conectores (classe Connector) e artefatos (classe Artifact) que serão avaliados pelo algoritmo de identificação no modelo do sistema. A classe Connector une um ou dois elementos que podem ser tags, artefatos ou outros conectores. Essa união pode ser um *and*, *or* ou *not*, no caso de *not* esse conector é unário e indica a ausência de uma tag, artefato ou de outro ramo de avaliação. Artefatos são relações ou anotações que podem ser diretamente identificados em classes do sistema analisado ou em arquivos de configuração.

A classe NodeMetrics possui duas métricas comuns às classes Tag e Artifact: severidade e credibilidade. Ao se atribuir uma tag a uma classe do modelo do sistema é calculado um valor de severidade e credibilidade baseado nos artefatos que levaram a essa atribuição e que possuem valores pré-definidos para esses atributos. Atualmente é feita a média desses dois atributos dos artefatos que levaram à associação de uma tag,

no entanto como trabalho futuro pode-se evoluir essa definição de métricas que auxiliem o analista a classificar os padrões identificados pela ferramenta.

4.2 Algoritmo de Identificação

O algoritmo de identificação é responsável por procurar os artefatos da base de conhecimento no modelo do sistema analisado e atribuir as tags respectivas. O ponto de partida desse algoritmo são as tags filhas diretamente de indicadores. Essas tags são os elementos iniciais da recursão que avalia toda a árvore de tags, artefatos e conectores. Para cada *compilation unit* do modelo do sistema o algoritmo de identificação é invocado para atribuir as suas tags.

Uma tag possui um objeto da classe Node que pode ser um conector unário ou binário, um artefato ou outra tag. Em caso de artefato ou tag, o ponto de parada da recursão, o algoritmo pergunta ao modelo se a classe sendo avaliada possui tal artefato ou tag, em caso positivo essa tag é atribuída à *compilation unit*. No caso desse objeto ser um conector são avaliados recursivamente ambos os ramos da árvore e é retornado verdadeiro ou falso para ambas as avaliações que são então unidas pelo tipo do conector. Finalmente, caso verdadeiro, a tag é atribuída à *compilation unit*.

Como podem existir dependências implícitas entre tags de diferentes ramos da árvore, o algoritmo de identificação é executado repetidamente sobre uma *compilation unit* do modelo até que nenhuma tag seja atribuída.

5 ESTUDO DE CASO

Para demonstrar a ferramenta Delphos foi desenvolvido um sistema simples em Java EE que implementa uma pizzaria online. Nesse sistema existe um cadastro de cliente, um cadastro de pizzas e um cadastro de pedidos onde o usuário pode se cadastrar e fazer pedidos de entrega de pizza. Todas essas informações são persistidas num banco de dados e consultadas conforme a interação com o usuário.

Na primeira fase da análise Delphos utiliza um *parser* para extrair do código fonte dessa aplicação o modelo exibido na Figura 5.1. Como podemos notar a aplicação possui três pacotes: presentation, boundary e entities. As classes dos pacotes presentation e boundary acessam as classes do pacote entities. No segundo pacote as classes da aplicação são filhas da classe AbstractFacade. Dessa forma, podemos documentar importantes características da arquitetura dessa aplicação, como relações entre entidades e possíveis violações à especificação original.

Para a extração dos artefatos do código fonte foi utilizado o *framework* JDT (Java Development Tools) que provê um *parser* para código fonte Java. Esse *parser* foi projetado utilizando o padrão Visitor, que permite o registro de *callbacks* para processamento dos artefatos de código encontrados. Dessa maneira Delphos percorre em profundidade o sistema analisado. Primeiramente são criados os artefatos no modelo de sistema que correspondem à hierarquia de pacotes da aplicação alvo. Logo após as classes contidas nesses pacotes são percorridas e suas relações e suas anotações são representadas no modelo de sistema.

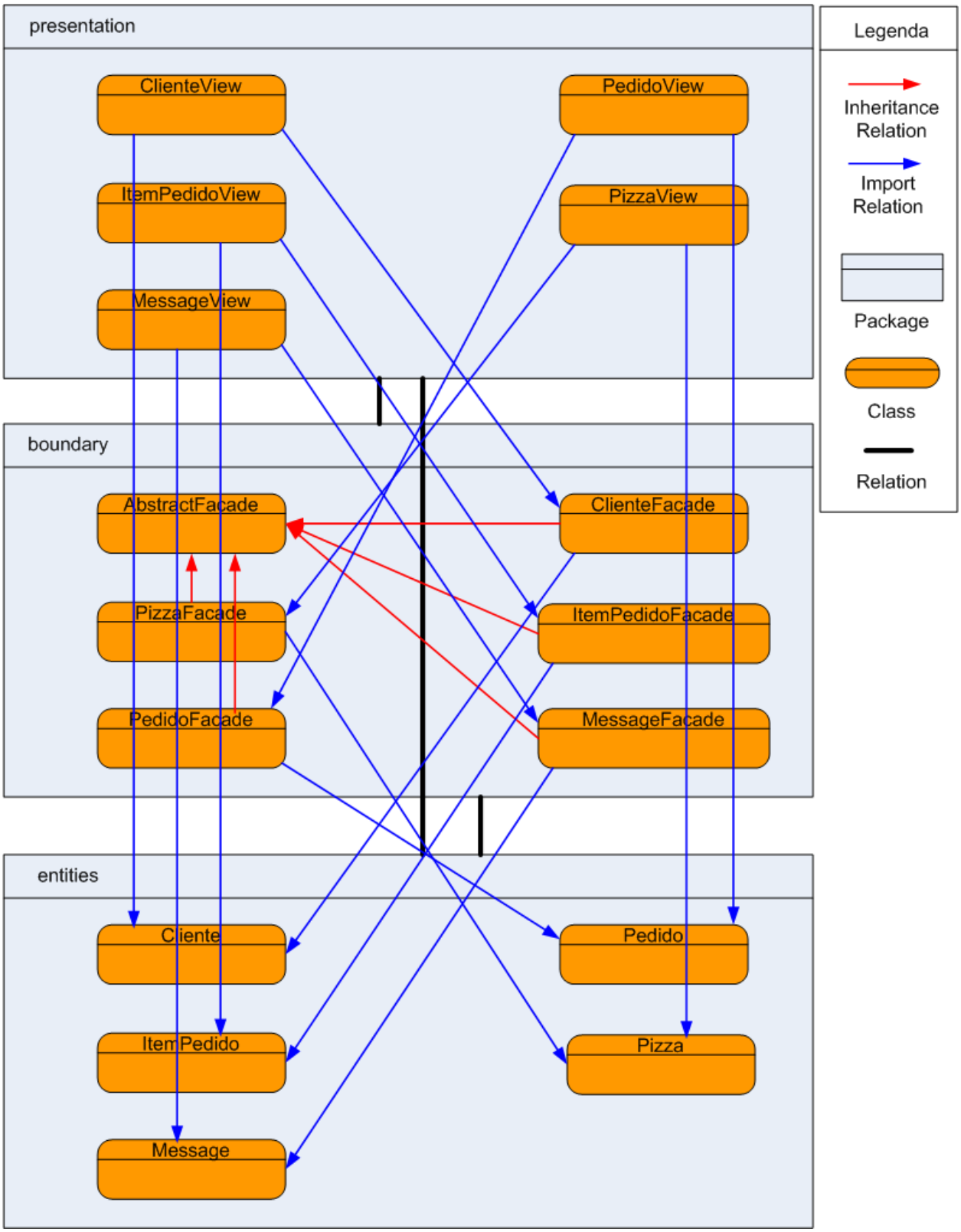

Figura 5.1: Modelo do sistema da pizzaria após a fase de extração

Na Figura 5.2 é exibida a representação gráfica de uma base de conhecimento em segurança. Podemos notar em laranja os 6 objetivos principais da análise do Delphos: encontrar aspectos relevantes à confidencialidade, à integridade, à autenticação, à autorização, ao não repúdio e à disponibilidade. Em azul estão representados elementos que auxiliam na caracterização dos objetivos acima, os indicadores. Um indicador de que existe confidencialidade na aplicação é a presença de criptografia e um indicador de disponibilidade é a maneira como são tratados os estados das requisições de um sistema por exemplo. Já em vermelho são representadas as tags, objetos que são armazenados no modelo do sistema analisado conforme o algoritmo de identificação os encontra. Como folhas dessa árvore, que representa o conhecimento do analista em segurança, estão os artefatos, em verde, que podem ser diretamente encontrados no código fonte da aplicação e em seus arquivos de configuração. Artefatos podem ser ligados por conectores lógicos que criam relações entre eles para definir semânticas de atribuição de tags.

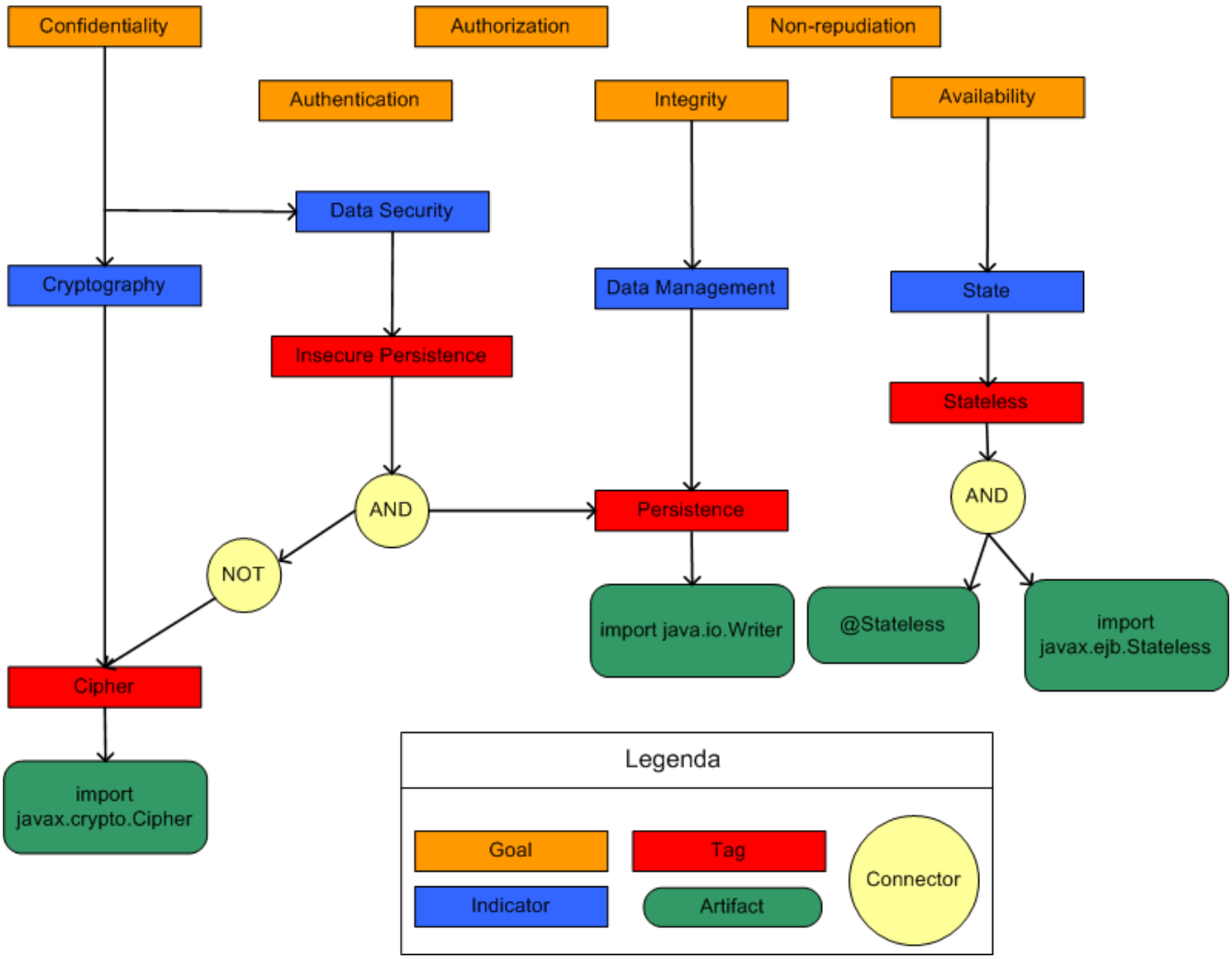

Figura 5.2: Base de conhecimento usada para analisar o sistema da pizzaria

O algoritmo de identificação recebe essa base de conhecimento (Figura 5.2) e o modelo de sistema (Figura 5.1) como entrada. A partir desses dados ele realiza o mapeamento das tags da base de conhecimento para o modelo de sistema. Esse procedimento cria, assim, um modelo avaliado para auxiliar o analista a encontrar características de segurança interessantes. O primeiro passo desse algoritmo é criar uma lista de tags filhas diretamente de indicadores. Essas tags são as raízes de árvores de avaliação.

O algoritmo de identificação, com a lista de árvores de avaliação, percorre cada artefato do modelo do sistema que representa uma classe e avalia as árvores da lista sobre cada um desses artefatos individualmente.

Cada árvore é então percorrida em profundidade, onde ao ser encontrado um artefato (obrigatoriamente folhas da árvore de avaliação) ou uma tag sem filhos, o artefato do modelo do sistema é questionado se possui tal elemento da base de conhecimento. A resposta a esse questionamento é unido conforme a semântica do conector pai desses elementos e assim, subindo na árvore de avaliação, encontra-se uma tag e o valor verdade representa a atribuição ou não dessa tag à classe do modelo de sistema.

A lista de árvores de avaliação é percorrida repetidamente sobre uma classe do modelo até que nenhuma nova tag seja atribuída. Esse procedimento é necessário porque podem existir dependências entre as árvores de avaliação. Uma tag pode ser referenciada ao criarmos uma outra tag com o mesmo nome e sem filhos em outra árvore de avaliação. Ao terminar a aplicação desse procedimento em todas as classes o algoritmo de identificação encerra e retorna o modelo de sistema avaliado.

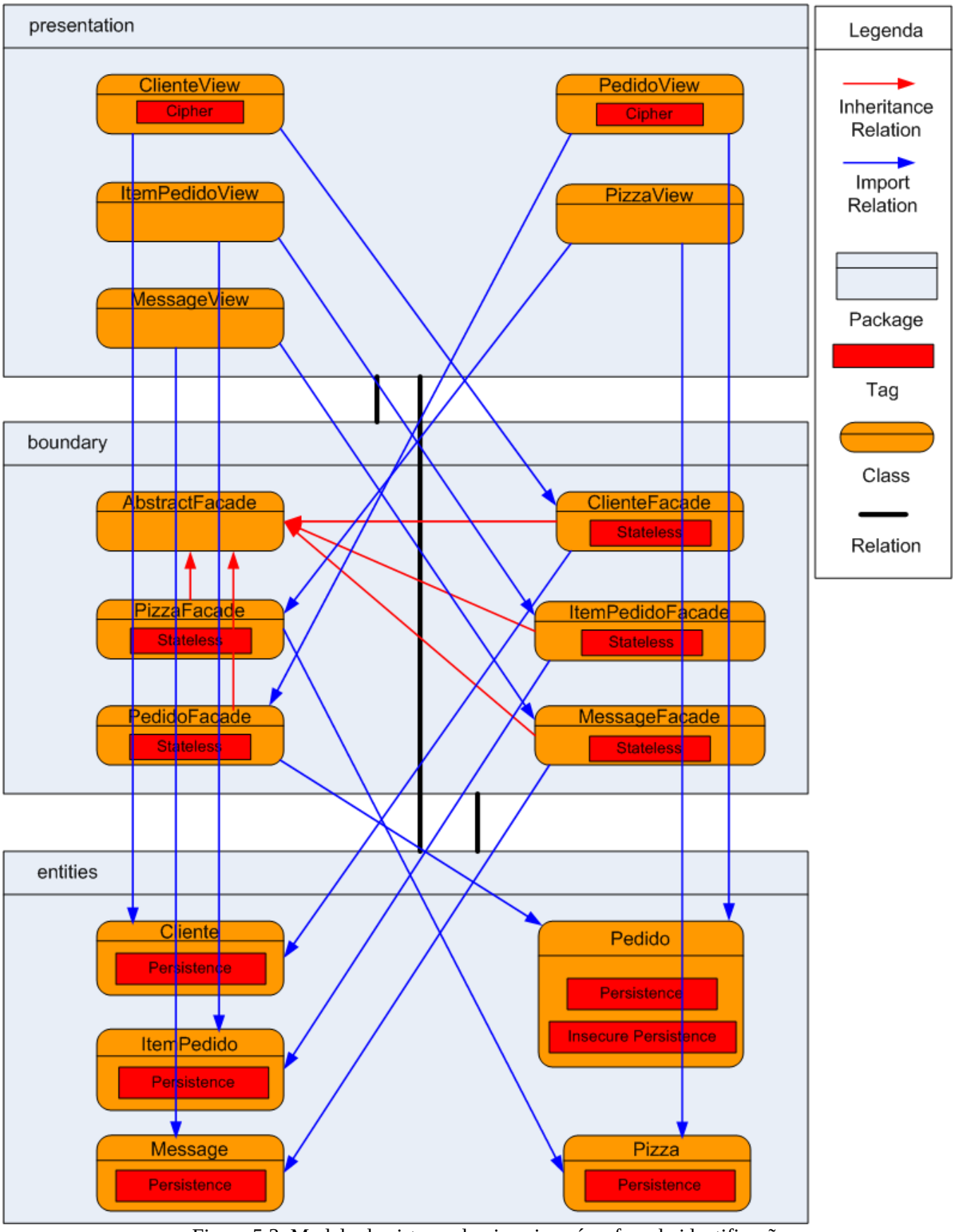

Figura 5.3: Modelo do sistema da pizzaria após a fase de identificação

Após a avaliação dessa base de conhecimento sobre o modelo do sistema temos um modelo avaliado, representado na Figura 5.3. Podemos notar agora as tags mapeadas nas entidades que representam as classes do sistema. A tag Stateless foi mapeada para as classes do pacote boundary, filhas da classe AbstractFacade, ou seja, as requisições que essas classes atendem não possuem estado, após o processamento de uma requisição os seus dados são perdidos. A tag Cipher foi mapeada para as classes ClienteView e PedidoView do pacote presentation, ou seja, os dados transferidos entre cliente e servidor são criptografados. A tag Persistence foi mapeada para as classes do pacote entities, ou seja, os dados processados por essas entidades são armazenados em arquivos. Por fim a tag Insecure Persistance foi mapeada para a classe Pedido. Essa última tag mapeada possui uma característica especial. Como podemos notar na Figura 5.2, a tag Insecure Persistence deriva da presença da tag Persistence, que indica que os dados do pedido são armazenados em arquivos, e da ausência da tag Cipher, o que indica que os dados não são cifrados. Dessa maneira, podemos concluir que a classe Pedido possui uma séria falha de segurança pois armazena dados em arquivos em claro, não cifrados.

Esse modelo avaliado é a saída da ferramenta Delphos. Ele é exibido ao analista e possibilita a edição e filtragem de tags, relações e entidades. Assim, o especialista em segurança pode avaliar a identificação realizada pela ferramenta e pode editar tanto a base de conhecimento, como o modelo. Após essa reavaliação é possível executar o processo de identificação novamente para que o modelo final reflita as alterações da base de conhecimento e do modelo de sistema. Esse processo pode ser repetido até que a qualidade e os aspectos desejados sejam atingidos.

6 CONCLUSÃO

A ferramenta Delphos apresenta uma nova abordagem para o método SiSOA. Desenvolvido para a análise de sistemas construídos com a arquitetura Java EE, essa ferramenta disponibiliza um método de identificação de características relevantes com relação à segurança da informação.

Através da análise dos componentes constituintes de um sistema computacional podemos fazer uma conferência com a sua especificação inicial e procurar violações originadas dos processos de projeto, de desenvolvimento e de manutenção. Tais anomalias podem gerar sérias falhas com relação à performance e à segurança de dados tanto de empresas como de clientes.

Além de fornecer uma visão geral da interação de componentes e suas relações, Delphos aplica um processo de reconhecimento de padrões baseado numa base de conhecimento que possibilita a automatização da análise do código fonte de aplicações. Em ambientes onde sistemas possuem milhões de linhas de código, se torna inviável para um analista de segurança percorrer todas as entidades de um sistema em busca de possíveis brechas de segurança. Dessa maneira Delphos se apresenta como uma ferramenta que indica possíveis pontos de falhas e possibilita a armazenagem do conhecimento do analista para uso posterior e automático.

REFERÊNCIAS

AVIZIENIS A.; LAPRIE J.; BRIAN R.; CARL L. Basic Concepts and Taxonomy of Dependable and Secure Computing. **IEEE Transactions on Dependable and Secure Computing**. 2004.

ARMSTRONG E. **The J2EE™ 1.4 Tutorial**. Sun Microsystems, Inc. 2005.

STONEBURNER G.; GOGUEN A.; FERINGA A. **Risk Management Guide for Information Technology Systems: Recommendations of the National Institute of Standards and Technology.** National Institute of Standards and Technology. 2002.

ROST D.; FORSTER T.; KNODEL J. **The SAVE Plug-in - Internal Data Model and Architecture Evaluation Functionality,** Kaiserslautern: Fraunhofer IESE, 2006.

DUSZYNSKI S.; KNODEL J.; LINDVALL M. SAVE: Software Architecture Visualization and Evaluation. **European Conference on Software Maintenance and Reengineerin**. 2009.

ANTONINO P.; DUSZYNSKI S.; JUNG C.; RUDOLPH M. Indicator-based Architecture-level Security Evaluation in a Service-oriented Environment. **ECSA '10 Proceedings of the Fourth European Conference on Software Architecture.** 2010.

KNODEL J.; LINDVALL M.; MUTHIG D.; NAAB M. Static Evaluation of Software Architectures, **Software Maintenance and Reengineering.** 2006.

MURPHY G.; NOTKIN D.; SULLIVAN K. J. Software Reflection Models: Bridging the Gap between Design and Implementation, **IEEE Transactions on Software Engineering**, vol. 27, n. 4, p. 364-380, Abr. 2001.

PEINE H.; JAWUREK M.; MANDEL S. Security Goal Indicator Trees: A Model of Software Features that Supports Efficient Security Inspection. **High Assurance Systems Engineering Symposium**, 2008.

GOODRICH M.; TAMASSIA R. **Introduction to Computer Security.** 1st ed. New York: Addison Wesley, 2010.

MURPHY G. C.; NOTKIN D.; SULLIVAN K. J. Software Reflexion Models: Bridging the Gap between Design and Implementation. **IEEE Transactions on Software Engineering**, vol. 27, n. 4, Abr. 2001.

BIBLIOGRAFIA

SCHNEIER, B. **Applied Cryptography: protocols, algorithms, and source code in C**. 2nd ed. New York: John Wiley & Sons, Inc., 1996.

HOWARD, M.; LEBLANC, D. **Escrevendo Código Seguro: estratégias e técnicas práticas para codificação segura de aplicativos em um mundo de rede**. 2nd ed. Porto Alegre: Bookman, 2005.

ISO / IEC 27000 (2009)