UNIVERSIDADE FEDERAL DE SANTA MARIA
COLÉGIO AGRÍCOLA DE FREDERICO WESTPHALEN
PROGRAMA DE PÓS-GRADUAÇÃO EM GESTÃO DE TECNOLOGIA DA INFORMAÇÃO

ARGOS: SOLUÇÃO CENTRALIZADA PARA LOGGING DE APLICAÇÕES

MONOGRAFIA DE ESPECIALIZAÇÃO

Ezequiel Juliano Müller

Frederico Westphalen, RS, Brasil

2013

ARGOS:
SOLUÇÃO CENTRALIZADA PARA LOGGING DE APLICAÇÕES

Ezequiel Juliano Müller

Monografia apresentada ao Curso de Especialização do Programa de Pós-Graduação em Gestão de Tecnologia da Informação, Área das Ciências Exatas, da Universidade Federal de Santa Maria (USFM, RS), como requisito parcial para obtenção do grau de
Especialista em Gestão de Tecnologia da Informação.

Orientador: Prof. Msc. Roberto Franciscatto

Frederico Westphalen, RS, Brasil

2013

Universidade Federal de Santa Maria
Colégio Agrícola de Frederico Westphalen
Programa de Pós-Graduação em Gestão de Tecnologia Da Informação

A Comissão Examinadora, abaixo assinada,
aprova a Monografia de Especialização

ARGOS:
SOLUÇÃO CENTRALIZADA PARA LOGGING DE APLICAÇÕES

elaborada por
Ezequiel Juliano Müller

como requisito parcial para obtenção do grau de
Especialista em Gestão de Tecnologia da Informação

COMISSÃO EXAMINADORA:

Prof. Msc. Roberto Franciscatto
(Presidente/Orientador)

Prof. Msc. Fernando de Cristo (UFSM)

Prof. Msc. Antonio Rodrigo Delepiane De Vit (UFSM)

Frederico Westphalen, 20 de julho de 2013.

DEDICATÓRIA

Dedico este trabalho a meu pai e minha mãe pelo esforço e dedicação em todos os momentos desta e de outras caminhadas, e pelo exemplo de vida e família. A minha noiva pela compreensão e companheirismo. Aos meus demais familiares por me ensinar a fazer as melhores escolhas, mostrando-me que honestidade e respeito são essências à vida e que nunca devemos desistir de nossos objetivos.

AGRADECIMENTOS

Agradeço primeiramente a Deus por criar o universo, por dar a capacidade e a inteligência às pessoas para desenvolverem e assim complementarem a criação e por iluminar meu caminho. Sem o universo e sem os recursos contemporâneos que o homem possui seria impossível desenvolver este trabalho, enquanto discentes e seres humanos na sagaz busca do saber.

Aos meus pais Egon e Natalina Müller, pois sem eles eu não estaria aqui, e por terem me fornecido condições para que me tornasse profissional e homem que sou.

A minha noiva Greiciane Simionato, pelo apoio, compreensão e dedicação.

Aos meus demais familiares que, com muito carinho e apoio não mediram esforços para que eu vencesse mais essa etapa em minha vida, estando sempre presentes nos momentos de dificuldade no decorrer desse curso.

Aos amigos, aos que já faziam parte de minha vida e aqueles que conheci no decorrer do curso. Amigos que estiverem presentes nos momentos de alegria e descontração e também nos momentos de dificuldade, para me dar o apoio que tanto precisava.

Ao professor e orientador Roberto Franciscatto por seu apoio e incentivo no amadurecimento de meu conhecimento, tornando possível a execução e conclusão deste trabalho.

A Microsys Sistemas e seus colaboradores pelo apoio, compreensão, incentivo e conhecimento repassado, estes foram essenciais na execução deste trabalho.

Finalmente, mas não menos importante, aos demais professores, que repassaram o conhecimento necessário para a elaboração deste trabalho e para a carreira profissional que segue.

RESUMO

Monografia de Especialização
Programa de Pós-Graduação em Gestão de Tecnologia da Informação
Universidade Federal de Santa Maria

ARGOS:
SOLUÇÃO CENTRALIZADA PARA LOGGING DE APLICAÇÕES

AUTOR: EZEQUIEL JULIANO MÜLLER
ORIENTADOR: ROBERTO FRANCISCATTO
Data e Local da Defesa: Frederico Westphalen, 20 de julho de 2013.

Este trabalho apresenta o desenvolvimento de um *software* sob a modalidade *Open Source*, que usa a *web* como plataforma de execução, dotado de funcionalidades capazes de realizar de forma centralizada o armazenamento, gerenciamento e análise de eventos de *logs* gerados por diferentes aplicações. O desenvolvimento deste tipo de solução visa garantir a segurança da informação, detectando atividades não autorizadas ou em não conformidade com as políticas de segurança dos sistemas de informações, pois a informação é um ativo essencial para qualquer organização e necessita ser protegida. Adotar medidas adequadas efetuando *logging* de aplicações por meio de uma solução capaz de fornecer uma análise precisa e que auxilie no processo de tomada de decisão, permitindo a resolução dos problemas do dia-a-dia organizacional, torna-se um processo necessário e irreversível. O *software*, estudo deste trabalho, possui como foco operacional a tarefa de *logging* em sistemas integrados de gestão empresarial, gestão de relacionamento com clientes, gestão de cadeia de logística e planejamento dos recursos de manufatura. Para compreender e fundamentar a importância do desenvolvimento deste sistema, a revisão bibliográfica aborda assuntos relacionados a evolução e importância da informação, sistemas de informação, classificação e ciclo de vida informação, segurança da informação (ambiente físico, lógico e aspecto humano), administração estratégica, políticas e planos de segurança, normas, desenvolvimento seguro de *software*, registros de eventos (*logs*) e alguns trabalhos relacionados. Para evidenciar o cenário de atuação, processos, testes e tecnologias envolvidas, o capítulo de desenvolvimento aborda todo o ciclo de construção da aplicação, resultados obtidos e testes funcionais executados, fundamentando a dimensão e reconhecendo a aplicabilidade da tarefa de *logging* em *softwares* comerciais.

Palavras-chave: *Logging* de Aplicações. *Software Open Source*. Gestão Centralizada de *Logs*. Segurança da Informação.

ABSTRACT

Monograph Specialization
Graduate Program in Management of Information Technology
Federal University of Santa Maria

ARGOS:
CENTRALIZED SOLUTION FOR LOGGING OF APPLICATIONS

AUTHOR: EZEQUIEL JULIANO MÜLLER
ADVISER: ROBERTO FRANCISCATTO
Date and Place of Defense: Frederico Westphalen, July 20, 2013.

This job presents the development a software beneath in modality Open Source, which uses the web as platform execute, endowed with features to make storage, management and analysis of events logs generated by different applications of form centrally. The development of this type of solution aims to ensure information security, enabling the detection unauthorized activity or non-compliance with security policies for information systems, because the information is an essential asset for any organization and needs to be protected. Adopt appropriate measures logging application by means of a solution capable of providing an accurate analysis for help the decision-making process and that allows the resolution of the problems of day-to-day organizational, becomes a necessary and irreversible process. The software, study of this work, has operational focus in the task logging for systems integrated business management, customer relationship management, supply chain management, logistics and planning of manufacturing resources. To understand and explain the importance of developing this system, the literature review addresses issues related to evolution and importance of information, information systems, classification and lifecycle the information, information security (physical environment, logical and human aspect), strategic management, policies and plans, standards, secure software development, event logs (logs) and some related work. To evidence the scene of acting, processes, tests and technologies involved, the chapter of development covers the entire cycle of building the application, results obtained and functional tests performed, basing the dimension and recognizing the applicability of the task of logging in commercial software.

Keywords: Logging Application. Open Source Software. Centralized management of logs. Information Security.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

A informação assume, nos dias de hoje, um papel fundamental e uma importância extremamente relevante para o funcionamento tático, estratégico e operacional de qualquer organização. Ela está presente e é fundamental em todos os níveis organizacionais, proporcionando introdução de novas tecnologias, exploração de oportunidades e principalmente como ferramenta para a consolidação da estratégia competitiva.

Considerada um ativo altamente significativo e essencial para os negócios de uma organização, a informação deve ser gerenciada e protegida com a adoção de medidas adequadas, principalmente em um ambiente de negócios cada vez mais interconectado. Garantir a segurança da informação significa dar continuidade aos negócios, minimizar riscos e ameaças e maximizar investimentos e oportunidades.

A segurança da informação está inserida em um contexto, onde para poder alcançá-la, se faz necessária a adoção de um conjunto de controles adequados, desde a definição de políticas e processos organizacionais, bem como soluções em *hardware* e *software*. Neste sentido, a tarefa de *logging* de aplicações se destaca, pois os *logs* são fontes riquíssimas de informações que são gerados sistemas informatizados a todo o momento que eventos significativos acontecem, possibilitando um verdadeiro rastreamento do processo de geração da informação, além de proporcionar uma análise da conformidade dos controles e processos internos.

Devido à grande quantidade de informação gerada a todo o momento, os registros de *logs* tem se tornado ao longo dos últimos anos, verdadeiros aliados dos profissionais de TI, pois oferecem mecanismos de registros de eventos que facilitam a resolução de problemas. Porém, a implementação destes mecanismos na maioria das vezes se constitui em uma tarefa complexa, pois para criar e gerenciar saídas de *logs* é necessária inclusão de comandos diretos no código fonte da aplicação, aumentando assim o tempo de desenvolvimento e a complexidade do código gerado. Além disso, existem outros problemas como o fato de não haver padrões para o conteúdo e formato dos arquivos de *logs* gerados, dificultando o seu uso por sistemas de análise, ou ainda o armazenamento dos *logs* na própria base de dados da aplicação, ocasionando perda de desempenho e possibilitando que o usuário possa negligenciar estas informações.

Quando a segurança da informação é negligenciada, ignorando o armazenamento de registros de *logs*, vários aspectos importantes como a confidencialidade, integridade e disponibilidade são afetados. Além disso, não é possível garantir que um usuário é de fato quem alega ser, o sistema perde a capacidade de provar que um usuário executou determinada ação e não se tem uma rastreabilidade de tudo o que foi realizado no sistema, impossibilitando a detecção de fraudes ou alterações indevidas. O *logging* é uma dimensão muito importante de uma aplicação e é extremamente importante reconhecer a sua aplicabilidade. Neste trabalho, é proposto o desenvolvimento de um *software* capaz de centralizar diferentes tipos de *logs*, gerados por diferentes aplicações em um único repositório, garantindo um armazenamento seguro das informações de *logs* administrando-os de forma objetiva.

O *software*, estudo deste trabalho, possui seu foco voltado para o gerenciamento de *logs* de aplicações de gestão empresarial. Seu objetivo é proporcionar para as organizações, em especial as *softwares houses*[1], uma solução especializada e com repositório central para gestão de *logs*, garantindo um maior monitoramento e análise dos eventos de segurança da informação, possibilitando uma tomada de decisão mais eficaz e proporcionando uma garantia de que problemas ou comportamentos anormais sejam identificados e atendidos com uma maior brevidade.

O que se buscou foi desenvolver um *software* sob a modalidade *Open Source*, que possui a *web* como plataforma de execução, e capaz de realizar de forma centralizada e em tempo real o gerenciamento e análise de eventos de *logs* gerados por diferentes aplicações. O *software* foi denominado Argos e oferece funcionalidades capazes de auxiliar na detecção de atividades não autorizadas ou em não conformidade com as políticas de segurança da informação adotadas por estas aplicações. Por se tratar de um *software* que necessita processar grande quantidade de dados, optou-se pela utilização de tecnologias emergentes capazes de suprir grandes demandas como o *Demoiselle*, que se trata de um *framework* integrador de diversas tecnologias especialistas *Java*, o *MongoDB*, banco de dados *NoSQL* distribuído e altamente escalável e o *Apache Lucene*, biblioteca de mecanismo de procura de texto também altamente escalável.

[1] Empresas ou organizações que se dedicam a construir programas para ambientes computacionais, normalmente para fins comerciais.

1.1 Caracterização do Trabalho

Este trabalho se caracteriza como desenvolvimento tecnológico e sob o ponto de vista de sua natureza é classificado como uma pesquisa aplicada, pois implica na utilização dos conhecimentos existentes, tendo em vista a resolução de problemas específicos. A sua abordagem é qualitativa, pois foca nos processos e seus significados e na interpretação de fenômenos. E do ponto de vista de seus objetivos é considerado uma pesquisa exploratória, pois visa proporcionar maior familiaridade com o problema, buscando torná-lo mais explícito.

1.2 Objetivos

1.2.1 Objetivo Geral

Desenvolver um *software* para a gestão centralizada e especializada de *logs* de aplicações, com funcionalidades capazes de detectar atividades não autorizadas ou em não conformidade com as políticas de segurança da informação, possibilitando que problemas sejam identificados e solucionados de maneira ágil.

1.2.2 Objetivos Específicos

- Possibilitar que registros de *logs* sejam armazenados, indexados e monitorados em um repositório central;
- Utilizar registros de *logs* para assegurar que os problemas de sistemas de informação sejam identificados;
- Armazenar registros de *logs* contendo atividades dos usuários, exceções do sistema ou qualquer outro evento de segurança da informação por um período de tempo determinado pela organização, possibilitando o seu uso em futuras investigações;

- Adotar medidas de proteção de privacidade para que os *logs* com dados pessoais ou confidenciais sejam armazenados e monitorados de forma segura;
- Certificar que os registros de *logs* sejam protegidos contra falsificação e acesso não autorizado para não causar falsa impressão de segurança;
- Oferecer mecanismo de análise dos registros de *logs* para facilitar a compreensão de problemas encontrados, a maneira pela qual ele pode ocorrer e qual a melhor ação a se adotar;
- Desenvolver a solução adotando a modalidade de *software Open Source*;

2 REVISÃO BIBLIOGRÁFICA

O presente trabalho trata do desenvolvimento de um *software* para a gestão centralizada e especializada de *logs* de aplicações, com funcionalidades que visam proporcionar segurança da informação, detectando atividades não autorizadas ou em não conformidade com as políticas de segurança da informação. Para compreender e fundamentar a importância do desenvolvimento desta solução, a revisão bibliográfica aborda assuntos relacionados a evolução e importância da informação, sistemas de informação, classificação e ciclo de vida informação, segurança da informação (ambiente físico, lógico e aspecto humano), administração estratégica, políticas e planos de segurança da informação, normas, desenvolvimento seguro de *software* e por fim a fundamentação de maior destaque do projeto: registros de eventos (*logs*).

2.1 Evolução e Importância da Informação

A história da humanidade se caracteriza por três grandes e fundamentais revoluções ao longo de sua evolução. A primeira foi a Revolução Agrícola de aproximadamente 10.000 anos atrás, a segunda foi a Revolução Industrial iniciada logo após o surgimento da máquina a vapor em 1776, substituindo o trabalho artesanal pela produção em massa. E a terceira grande mudança trata-se da Revolução da Informação, esta teve seu início na década de 40 com o surgimento do primeiro computador e está em curso até hoje. A Revolução da Informação se caracteriza pelo rápido avanço das tecnologias da informática e das telecomunicações e tem proporcionado mudanças consideráveis na forma de armazenar, processar e recuperar informações (CAVALCANTI, 1995).

A rápida expansão da Revolução da Informação e seus diversos componentes tornou necessária a criação de uma ciência que tivesse por objeto de estudo a informação, ou seja, a ciência da informação, além de tecnologias e técnicas resultantes das descobertas realizadas por esta nova ciência. Esta revolução ocasionou uma relação muito forte entre ciência da informação, tecnologia da informação e sociedade, a tal ponto que acabou por assumir seu nome transformando-a em “Sociedade da Informação”, abandonando qualitativos materiais de

“Industrial” e “Pós-Industrial” herdados do século passado, inaugurando assim, a Era da Informação (LE COADIC, 1996).

Com o advento da Era da Informação desenvolve-se o conceito de sociedade do conhecimento como novo paradigma socioeconômico. Neste novo paradigma a informação atua de forma intensa em nossas vidas, o mundo passa a ser uma economia global e interdependente capaz de gerar um fluxo inimaginável de informações. Neste novo conceito que surge, o real valor dos produtos está no conhecimento neles embutidos, a economia adota uma estrutura mais diversa, alterando-se contínua e rapidamente. Neste novo cenário o poderio econômico internacional de um país ou de uma empresa está diretamente relacionado ao fator conhecimento (BORGES, 1995).

Com este novo panorama econômico a informação ganha destaque sendo um fator determinante para o sucesso de uma organização. A informação passa a ser utilizada como recurso gerencial capaz de garantir o funcionamento tático, estratégico e operacional. Neste ambiente empresarial evoluído a tecnologia da informação se consolida como ferramenta gerencial capaz de analisar dados, transformando-os em informações realmente úteis aos negócios das empresas. Desta forma, à medida que os dados são convertidos em informações, modificarão necessariamente os processos de decisão, a estrutura administrativa e a maneira de trabalhar, onde decisões financeiras oportunistas transforam-se em diretrizes e pressupostos estratégicos (BORGES, 1995).

De acordo com Lesca e Almeida (1994) “A informação é um vetor estratégico importantíssimo, pois pode multiplicar a sinergia dos esforços ou anular o resultado do conjunto dos esforços”, ou seja, a informação é essencial não apenas para controle, mas para outras funções administrativas, como tomadas de decisões, planejamento estratégico, desenvolvimento de produtos etc. A probabilidade de acerto em uma decisão sem uma base em informações é praticamente nula, as empresas inseridas neste novo contexto econômico não podem depender da sorte para a tomada de decisão, principalmente em um mercado cada vez mais competitivo (CAVALCANTI, 1995).

Estar preparado na Era da Informação é fator determinante para o sucesso econômico. Desta forma, a administração estratégica da informação se tornou perceptível para as organizações com base em três hipóteses. A primeira é que as empresas que desenvolvem a administração da informação de maneira eficaz estão no grupo de melhor desempenho e dominam a concorrência. A segunda é que nas empresas onde não existe a administração da informação se desenvolve um processo de degradação do desempenho, sem haver percepção dessa ocorrência, fazendo com que se tornem presas fáceis para a concorrência. A terceira é

que uma empresa pode melhorar significativamente seu desempenho a partir do desenvolvimento de um processo de administração da informação com orientação estratégica, com o objetivo de obter vantagem competitiva (LESCA; ALMEIDA, 1994).

2.2 Sistemas de Informação

A principal premissa dos sistemas de informação é gerar informações destinadas a apoiar a tomada de decisões, coordenando e controlando uma organização, além de oferecer suporte aos gerentes e trabalhadores a analisar problemas, visualizar assuntos complexos e criar novos produtos. STAIR (1998, p. 11), afirma que: "... sistemas de informação é uma série de elementos ou componentes inter-relacionados que coletam (entrada), manipulam e armazenam (processo), disseminam (saída) os dados e informações e fornecem um mecanismo de feedback". GIL (1999, p.14), define que "... os sistemas de informação compreendem um conjunto de recursos humanos, materiais, tecnológicos e financeiros agregados segundo uma sequência lógica para o processamento dos dados e a correspondente tradução em informações".

Todos estes componentes inter-relacionados possuem como objetivo lidar de forma mais eficaz com os problemas internos e externos ao ambiente a qual estão inseridas, as organizações buscam desenvolver sistemas de informações como suporte a resolução destes problemas. Neste sentido, LAUDON e LAUDON (1999, p. 26), afirmam que "a razão mais forte pelas quais as empresas constroem os sistemas, então, é para resolver problemas organizacionais e para reagir a uma mudança no ambiente".

Portanto, os sistemas de informação contêm informações significativas para as organizações ou o ambiente que a cerca e são mecanismos de apoio à gestão, tendo em sua base a tecnologia da informação, sendo que este suporte da informática atua como condutor das informações que visam facilitar, agilizar e otimizar o processo decisório. Eles são parte integrante das organizações, em alguns casos sem sistemas de informação não haveria negócios (PEREIRA e FONSECA 1997).

2.3 Classificações da Informação

Devido a grande quantidade de informações que precisam ser administradas a todo o momento, muitas vezes se torna difícil classificar toda informação como crucial ou essencial a ponto de merecer cuidados especiais. Por outro lado, determinada informação pode ser tão importante que o custo de sua integridade ainda será menor que o custo de não dispor dela de forma adequada. Desta forma, surge a necessidade de classificar a informação em níveis de prioridade, respeitando a necessidade de cada organização e a importância da classe de informação para manutenção das atividades do dia-a-dia (LAUREANO, 2005).

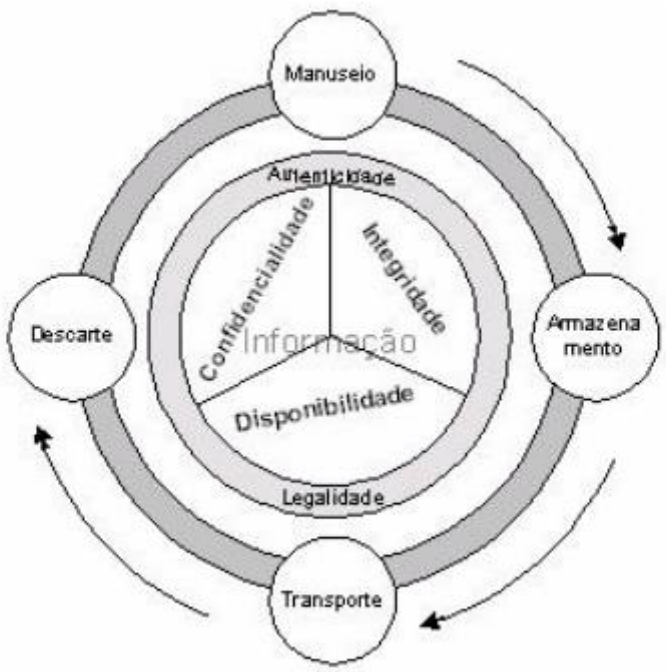
Segundo (Abreu, 2001; Boran, 1996; Wadlow, 2000), ao identificar os responsáveis pelas informações, esta poderá ser mais facilmente classificada de acordo com sua sensibilidade e importância podendo ser agrupadas em quatro níveis:

- **Pública:** informação que pode vir ao público sem maiores consequências danosas ao funcionamento normal da empresa, e cuja integridade não é vital;
- **Interna:** o acesso a esse tipo de informação deve ser evitado, embora as consequências do uso não autorizado não sejam sérias. Sua integridade é importante, mesmo que não seja vital;
- **Confidencial:** informação restrita aos limites da empresa, cuja divulgação ou perda pode levar ao desequilíbrio operacional, e eventualmente, perdas financeiras ou de confiabilidade perante o cliente externo, além de permitir vantagem expressiva ao concorrente;
- **Secreta:** informação crítica para as atividades da empresa, cuja integridade deve ser preservada a qualquer custo e cujo acesso deve ser restrito a um número bastante reduzido de pessoas. A manipulação desse tipo de informação é vital para a companhia.

Este processo de classificação deve ser orientado por definições claras dos diversos graus de sensibilidade estabelecidos pela organização, bem como pela definição exata dos responsáveis pela classificação. A classificação deve iniciar mediante estas definições, mas deve ser definido também o processo de desclassificação da informação que vai auxiliar no momento em que uma determinada informação descerá a escala de classificação (SILVA; CARVALHO; TORRES, 2003).

2.4 Ciclo de Vida da Informação

Todo e qualquer momento vivido pela informação, até mesmo situações de riscos, é designado de ciclo de vida da informação. Sobre isso Laureano (2005) coloca que "Os momentos são vivenciados justamente quando os ativos físicos, tecnológicos e humanos fazem uso da informação, sustentando processos que, por sua vez, mantêm a operação da empresa.". Existem quatro momentos do ciclo de vida da informação (Figura 1) que são merecedores de atenção, pois correspondem às situações onde a informação é exposta a ameaças colocando em risco sua integridade (LAUREANO, 2005).



Fonte: Laureano (2005).

Figura 1 - Ciclo de vida da informação

Conforme pode ser observado na Figura 1, o manuseio corresponde ao momento em que a informação é criada e manipulada, seja ao folhar papéis, digitar informações recebidas, ou até senha de acesso para autenticação. O armazenamento corresponde ao momento em que a informação é guardada, seja em banco de dados, anotações em papel, mídia óptica etc. O

transporte corresponde ao momento em que a informação é transportada, seja por correio eletrônico, postal, telefone etc. E por fim o descarte, momento em que a informação já não é mais útil, podendo ser depositada em uma lixeira, apagada do banco de dados etc. (LAUREANO, 2005).

2.5 Segurança da Informação

A informação é um bem de grande valor e como qualquer outro ativo de uma organização é essencial para os negócios e consequentemente deve ser protegida. E isto se torna importante ao passo que cada vez mais o mundo se encontra interconectado. Como resultado desta interconexão as organizações e até mesmo as pessoas estão expostas a um crescente número e uma grande diversidade de vulnerabilidades. A informação existe em diversas formas e não importa qual for esta forma ou o meio onde ela é compartilhada é necessário que ela seja protegida de forma adequada (NBR ISO/IEC 17799:2005, 2005).

Sêmola (2003) define segurança da informação como "uma área do conhecimento dedicada à proteção de ativos da informação contra acessos não autorizados, alterações indevidas ou sua indisponibilidade." A NBR ISO/IEC 17799:2005 (2005), em sua seção introdutória, define segurança da informação como "a proteção da informação de vários tipos de ameaças para garantir a continuidade do negócio, minimizar o risco ao negócio, maximizar o retorno sobre os investimentos e as oportunidades de negócio". Desta forma, é possível definir segurança da informação como a área do conhecimento que visa proteger a informação de ameaças a sua confidencialidade, disponibilidade e integridade.

Conforme (NBR ISO/IEC 17799:2003, 2003; Krause e Tipton, 1999; Albuquerque e Ribeiro, 2002) estes princípios básicos possuem como objetivo:

- **Confidencialidade:** a informação somente pode ser acessada por pessoas explicitamente autorizadas. É a proteção de sistemas de informação para impedir que pessoas não autorizadas tenham acesso ao mesmo. O aspecto mais importante deste item é garantir a identificação e autenticação das partes envolvidas;
- **Disponibilidade:** a informação ou sistema de computador deve estar disponível sempre que haver uma requisição;

- **Integridade:** a informação deve ser retornada em sua forma original no momento em que foi armazenada. É a proteção dos dados ou informações contra modificações intencionais ou acidentais não autorizadas.

O tema segurança da informação a cada dia vem ganhando mais destaque na medida em que as empresas possuem cada vez mais informações processadas e armazenadas em um ambiente computacional responsável por realizar e consolidar negócios. Cada organização define uma estrutura adequada para a proteção da informação, os regulamentos (políticas, normas e regras) têm como objetivo garantir que o acesso e uso da informação aconteçam de forma estruturada, possibilitando desta forma que o negócio não seja comprometido por mau uso desta informação, seja por erro ou por acidente (FONTES, 2006).

Para proteger a informação as medidas de segurança devem ser implementadas antes da concretização do risco, ou seja, antes do incidente ocorrer. Estas medidas são classificadas em função da forma como abordam as ameaças, em duas grandes categorias: prevenção e proteção (SILVA; CARVALHO; TORRES, 2003). Silva, Carvalho e Torres (2003) explicam que "A prevenção é o conjunto das medidas que visam reduzir a probabilidade de concretização das ameaças existentes." e a "A proteção, por seu lado, é o conjunto das medidas que visam dotar os sistemas de informação com capacidade de inspeção, detecção, reação e reflexo, permitindo reduzir e limitar o impacto das ameaças quando estas se concretizam.".

Para que a informação esteja realmente protegida alguns autores (Dias, 2000; Wadlow, 2000; Shirey, 2000; Krause e Tipton, 1999; Albuquerque e Ribeiro, 2002; Sêmola, 2003; Sandhu e Samarati, 1994) argumentam que o sistema que esta administrando esta informação deve respeitar:

- **Autenticidade:** garante que a informação ou o usuário da mesma é autêntico. Atesta com exatidão, a origem do dado ou informação;
- **Não repúdio:** não é possível negar (no sentido de dizer que não foi feito) uma operação ou serviço que modificou ou criou uma informação. Não é possível negar o envio ou recepção de uma informação ou dado;
- **Legalidade:** garante a legalidade (jurídica) da informação. Aderência de um sistema à legislação. Característica das informações que possuem valor legal dentro de um processo de comunicação, onde todos os ativos estão de acordo com as cláusulas contratuais pactuadas ou a legislação política institucional, nacional ou internacional vigentes;

- **Privacidade:** foge do aspecto de confidencialidade, pois uma informação pode ser considerada confidencial, mas não privada. Uma informação privada deve ser vista / lida / alterada somente pelo seu dono. Garante ainda, que a informação não será disponibilizada para outras pessoas (neste caso é atribuído o caráter de confidencialidade a informação). É a capacidade de um usuário realizar ações em um sistema sem que seja identificado;
- **Auditoria:** rastreabilidade dos diversos passos que um negócio ou processo realizou ou que uma informação foi submetida, identificando os participantes, os locais e horários de cada etapa. Auditoria em *software* significa uma parte da aplicação, ou conjunto de funções do sistema, que viabiliza uma auditoria. A auditoria consiste no exame do histórico dos eventos dentro de um sistema para determinar quando e onde ocorreu uma violação de segurança.

É necessário proteger a informação da empresa, pois ela é o sangue que move toda a organização. A segurança da informação existe para possibilitar uma diminuição dos riscos do negócio em relação à dependência do uso dos recursos de informação para o funcionamento tático e estratégico da empresa. Não possuir a informação ou possuir a informação errada pode trazer perdas aos negócios da organização que comprometam o seu funcionamento e o retorno de investimentos (FONTES, 2006).

2.5.1 Segurança do Ambiente Físico

O ambiente físico abrange todo o ambiente onde os sistemas de informação estão instalados como prédios, portas de acesso, pisos, salas, entre outros. A NBR ISO/IEC 17799:2005 (2005) define que o objetivo de um ambiente físico seguro é “Prevenir o acesso físico não autorizado, danos e interferências com as instalações e informações da organização.”.

A todo o momento os ativos físicos são alvos de ameaças que buscam identificar um elo fraco para poder realizar sua ação. Neste sentido, a NBR ISO/IEC 17799:2005 (2005) coloca que para atingir um bom nível de segurança física e do ambiente “Convém que as instalações de processamento da informação críticas ou sensíveis sejam mantidas em áreas seguras, protegidas por perímetros de segurança definidos, com barreiras de segurança e controles de acesso apropriados.”. Por meio destes procedimentos é possível proteger

fisicamente a informação contra o acesso não autorizado, danos e interferências com métodos compatíveis aos riscos identificados.

2.5.2 Segurança do Ambiente Lógico

O ambiente lógico compreende todos os mecanismos de proteção que são baseados em *softwares* como criptografia, senhas, listas de acessos, *firewall*, redes privadas, etc. De acordo com Adachi (2004), é nessa camada que estão as "regras, normas, protocolo de comunicação e onde, efetivamente, ocorrem as transações e consultas". Desta forma, o objetivo da adoção de medidas de segurança no ambiente lógico é proteger os dados, programas e sistemas contra tentativas de acessos, tanto de usuários como de outros sistemas, não autorizados.

A segurança do ambiente lógico remete ao acesso que os indivíduos têm em um ambiente informatizado, onde apenas usuários autorizados devem ter acesso aos recursos computacionais. Além disso, estes usuários devem acessar somente os recursos realmente necessários para a execução de suas tarefas, recursos críticos devem ser monitorados e restritos, impedindo o usuário de executar alguma ação incompatível com seu nível de segurança (CARUSO e STEFFEN, 1999).

2.5.3 Aspectos Humanos da Segurança da Informação

O aspecto humano trata-se do elemento central de um sistema de segurança da informação. De acordo com Lyra (2008) "Os incidentes de segurança sempre envolvem pessoas, quer do lado das vulnerabilidades exploradas, quer do lado das ameaças que exploram essas vulnerabilidades.", ou seja, as pessoas devem ter uma relevância considerada em um sistema de gestão de segurança (LYRA, 2008).

O aspecto humano é formado por todos os recursos humanos presentes em uma organização, com maior destaque para aquelas que possuem acesso aos recursos de tecnologia da informação, seja este acesso para manutenção ou uso na execução de suas tarefas diárias. A percepção do risco pelas pessoas (como elas lidam com incidentes de segurança), a habilidade dos usuários com a TI, o perigo com intrusos maliciosos e a engenharia social (arte de utilizar

o comportamento humano para quebrar a segurança sem que a vítima perceba que foi manipulada), são pontos importantes e de destaque na busca pela segurança da informação por meio do aspecto humano (ADACHI, 2004).

2.6 Administração Estratégica da Segurança da Informação

O termo estratégia pode ser compreendido como o conhecimento necessário para a consolidação de planos hábeis que proporcionam uma posição privilegiada para a organização. Neste sentido, a organização tem em suas mãos, uma perspectiva racional para o ato estratégico, definindo-o com coerência, possibilitando realizar o plano de forma hábil e eficaz (SACCONI, 1998).

Atualmente, os gestores organizacionais buscam garantir que seus produtos, serviços e ativos, sejam bem sucedidos no mercado. Desta forma, a estratégia organizacional tem papel determinante, pois possibilita tomar decisões fundamentadas, decisões que ponderem as relações internas e externas à organização. A estratégia organizacional é também uma forma de criar uma cultura organizacional nas atitudes dos administradores, elas são particulares de cada organização e se adaptam ao meio em que estão inseridas (LAUREANO; MORAES, 2005).

A estratégia organizacional deve ser capaz de garantir a continuidade dos negócios e oferecer métodos capazes de proteger e garantir a integridade das informações geradas pela organização, pois a informação é um ativo que, como qualquer outro ativo é importante para os negócios, tem um valor para a organização e, consequentemente, necessita ser adequadamente protegida (NBR ISO/IEC 17799, 2003). Além disso, a informação faz parte da inteligência competitiva e deve ser administrada em seus particulares, diferenciada e salvaguarda. Ela é um recurso vital para a definição de estratégias alternativas capazes de constituir uma organização mais flexível onde o aprendizado é constante (LAUREANO; MORAES, 2005).

2.6.1 Política de Segurança da Informação

De acordo com a Academia Latino Americana de Segurança da Informação (2013):

> "Uma política de segurança é um conjunto de diretivas, normas, procedimentos e instruções que comanda as atuações de trabalho e define os critérios de segurança para que sejam adotados em nível local ou institucional com o objetivo de estabelecer, padronizar e normalizar a segurança tanto no escopo humano como no tecnológico. A partir desses princípios, é possível fazer da segurança das informações um esforço comum, desde que todos possam contar com um arsenal informativo documentado e normalizado dedicado à padronização do método de operação de cada um dos indivíduos envolvidos na gestão da segurança da informação".

O objetivo principal da política de segurança da informação é oferecer subsídios para que seja possível alcança a segurança da informação. Segundo a NBR ISO/IEC17799 (2003) "Convém que a direção estabeleça uma política clara e demonstre apoio e comprometimento com a segurança da informação através da emissão e manutenção de uma política de segurança da informação para toda a organização". Ou seja, para que a política seja implantada faz-se necessário comunicar e publicar de maneira adequada para todos os colaboradores da organização.

As politicas de segurança devem conter informações claras sobre o comportamento do colaborador ao guardar informações da organização, neste sentido, elas são fundamentais no desenvolvimento de controles efetivos para contra-atacar possíveis ameaças à segurança e integridade da informação. Essas políticas ganham seu devido destaque principalmente no que diz respeito a evitar e detectar ataques provenientes da engenharia social (FONSECA, 2009).

2.6.2 Os Planos da Segurança

As políticas de segurança são apresentadas como códigos de conduta aos quais as pessoas devem se adequar para garantir a integridade e segurança da informação. Além disso, a necessidade de uma segurança da informação efetiva deve estar alinhada com a necessidade e capacidade da organização, desta forma, é importante definir de forma clara e objetiva os requisitos almejados para satisfazê-la. É neste momento que entram em cena os planos de segurança. Eles são capazes de proporcionar um gerenciamento organizado da segurança da

informação (LYRA, 2008). A seguir são expostos alguns destes planos apresentados por Lyra (2008).

2.6.2.1 Plano Diretor de Segurança

O Plano Diretor de Segurança (PDS) trata-se do ponto de partida para quem pretende gerir a segurança da informação de forma organizada. É caracterizado por ser um documento dinâmico e flexível, capaz de se adequar às novas necessidades de segurança quem possam surgir no contexto organizacional. Este plano deve fornecer informações sobre como a organização deve agir frente à segurança da informação, proporcionado o seu funcionamento sob-risco controlado e em nível tolerado. Cada organização deve produzir um PDS voltado para suas necessidades, ameaças, vulnerabilidades e exposição a riscos, com o objetivo de montar um mapa de relacionamento e dependência entre processos de negócios, aplicações e infraestrutura física, tecnológica e humana (LYRA, 2008).

2.6.2.2 Plano de Continuidade de Negócios

O Plano de Continuidade de Negócios (PCN) é composto por um conjunto de procedimentos previamente definidos e testados para garantir a continuidade dos processos e serviços vitais de uma organização, mesmo que sob impacto de um desastre inesperado, previamente identificado, ou seja, é voltado para a manutenção das funções críticas do negócio de uma organização durante e após a ocorrência de algum tipo de sinistro, assegurando a continuidade das atividades exercidas por cada processo dentro da organização. O PCN deve garantir a segurança dos colaboradores e visitantes, minimizar danos imediatos e perdas em uma situação de emergência, assegurar a restauração das atividades, instalações e equipamentos o mais rápido possível, assegurar a rápida ativação dos processos de negócios críticos e fornecer conscientização e treinamento para as pessoas envolvidas nos processos organizacionais (LYRA, 2008).

2.6.2.3 Plano de Contingência

O Plano de Contingência deve ser elaborado para cada ameaça considerada em cada um dos processos do negócio pertencentes ao escopo, definindo em detalhes os procedimentos que devem ser executados em um estado de contingência. Ele deve ser capaz de detalhar procedimentos e capacidades para a recuperação de uma aplicação específica ou sistemas mais complexos. Por meio das estratégias de contingência é possível adotar métodos e procedimentos que auxiliam a tomada de decisão quando uma situação de risco ocorrer (LYRA, 2008).

2.6.2.4 Plano de Administração de Crise

Este plano possui como premissa definir, de forma detalhada, o funcionamento das equipes envolvidas com o acionamento da contingência antes, durante e depois da ocorrência de algum incidente que coloca em risco a segurança da informação. Além disso, o Plano de Administração de Crise deve definir os procedimentos que serão executados pela mesma equipe no período de retorno a normalidade. Como exemplo deste plano é possível citar o comportamento da organização na comunicação de um fato à imprensa (LYRA, 2008).

2.6.2.5 Plano de Continuidade Operacional

O Plano de Continuidade Operacional possui como premissa, definir os procedimentos para a contingência dos ativos que suportam cada processo de negócio, objetivando diminuir o tempo de indisponibilidade e, consequentemente, os impactos potenciais ao negócio. A orientação para a execução de ações diante de uma queda da *Internet* exemplifica os desafios organizados por este plano (LYRA, 2008).

2.6.2.6 Plano de Recuperação de Desastres

Este plano é incumbido de definir um plano de recuperação e restauração das funcionalidades dos ativos afetados que suportam os processos de negócio, com o objetivo de restabelecer o ambiente e as condições originais de operação. O Plano de Recuperação de Desastres serve ao propósito de juntar todos os detalhes do processo de negócio, sendo este nível de detalhe vital porque, no caso de uma emergência, o plano pode ser a única coisa que restou do seu centro de dados (*backups*, por exemplo) para ajudar na reconstrução e restauração das operações (LYRA, 2008).

2.7 Normas da Segurança da Informação

Segundo Beal (2005, p.36) as "Normas e padrões têm por objetivo definir regras, princípios e critérios, registrar as melhores práticas e prover uniformidade e qualidade a processos, produtos ou serviços, tendo em vista sua eficiência e eficácia." Concomitantemente, Sêmola (2003) diz que "uma norma tem o propósito de definir regras, padrões e instrumentos de controle que dêem uniformidade a um processo, produto ou serviço". Ou seja, as normas são procedimentos e regras que visam garantir que um processo seja feito sempre de forma igual e da maneira mais correta possível.

A rápida expansão da TI aliada a constante preocupação em garantir a integridade das informações fez surgir normas capazes de garantir a segurança da informação. Esta expansão fez com que houvesse um interesse internacional em uma norma de segurança da informação, desta forma, em dezembro de 2000, foi publicada a norma internacional ISO 17799:2000 (OLIVA; OLIVEIRA, 2003).

Em 2001, a Associação Brasileira de Normas Técnicas (ABNT) publicou a versão brasileira que ficou denominada como NBR/ISO 17799 – Código de Prática para a Gestão da Segurança da Informação (OLIVA; OLIVEIRA, 2003). Em setembro de 2005, a norma foi revisada e publicada como NBR ISO/IEC 17799:2005. (ISO 17799, 2005). O comitê que trata da segurança da informação na ISO aprovou a criação de uma família de normas sobre gestão da segurança da informação, denominada pela série 27000, onde a então ISO IEC 17799:2005 foi renomeada para ISO IEC 27002:2005 (HOLANDA, 2006).

2.7.1 ISO 17799

Esta norma trata-se de um código de práticas com orientações para a gestão da segurança da informação. Possui como característica descrever de forma geral as áreas consideradas importantes de implantação ou gestão de segurança da informação dentro de uma organização (PARRA, 2002).

De acordo com Lyra (2008), a ISO 17799 "é um código de prática de gestão de segurança da informação e sua importância pode ser dimensionada pelo número crescente de pessoas e variedades de ameaças a que a informação é exposta". Estabelecer um referencial para as organizações desenvolverem, implementarem e avaliarem a gestão da tecnologia da informação e promover a confiança nas transações comerciais entre as organizações são os objetivos explícitos desta norma (LYRA, 2008).

Segundo Parra (2002) a norma ISO 17799 está dividida em 12 tópicos:

1. Objetivo;
2. Termos e definições;
3. Política de segurança;
4. Segurança organizacional;
5. Classificação e controle dos ativos de informação;
6. Segurança de recursos humanos;
7. Segurança física e do ambiente;
8. Gerenciamento das operações e comunicações;
9. Controle de acessos;
10. Desenvolvimento e manutenção de sistemas de informação;
11. Gestão de continuidade de negócios;
12. Conformidade.

Estes 12 tópicos são macros áreas de gestão da segurança da informação e esta divisão ocorre para que a gestão seja feita da forma mais eficaz possível. Se não houvesse esta divisão e a segurança fosse tratada como um todo, os controles não seriam tão efetivos o que certamente refletiria em uma má gestão da segurança da informação. Por meio destes tópicos é possível realizar uma implantação em estágios, ou seja, pode ser implantada a política de segurança e depois a segurança organizacional, ou vice-versa, para depois colocar em prática as demais orientações da norma. Dependendo do tamanho da organização é possível realizar a implantação de apenas alguns tópicos desta norma (PARRA, 2002).

2.7.2 ISO 27001

Trata-se da única norma internacional auditável que define os requisitos para um Sistema de Gestão de Segurança da Informação (SGSI). Possui como premissa assegurar a seleção de controles de segurança adequados e proporcionais, ajudando a empresa a proteger seus ativos da informação e dar confiança para todas as partes interessadas, de forma especial aos clientes. A ISO 27001 adota uma abordagem de processo que engloba a implementação, operação, monitoramento, revisão, manutenção e melhoria de um SGSI (BSI BRASIL, 2013).

A norma define sua abordagem na aplicação de um sistema de processos em uma organização, junto com a identificação e interações desses processos e sua correta gestão. Desta forma, a ISO 27001 emprega o PDCA, *Plan-Do-Check-Act* (ciclo de desenvolvimento que tem foco na melhoria contínua) para estruturar os processos. O ciclo PDCA proporciona uma definição do escopo do sistema de gerenciamento de segurança da informação, planos para tratamento de riscos, implementação de controles, medição da eficácia dos controles, monitoramento e controle, revisões periódicas no sistema de segurança e a implementação de melhorias identificadas (THE ISO 27000 DIRECTORY, 2013).

A ISO 27001 é aplicável para qualquer organização, grande ou pequena, em qualquer setor ou parte do mundo, sendo mais utilizada onde a proteção da informação é crítica, como em finanças, saúde, setores públicos e de TI. É altamente eficaz para as organizações que gerenciam informação em nome de terceiros, bem como, companhias terceirizadas de TI, pois ela pode ser usada para garantir aos clientes que suas informações estão sendo protegidas (BSI BRASIL, 2013).

2.7.3 ISO 27002

A norma ISO 27002 é a renomeação da norma ISO 17799 com a inclusão de mais alguns procedimentos de segurança, e como exposto anteriormente, trata-se de um código de práticas para garantir a integridade e segurança da informação. Ela basicamente descreve mecanismos potenciais de controle que podem ser implementados conforme orientação fornecida na ISO 27001. Estes controles são destinados a abordar as necessidades específicas identificadas através de uma avaliação de risco formal. A norma também se destina a fornecer

um guia para o desenvolvimento de padrões de segurança organizacional e práticas de gestão eficazes de segurança, além de ajudar a construir a confiança em atividades inter-organizacionais (THE ISO 27000 DIRECTORY, 2013).

2.8 Auditoria em Sistemas de Informação

A auditoria em sistemas de informação possui como característica principal o foco na análise e avaliação. Estes dois pontos são utilizados tanto em processos de planejamento, desenvolvimento, testes e aplicações de sistemas, como na verificação da estrutura lógica, física, ambiental, organizacional, segurança e proteção da informação. Ou seja, a auditoria de sistemas não possui como objetivo apenas a função da informática, mas também todos os sistemas de informação, informatizados ou não que estão presentes no dia-a-dia organizacional (CARNEIRO, 2004).

A função da auditoria de sistemas de informação é possibilitar uma adequação, revisão, avaliação e recomendações para o aprimoramento dos controles internos nos sistemas de informação, visando avaliar a utilização de recursos humanos e tecnológicos. Ela deve atuar em todos os sistemas da organização, seja no nível operacional, tático ou estratégico e deve possuir seus objetivos pautados em: efetividade, eficiência, confidencialidade, integridade, disponibilidade, *compliance*[2] e confiança (LYRA, 2008).

Os objetivos globais da auditoria de sistemas de informação, de acordo com Lyra (2008), se resumem em:

- **Integridade:** o sistema deve garantir a consistência e confiança nas transações processadas;
- **Confidencialidade:** as informações são reveladas somente às pessoas que necessitam conhece-las;
- **Privacidade:** as funções incompatíveis nos sistemas são segregadas;
- **Acuidade:** as transações processadas podem ser validadas;
- **Disponibilidade:** o sistema deve estar disponível para cumprimentos dos objetivos organizacionais;

[2] Conjunto de disciplinas para fazer cumprir as normas legais e regulamentares.

- **Auditabilidade:** os sistemas devem documentas *logs* operacionais que permitam trilhas de auditoria;
- **Versatilidade:** o sistema deve ser amigável, de fácil adaptação ao fluxo operacional da organização;
- **Manutenibilidade:** políticas e procedimentos operacionais devem contemplar controles quanto a teste, conversão, implantação e documentação de sistemas novos ou modificados.

Para a correta análise e avaliação, a auditoria de sistemas é dividida em modalidades que tendem a tornar mais eficaz sua implantação. A auditoria durante o desenvolvimento de sistemas compreende auditar todo o processo de construção de sistemas de informação, da fase de requisitos até a sua implantação. A auditoria em sistemas em produção preocupa-se com os procedimentos e resultados dos sistemas já implantados, na segurança, corretude e tolerância a falhas. A auditoria no ambiente tecnológico compreende a análise do ambiente de informática em termos de estrutura organizacional, contratos, normas técnicas, custos, nível de utilização de equipamentos e planos de segurança e de contingência. E por fim, a auditoria em eventos específicos que compreende a análise das causas, consequências e ações corretivas cabíveis em eventos não cobertos pelas auditorias anteriores (LYRA, 2008).

2.9 Segurança no Desenvolvimento de *Softwares*

A cada dia um número maior de computadores está conectado a *Internet* e junto a esta expansão crescem também os ataques aos sistemas informatizados, colocando *softwares*, organizações e usuários em grande risco. Mesmo as organizações adotando mecanismos de controles e políticas de segurança, o *software* sempre foi e será o vetor de ataques (REGO; BROSSO, 2013).

Oferecer um *software* seguro é obrigação do desenvolvedor e para que este objetivo seja alcançado é necessário avaliar a segurança de todo o ciclo de vida de desenvolvimento da aplicação. Esta avaliação melhora os produtos de TI, pois identifica erros ou vulnerabilidades que podem ser corrigidas pelo desenvolvedor, reduzindo a probabilidade de futuras falhas de segurança, além disso, faz o desenvolvedor tomar mais cuidado com a estrutura e desenvolvimento do *software* (BRASIL, 2013a).

Para ter segurança no desenvolvimento é necessário manter fontes em segurança, evitando roubos ou indisponibilidade da equipe de desenvolvimento, seguir especificações de segurança evitando vulnerabilidades que comprometam a segurança e oferecer garantias aos clientes quanto à segurança e integridade da aplicação (BRASIL, 2013a).

Um ambiente de desenvolvimento seguro se caracteriza por ter um espaço físico restrito, com controle de acesso físico e proteção lógica dos servidores, separação entre ambiente de desenvolvimento, teste e construção (*build*), gerência de configuração dos fontes e processos de desenvolvimento bem estabelecidos e controlados (BRASIL, 2013a).

A segurança no desenvolvimento do *software* é alcançada ao se adotar normas e práticas de boa programação como funções intrinsecamente seguras, verificar os códigos de erro retornado por funções ou métodos, atentar para o tamanho de *buffers* e *arrays* do *software* e documentar o código. A garantia desta segurança é obtida com a construção do *software* conforme o especificado e na execução e validação de testes para verificar se atende às especificações iniciais (BRASIL, 2013a).

A segurança no desenvolvimento de *softwares* está apoiada na norma ISO/IEC 15.408 (*Common Criteria for Information Technology Security Evaluation*) também chamada de *Common Criteria* ou CC. Esta norma trata-se de uma metodologia para desenvolvimento de *software* seguro e possui como objetivo fornecer um conjunto de critérios fixos que permitem especificar a segurança da aplicação de forma não ambígua a partir de características do ambiente e definir formas de garantir a segurança do sistema para o cliente final, sendo definida em quatro níveis de garantia da segurança (*EAL – Evaluation Assurance Level*): EAL 1 – testado funcionalmente; EAL 2 – testado estruturalmente; EAL 3 – metodicamente testado e verificado e EAL 4 – metodicamente projetado, testado e verificado (BRASIL, 2013a).

2.10 Registros de Eventos (*Logs*)

De acordo com Brasil (2013b) “Log é o registro de atividade gerado por programas e serviços de um computador. Ele pode ficar armazenado em arquivos, na memória do computador ou em bases de dados”. Clemente (2008) argumenta que *log* “é [...] um conjunto de registros com marcação temporal, que suporta apenas inserção, e que representa eventos que aconteceram em um computador ou equipamento de rede”. Brasil (2013b) também afirma que “Os logs são registros de atividades gerados por programas de computador” e destaca que

“No caso de logs relativos a incidentes de segurança, eles normalmente são gerados por firewalls ou por sistemas de detecção de intrusão”. Ou seja, são registros de eventos do sistema operacional, redes, aplicações e de sistemas informatizados específicos, capazes de prover informações vitais para a notificação de incidentes no ambiente computacional.

Os registros de *log* constituem uma fonte básica de informação tanto para a detecção, como para a resolução de problemas. Com a análise destas informações providas pelos *logs* é possível detectar o uso indevido do ambiente de TI, ataques, exploração de vulnerabilidades, rastrear (auditar) as ações executadas por um usuário e detectar problemas de *hardware* ou nos programas e serviços instalados no computador. Com base nestas informações é possível tomar medidas preventivas para tentar evitar que um problema maior ocorra ou, caso não seja possível, tentar reduzir os danos (BRASIL, 2013b).

2.10.1 Geração de Arquivos de *Logs*

Geração de *logs* é o procedimento pelo qual um sistema operacional ou aplicativo registra eventos à medida que eles acontecem e preserva esses registros para posterior análise. Os registros de *log* fornecem a única evidência real de que um crime realmente aconteceu (DALLMANN, 2005). Os arquivos de *logs* são essenciais para a notificação de incidentes, pois são capazes de armazenar diversas informações importantes, como a data e o horário em que uma determinada atividade ocorreu, o fuso horário, o endereço IP de origem da atividade, os dados completos que foram enviados, alterados ou excluídos e o resultado da atividade (se ela ocorreu com sucesso ou não) (BRASIL, 2013b).

De acordo com Brasil (2003a), a geração de *logs* é a “Geração de registros de eventos ou estatísticas para prover informações sobre a utilização e performance de um sistema”, ou seja, são muito importantes para a administração segura de sistemas, pois possuem como premissa registrar informações sobre o seu funcionamento e sobre eventos por ele detectados. Na maioria das vezes, os *logs* são o único recurso que um administrador possui para descobrir as causas de um problema ou um comportamento anormal (BRASIL, 2003b).

Os arquivos de *logs* se caracterizam por ser flexíveis, onde mediante configurações é possível registrar desde eventos críticos até praticamente todos os eventos de um sistema, desta forma, são considerados a principal fonte de informação sobre o funcionamento dos

programas servindo como ferramenta para a correção de erros e verificações de rotina (FONSECA, 2005).

Qualquer registro de *log* gerado por um sistema pode ajudar a descobrir problemas na execução de *softwares*, problemas de *hardwares*, tentativas de invasão, acessos indevidos, entre outras coisas. Neste sentido, é importante manter estes registros seguros e intactos, pois em uma eventual auditoria de sistemas estes *logs* serão importantes. Portanto, a integridade e disponibilidade dos *logs* são fatores vitais para garantir a continuidade dos negócios. Neste cenário destacam-se as ferramentas que unificam a administração destes registros, pois possibilitam gerenciar redes e sistemas informatizados mais complexos (CABRAL, 2012).

Brasil (2008) explica como os arquivos de *logs* são utilizados:

> "Os arquivos de log são usados para registrar ações dos usuários, constituindo-se em ótimas fontes de informação para auditorias futuras. Os logs registram quem acessou os recursos computacionais, aplicativos, arquivos de dados e utilitários, quando foi feito o acesso e que tipo de operações foram efetuadas".

Desta forma, é possível identificar um invasor ou usuário não autorizado que realizou acesso a um determinado sistema, apagou ou alterou dados, acessou aplicativos, mudou configurações do sistema operacional com o intuito de prejudicar a organização e facilitar futuras invasões. Sem os registros de *logs*, estas ações não seriam identificadas fazendo com que o administrador sequer ficasse sabendo que houve uma invasão (BRASIL, 2008).

2.10.2 A Importância dos *Logs*

O tema *logs* para usuários, gerentes e analistas de segurança, técnicos de redes e especialistas voltados para a área de segurança, é estratégico para a segurança de suas arquiteturas e sistemas. Quando ocorrem incidentes de segurança como problema com *desktop*, parada de um servidor, ataque a uma rede, alterações em banco de dados entre outras situações de um ambiente tecnológico, os *logs* se tornam um ponto comum de informação. Desta forma, o registro, a coleta e análise de *logs* são procedimentos vitais em auditorias de segurança dentro de uma organização (CADERMAN, 2013).

Uma auditoria de segurança bem sucedida depende da existência de registros de *logs* íntegros e confiáveis. Independente do quão seguro é um computador, uma rede ou um sistema, nunca será possível confiar totalmente nos registros de um sistema que foi

comprometido, pois isso dificulta ou até mesmo impossibilita uma auditoria de sucesso. Quando os registros de auditoria estão seguros é possível aumentar as chances de sucesso ao se correlacionar e identificar padrões ou rever os incidentes de segurança ocorridos em um sistema. Para alcançar estes objetivos é recomendável estabelecer um sistema de *logs* centralizado e dedicado, ou seja, que tenha como função exclusiva a coleta, registro e análise de eventos de *logs* (CANSIAN, 2001).

Devido ao fato de armazenar dados gerados pelo sistema, aplicações, rede, atividades dos usuários, entre outros, os *logs* fornecem inúmeras informações que se transformam em indicadores capazes de medir os níveis de segurança e de avaliar se medidas de segurança estão surtindo o efeito esperado e planejado. A melhor forma de verificar a extensão de um incidente de segurança, identificando que ativo foi violado e que a informação foi exposta é por meio dos registros de *logs*, por isso eles são vitais para a continuidade dos negócios de uma organização (CADERMAN, 2013). Neste ponto, é importante ressaltar quanto ao uso correto dos *softwares* de analise de *logs*, pois estes registros são gerados de diversas formas e devem ser interpretados corretamente, possibilitando uma compactação e consolidação das informações no sentido de melhorar a extração do resultado final.

2.10.3 Caracterização e Padrões de *Logs*

Devido ao fato de existir uma grande quantidade de aplicações distintas, torna-se inviável haver um padrão único para todas as mensagens de *logs*. Cada aplicação específica possui um formato diferente para estes registros (CLEMENTE, 2008). Clemente (2008) argumenta que mesmo havendo estas particularidades, existem informações básicas que estão presente na maioria dos *logs*:

- **Marcação temporal (*timestamp*):** contém a informação de data e hora que o registro de *log* foi gerado. É muito importante para que a pessoa possa comparar a ocorrência de um evento com outros registros de *log* no tempo e, assim, correlaciona-los;
- **Severidade:** caracteriza a importância daquele registro de *log* tendo em vista a integridade do sistema. É determinada em escala gradativa contendo, no mínimo, cinco níveis:

- ***Debug* ou *trace*:** informações sobre o fluxo de execução do sistema, muito utilizada na fase de desenvolvimento de um *software*;
- ***Info* ou *notice*:** caracteriza um evento meramente informacional;
- ***Warn:*** caracteriza um evento sobre o qual se deve ter uma maior atenção e, possivelmente, executar uma ação para prevenção de um erro;
- **Erro:** caracteriza um evento de erro no sistema;
- **Fatal:** caracteriza um erro grave, que pode causar danos ao sistema.

Quando existir um gerenciamento centralizado de *logs*, outra informação importante que deve estar presente nestes registros é a fonte que o originou. Esta informação vai identificar o servidor, ou aplicação, que gerou aquele evento de *log* (CLEMENTE, 2008).

2.10.4 Armazenamento de *Logs*

Os eventos de *logs* podem ser armazenados de duas formas: *on-line* e *off-line*. A forma *on-line* se refere à implantação de um *loghost* centralizado e dedicado à coleta e ao armazenamento de *logs* de outros sistemas em uma rede, este serve como um repositório redundante de *logs*. A grande vantagem desta forma de armazenamento é que um *loghost* centralizado facilita a análise dos *logs* e correlação de eventos ocorridos em sistemas distintos. Sempre que possível, o uso de *loghosts* centralizados é fortemente recomendado. A forma *off-line* se refere ao armazenamento de *logs* em dispositivos, tais como fita, HD externo, CD-R ou DVD-R. Na forma *off-line* é recomendável gerar um *checksum* criptográfico (tal como MD5 ou SHA-1) dos *logs* que são armazenado. Desta forma é possível verificar a integridade destes registros, no caso de eles serem alterados (BRASIL, 2003b).

2.10.5 Sistema de Gerenciamento de *Logs*

Os *logs* possibilitam o acompanhamento do que acontece em um ambiente informatizado. Desta forma, é importante que eles sejam monitorados com frequência para possibilitar que eventuais problemas sejam identificados e solucionados. Neste sentido, o gerenciamento de *logs* segundo Clemente (2008) envolve um amplo conjunto de atividades:

- **Análise Léxica e Transformação:** processo que analisa as linhas de mensagens de *log* e produz uma saída formatada em um padrão mais adequado para futuro processamento;
- **Transmissão e Recepção:** processo que transmite os eventos de *log* para um servidor central em conjunto com o processo que recebe as mensagens de *log* no servidor;
- **Análise Sobre Eventos de *Log*:** processo, que por meio de algoritmos, regras ou consultas extrai as informações relevantes sobre mensagens de *logs*;
- **Compactação:** processo que reduz o tamanho de uma informação por meio de algoritmos de compactação de texto;
- **Armazenamento:** processo que compreende guardar os registros de *logs* por um tempo determinado;
- **Indexação e Busca:** processo responsável por estruturar os registros de *logs* de forma que sejam posteriormente pesquisados;
- **Visualização:** processo que permite a visualização dos registros de *logs* sejam eles em tempo real ou em registros já armazenados.

Este conjunto de atividades deu origem a uma nova classe de sistemas denominados de Sistemas de Gerenciamento de *Log* (*Log Management Systems* - LMS). Estes sistemas implementam os procedimentos acima listados e normalmente consistem em utilizar a forma de armazenamento *on-line*, ou seja, um *loghost* centralizado para receber, indexar, analisar e visualizar os eventos de *logs*. Um LMS é capaz de suprir a necessidade de uma pequena aplicação até sistemas mais complexos com grande volume de acessos e distribuídos (CLEMENTE, 2008).

2.11 Trabalhos Relacionados

Este trabalho trata do desenvolvimento de uma solução especializada e centralizada para gestão de *logs* de aplicações capaz de garantir a segurança da informação, detectando atividades não autorizadas ou em não conformidade com a política organizacional, registrando e monitorando ocorrências de um estado da aplicação auditada em um único repositório, possibilitando que problemas sejam identificados e solucionados de maneira ágil.

Inseridos nesta mesma linha de pesquisa, atualmente existem algumas ferramentas e trabalhos propostos que servem como modelo e possuem como premissa a incorporação do processo de *logging* em sistemas informatizados, bem como o gerenciamento e armazenamento centralizado de registros de *logs*.

Por exemplo, para a infraestrutura de desenvolvimento de *log* existem projetos (LogKit (2013), Apache Log4j (2013), JLog (2013)) que oferecem um conjunto de bibliotecas, as quais possuem operações para instanciar, configurar, formatar as saídas e envio das mensagens para o mecanismo de *log*. Estes projetos apenas geram as informações de *logs* e não possuem extensões que possibilitem a captura e armazenamento centralizado destas informações. São bibliotecas utilizadas no estágio de desenvolvimento de um projeto de *software*, ou seja, muitas vezes fica a critério do programador utilizar estes projetos para gerar *logs* de dados na sua aplicação. Normalmente são utilizados com ferramenta de *debug*, gerando *logs* de exceções ou erros da aplicação. Servem apenas como ferramenta auxiliar no ciclo de desenvolvimento de uma aplicação.

Relacionado à área de redes de computadores, existem diversos projetos que auxiliam na difícil tarefa de coletar registros de *logs*. De acordo com Solha (1999) neste contexto está inserido o TCP *Wrappers* - ferramenta capaz de interceptar e filtrar pedidos de conexão aos serviços de rede inicializados pelo *inetd* (*FINGER*, *FTP*, *TELNET*, *RLOGIN*, *RSH*, *EXEC*, *TALK* e outros), o *PingLogger* - ferramenta que registra pacotes ICMP (*Internet Control Message Protocol*) e o *Secure Syslog* (*ssyslogd*) - ferramenta de auditoria de *logs* gerados em uma rede de computadores.

Na área relacionada ao armazenamento e análise de *logs*, existem projetos (Facebook Scribe (2013), Splunk (2013), LogZilla (2013)), capazes de oferecer um mecanismo para a centralização de *logs*. Estas ferramentas possuem como foco principal realizar *logging* de redes de computadores e sistemas operacionais, deixando um pouco a desejar quanto à necessidade de *logging* sobre aplicações empresariais identificando principalmente alterações em banco de dados.

O *Scribe* foi desenvolvido pelo *Facebook* usando o *Apache Thrift* (linguagem de definições de interfaces) e lançado em 2008 como código aberto. Este projeto trata-se de um servidor para armazenar dados de *logs* transmitidos em tempo real a partir de um grande número de servidores. É uma aplicação distribuída, escalável e extensível onde os servidores se tornam um conjunto de nós, enviando informações uns para os outros garantindo uma segurança maior quando ocorrer falhas na rede. Ele foi projetado para utilizar o mínimo possível de recursos de rede e de disco, ou seja, muita informação fica em memória e caso o

cliente *Scribe* não consiga se conectar a um servidor esta informação pode ser perdida. Além disso, se um servidor falhar a informação que não está em disco é perdida. O *Scribe* foca seu armazenamento em *logs* de acesso, estatísticas de desempenho e ações de usuários para alimentação de informações, sendo indicado para organizações que possuem uma boa estrutura física de rede. O seu uso requer um alto grau de conhecimento, pois a aplicação não é oferecida como serviço em nuvem e caso deseje utiliza-lo será necessário realizar manualmente a sua complexa instalação. A integração com outros sistema é feito por meio do *Thrift* e requer programadores com conhecimento desta biblioteca, além disso, não existe uma interface específica para analisar os registros *logs* tendo que fazer uso de comandos específicos ou utilizar bibliotecas adicionais.

O *Splunk* é uma ferramenta de armazenamento, pesquisa e análise de *logs* de TI. Por meio dele é possível investigar incidentes de segurança em questão de minutos e também monitorar a infraestrutura de TI de ponta a ponta para evitar a degradação ou a interrupção de serviços. Ele não utiliza um banco de dados e faz uso do seu próprio *engine* para organizar os dados em disco, ou seja, requer discos rápidos de no mínimo 800 I/O por segundo. O *Splunk* é instalado como repositório central sendo possível apontar servidores de *logs* ou equipamento de redes para direcionar seus *logs* para ele. Além disso, é possível instalar um aplicativo adicional em qualquer máquina da rede para varrer os *logs* do sistema operacional e enviar ao *Splunk*. O seu foco são *logs* de equipamentos de redes e sistemas operacionais e possui uma interface rica de fácil utilização e instalação. Por se tratar de uma ferramenta proprietária, a sua desvantagem está no seu alto custo, sendo inviável sua aquisição em empresas de pequeno e médio porte. Além disso, este *software* não é vendido como serviço em nuvem, caso deseje utiliza-lo terá que comprar sua licença e realizar sua instalação em uma rede corporativa.

O *LogZilla* é uma ferramenta para armazenamento e análise de *logs* de redes de computadores. É possível direcionar os *logs* dos equipamentos de redes para enviar suas informações ao *LogZilla*, bem como é possível instalar bibliotecas e aplicativos nas mais diversas máquinas da rede para enviar seus *logs* ao repositório central. O *LogZilla* é capaz de coletar dados e índices de máquinas em grande escala permitindo aos usuários pesquisar, analisar, monitorar e reportar esses dados em tempo real. Esta ferramenta foi projetada para *logs* de redes sendo inviável seu uso para monitorar *logs* de aplicações comerciais, além disso, trata-se de ferramenta proprietária que não oferece serviço em nuvem e mesmo possuindo uma licença *free*, para pequenas quantidades de dados e até dez *hosts*, quando houver uma demando maior a sua aquisição pode se tornar inviável.

Na área acadêmica também existem projetos que estudam a captura e centralização de arquivos de *logs*. Neste cenário é possível destacar o projeto de Alves *et al*. (2007) denominado *Middlog* - infraestrutura de serviços de *log* de aplicações baseada em tecnologias de *Middleware*[3] e o projeto de Clemente (2008) que trata de uma arquitetura para processamento de eventos de *log* em tempo real.

O projeto *MiddLog* estende a atual tecnologia de infraestrutura de *log*, agregando novas funcionalidades que oferecem ao desenvolvedor um conjunto completo e flexível de serviços de *logging*. As atuais infraestruturas de *log* permitem que programadores possam incluir *logs* em suas aplicações, porém cabe ainda ao desenvolvedor escolher os pontos de monitoramento de sua aplicação, incluindo em seu código fonte chamadas para a geração de *logs*. Desta forma, o *MiddLog* eleva as funcionalidades de gerenciamento de *log* ao nível de um serviço de *middleware*, permitindo que desenvolvedores de aplicações não tenham que se preocupar com os problemas associados a geração de *logs* (ALVES et al., 2007). Ou seja, basta configurar o *Middlog* na aplicação que ele se responsabiliza por capturar e armazenar os *logs*, seja em ambiente de desenvolvimento ou ambiente de produção. Este projeto é útil como ferramenta de depuração, pois é possível capturar erros ou exceções que uma aplicação está gerando. Seu uso não é apropriado para *logs* mais específicos (*logs* de acesso, alterações em banco de dados etc.), além de não possuir interface de consulta, dificultando assim sua utilização.

O projeto de Clemente (2008) especifica os requisitos de um sistema de gerenciamento de *logs*, apresentando conceitos relacionados ao padrão de arquitetura orientada a eventos e propõe esta arquitetura para o gerenciamento de *logs*. Clemente (2008) desenvolveu um protótipo capaz de receber, analisar e correlacionar diversos tipos de *logs*. Este projeto possui uma estrutura mais genérica (pode ser usada por diferentes tipos de aplicativos), mas seu foco são arquivos de *logs* de servidores *web*. É possível fazer consultas e correlacionar *logs*, mas isto só é possível através de comandos SQL (*Structured Query Language*) para um banco de dados específico o que dificulta bastante seu uso, já que é necessário possuir conhecimentos específicos.

[3] Programa de computador que faz uma intermediação em um *software* e demais aplicações.

	Scribe	Splunk	LogZilla	Middlog	Prj. Clemente	Argos
SaaS (Software as a service)	Não	Não	Não	Não	Não	Sim
Software Livre	Sim	Não	Não	-	-	Sim
Investimento	Médio	Alto	Médio	Baixo	Baixo	Baixo
Grau de conhecimento necessário para utilização	Alto	Médio	Médio	Alto	Alto	Baixo
Foco	Logs de Redes	Logs de SO e Redes	Logs de Redes	Logs de Debug	Logs de Servidores Web	Logs de Aplicações Empresariais
Interface de Pesquisa	Complexa via comandos	Fácil e Rica	Fácil	-	-	Fácil e Rica

Fonte: O Autor (2013).

Quadro 1 - Comparação de alguns trabalhos relacionados ao Argos

Conforme pode ser observado no Quadro 1, o Argos se diferencia dos demais trabalhos relacionados pelo fato de poder ser utilizado como uma aplicação SaaS[4], ou seja, é responsável por toda a estrutura necessária para a disponibilização do sistema (servidores, conectividade, cuidados com segurança), possibilitando ao cliente (empresa, organização que vai integrar ao *software*) a utilização da aplicação via *internet*. Além disso, o Argos trata-se de um *software* livre, necessita de pouco investimento e possibilita a sua implantação e utilização de forma fácil e descomplicada, sem requerer um alto grau de conhecimento.

[4] *Software as a service* ou *Software* como serviço onde quem disponibiliza o *software* se responsabiliza por toda a estrutura necessária para a disponibilização do sistema.

3 DESENVOLVIMENTO

Este trabalho trata de uma solução para a gestão centralizada de *logs* que engloba uma série de métodos e tecnologias especialistas. Desta forma, para evidenciar o cenário de atuação, processos, testes e tecnologias utilizadas, este capítulo aborda todo o ciclo de construção da aplicação e os resultados obtidos, fundamentando a dimensão e reconhecendo a aplicabilidade da tarefa de *logging* em aplicações.

3.1 Cenário Atual do *Logging* em Aplicações Empresariais

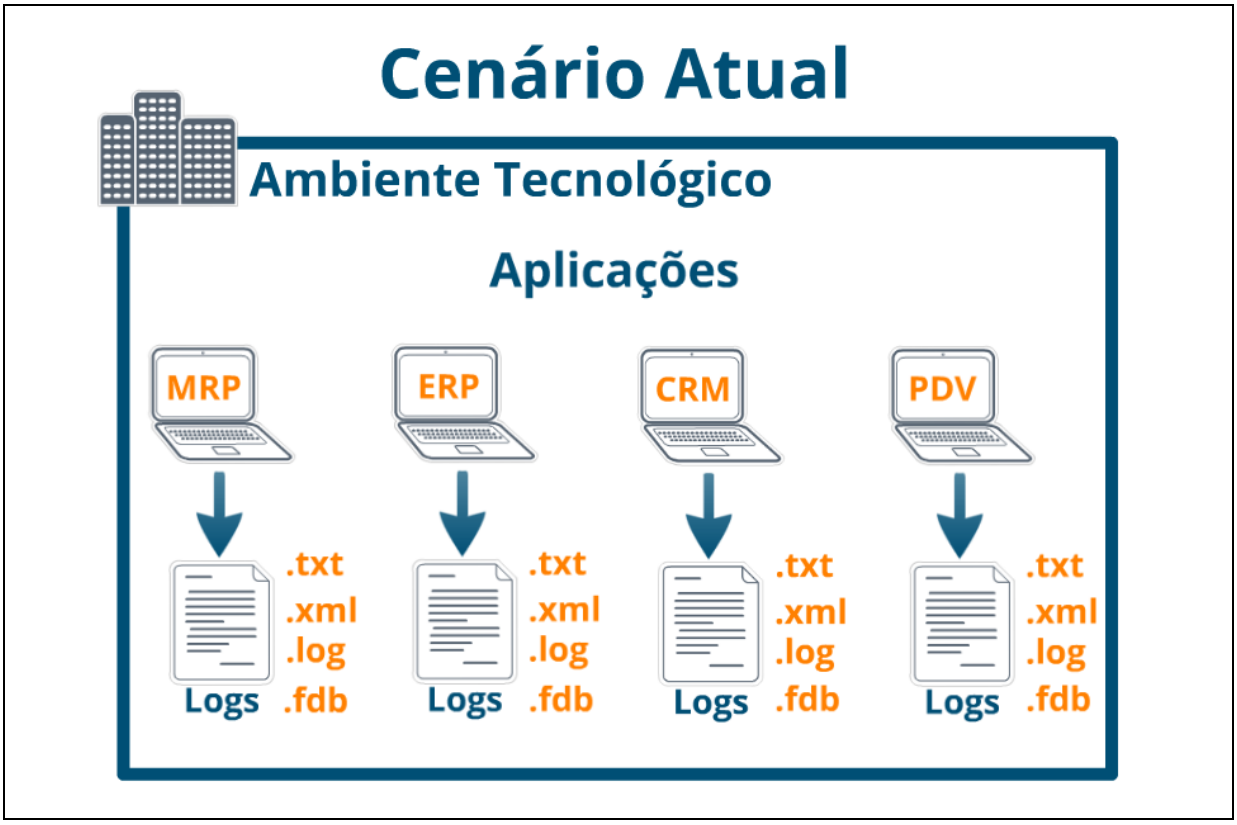
Armazenar e gerenciar *logs* em aplicações empresariais é importante ao ponto que cada vez mais a informação é considerada um ativo altamente significativo e essencial para os negócios de qualquer organização. Desta forma, ela deve ser gerenciada e protegida com a adoção de medidas adequadas, principalmente em um ambiente de negócios cada vez mais interconectado. Ao garantir a segurança desta informação é possível dar continuidade aos negócios, minimizar riscos e ameaças e maximizar investimentos e oportunidades.

Empresas que desenvolvem aplicações empresariais estão percebendo que a segurança da informação vem ganhando mais notoriedade e precisa ser estudada, principalmente com o desenvolvimento de funcionalidades que atendam seus requisitos. Muitos controles e métricas são criados para monitorar estas aplicações consideradas elementos críticos que compõe o ambiente de TI de pequenas, médias ou grandes empresas. Neste cenário, o grande problema é que cada elemento possui seus controles de segurança e, na maioria das vezes, as práticas existentes não permitem uma análise real do que está acontecendo, devido a grande fragmentação da informação (BIANCHI, 2012).

As *softwares houses*[5] que não possuem um repositório central para *logs* de suas aplicações encontram dificuldades para solucionar problemas, pois segundo Bianchi (2012), "quando um problema acontece, não existe uma ferramenta gráfica que colete a informação correta para que se possa fazer uma análise mais detalhada sem passar antes por todos os profissionais do departamento de TI e acabar naquele jogo de empurra entre as áreas".

[5] Empresas ou organizações que se dedicam a construir programas para ambientes computacionais, normalmente para fins comerciais.

A Figura 2 ilustra o cenário onde se encontram a maioria das *softwares houses*. Nestas empresas a tarefa de *logging* é realizada armazenando os *logs* em diversos locais, em vários tipos de arquivos, inclusive na própria base de dados da aplicação. Isto ocasiona perca de desempenho, possibilita que as informações de *logs* sejam negligenciadas, além aumentar a dificuldade em utilizar estes registros para resolver problemas. Em muitos casos os registros de *logs* acabam se tornando mais um problema para ser administrado.



Fonte: O Autor (2013).

Figura 2 - Cenário atual do *logging* em aplicações empresariais

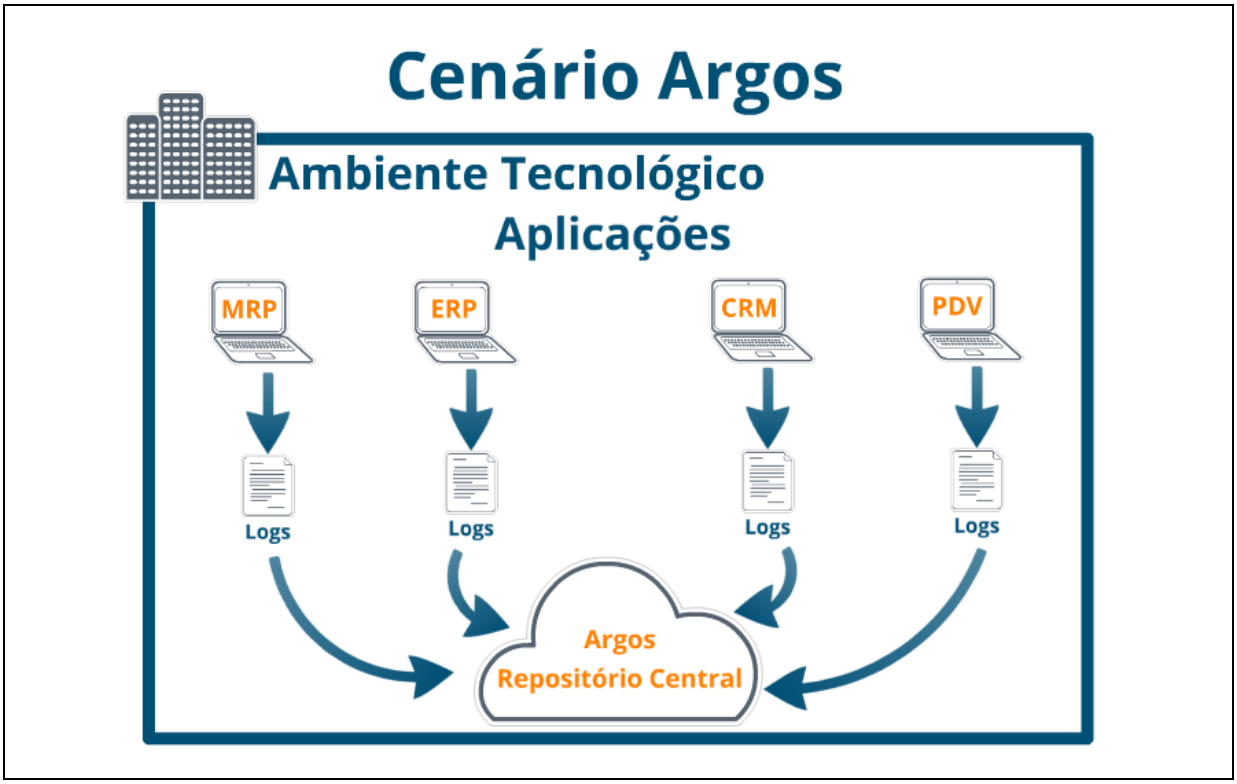
O objetivo deste trabalho foi desenvolver uma solução que resolvesse o problema de *logging* em aplicações empresariais, auxiliando principalmente as *softwares houses* na gestão centralizada de *logs* de seus sistemas. A solução desenvolvida é capaz de garantir maior monitoramento e análise dos eventos de *logs*, o que possibilita uma tomada de decisão mais eficaz e proporciona uma garantia de que problemas ou comportamentos anormais destes sistemas sejam identificados e atendidos com uma maior brevidade.

3.2 Argos: Solução Centralizada para *Logging* de Aplicações

O presente trabalho de conclusão trata do desenvolvimento de um *software* sob a modalidade *Open Source*[6], que possui a *web* como plataforma de execução e capaz de realizar de forma centralizada o gerenciamento e análise de eventos de *logs* gerados por diferentes aplicações. Este *software* é dotado de funcionalidades capazes de auxiliar na detecção de atividades não autorizadas ou em não conformidade com as políticas de segurança da informação adotadas por estas aplicações.

O *software*, estudo deste trabalho, foi denominado Argos e possui como premissa ser utilizado por *softwares houses* na gestão de *logs* de aplicações empresariais, sendo que neste quadro se encaixam os sistemas ERP (*Enterprise Resource Planning*) – sistemas integrados de gestão empresarial; CRM (*Customer Relationship Management*) – gestão de relacionamento com clientes; SCM (*Supply Chain Management*) – gestão de cadeia de logística; MRP (*Manufacturing Resource Planning*) – planejamento dos recursos de manufatura. Conforme pode ser visto na Figura 3, o Argos oferece um novo cenário para a gestão de *logs*. Neste novo cenário a sua principal função é receber, armazenar e indexar eventos de *logs* relacionados a alterações em banco de dados, acesso ao sistema, módulos e configurações, erros ou exceções que possam ocasionar perigo a segurança da informação.

[6] *Software* livre de código aberto.



Fonte: O Autor (2013).

Figura 3 - Cenário do *logging* de aplicações proposto pelo Argos

As aplicações de terceiros que desejam gerenciar seus *logs* devem realizar uma integração com o sistema Argos, ou utilizar um sistema de *Middleware*[7] que faça este serviço. Desta forma, o Argos vai disponibilizar um serviço que pode estar disponível na nuvem (SaaS - *Software as a servisse* ou *Software* como serviço) ou até mesmo em uma rede local, que vai receber os registros de eventos de *logs*, armazená-los, indexá-los, possibilitando futuras consultas ou relatórios, e no caso de *logs* críticos são enviados alertas ao administrador da aplicação auditada.

3.2.1 Metodologia de Desenvolvimento

A metodologia de desenvolvimento do Argos está divido em seis processos: levantamento de requisitos, modelagem do sistema, definição de tecnologias, programação,

[7] Programa de computador que faz uma intermediação em um *software* e demais aplicações.

testes e implantação. Estes processos serão abordados na sequência e possibilitam compreender de forma detalhada todo o ciclo de desenvolvimento do *software*.

3.2.1.1 Levantamento de Requisitos

O ponto de partida para o desenvolvimento do Argos foi o levantamento de requisitos. Este processo foi importante para compreender a extensão do domínio da aplicação e quais eram as suas principais funcionalidades. A técnica utilizada para definição dos requisitos do sistema foi a realização de entrevistas com profissionais da área de desenvolvimento de *software*, suporte e gestão de TI. Para isto, foi elaborado um plano de entrevista para que não houvesse uma dispersão do assunto principal. O entrevistador deu margem ao entrevistado para expor suas ideias e após as entrevistas houve um processo de contextualização das informações, dando inicio ao processo de modelagem do *software* que será abordado nos próximos tópicos.

3.2.1.2 UML (*Unified Modeling Language*)

A cada dia que passa cresce o volume de informações para auxiliar nas tomadas de decisões de pequenas, médias e grandes empresas. Estas informações devem estar corretas e disponíveis de forma imediata, pois nos dias atuais a maioria dos problemas encontrados nas organizações reside na adoção de métodos inapropriados para análise e levantamento de requisitos, resultando em projetos com objetivos incorretos ou não muito claros (LIMA, 2007).

A modelagem surge, pois não é possível compreender os sistemas completamente, e sobre isso Lima (2007, p.29), argumenta:

> Os melhores modelos expressam a realidade, e quanto maior a complexidade do produto a ser entregue, maior a necessidade de boas técnicas de modelagem, com um conjunto de visões (pontos de vista ou perspectivas), suficientes para garantir a sua compreensão. [...] o exercício de concentrar-se no que é relevante e ignorar os demais detalhes, é essencial para apreender e comunicar o domínio da aplicação e, com bons modelos, podemos assegurar a comunicação entre a equipe de projeto e uma arquitetura sólida.

O *software* se tornou uma ferramenta indispensável para gerenciamento dos processos organizacionais. Neste sentido, as empresas perceberam a quantidade de benefícios oferecidos pela orientação a objetos no desenvolvimento de aplicações corporativas e, a partir daí, começaram a adotar técnicas que possibilitassem implementar o modelo orientado a objeto, definindo padrões claros e objetivos, principalmente com o crescimento da *Web* (LIMA, 2007).

Com o objetivo de especificar, documentar e estruturar os projetos de *softwares*, permitindo padronizar as formas de modelagem, surge a UML e segundo Lima (2007, p.40), "A UML é uma linguagem visual para especificação, construção, visualização e documentação de artefatos de um software intensivo, para representar projetos orientados a objetos utilizando uma notação comum." Sobre a *Unified Modeling Language* (UML), Pender (2004, p.3) argumenta que "[...] é uma destilação de três notações principais e uma série de técnicas de modelagem retiradas de metodologias bastante diversificadas, que já existem na prática desde as duas últimas décadas.".

Suas características permitem criar modelos que admitam escalabilidade e segurança, e sobre isso Pender (2004, p.3) cita:

> Como a modelagem UML eleva o nível de abstração por todo o processo de análise e projeto, é mais fácil identificar padrões de comportamento e, portanto, definir oportunidades para recriação e reuso. Conseqüentemente, a modelagem UML facilita a criação de projetos modulares, resultando em componentes e bibliotecas de componentes que agilizam o desenvolvimento e ajudam a garantir a coerência através de sistemas e implementações.

A UML se destaca por oferecer um conjunto de recursos extremamente valiosos para o mundo da modelagem, existem os mecanismos de extensibilidade, onde em um primeiro momento a UML foca em conceitos básicos de funcionalidades, sendo que por meio desta extensibilidade estes recursos iniciais podem ser ajustados ou aumentados sem alterar a sua integridade inicial. Para solucionar problemas comuns, existem os padrões e colaborações, para modelar processos lógicos existem os diagramas de atividades, para tratar relacionamentos entre níveis de abstração existe o refinamento, para modelar focalizando conectividade e comunicação existe as interfaces e componentes e para garantir a integridade dos modelos existe a linguagem de restrição (PENDER, 2004).

Descrever os elementos que compõe o sistema por meio de diagramas de classes, objetos, estrutura de composição, componentes e implantação é característica dos diagramas estruturais. Já a característica de descrever o comportamento dos elementos e suas interações

por meio de diagramas de caso de uso, atividades, interação e máquina de estados referenciam os diagramas comportamentais (PENDER, 2004).

A UML possui uma grande área de abrangência, oferecendo mecanismos capazes de modelar funcionalidades e processos de forma simples, favorecendo a rápida compreensão e análise de pontos importantes de um *software*. Em virtude disso, foram utilizados alguns de seus conceitos na construção do Argos com o objetivo de modelar este projeto na estrutura orientada a objetos e em uma notação comum e de fácil entendimento. Desta forma, foram estudados e desenvolvidos um diagrama de caso de uso e um diagrama de classes.

3.2.1.2.1 Diagrama de Caso de Uso

Tem como característica identificar os recursos propostos para o sistema de uma forma mais simples, facilitando a visualização dos requisitos por parte do cliente. É por meio dele que o cliente saberá se o sistema está atingindo as suas expectativas, além de servir como base para o acompanhamento do progresso dos trabalhos (PENDER, 2004).

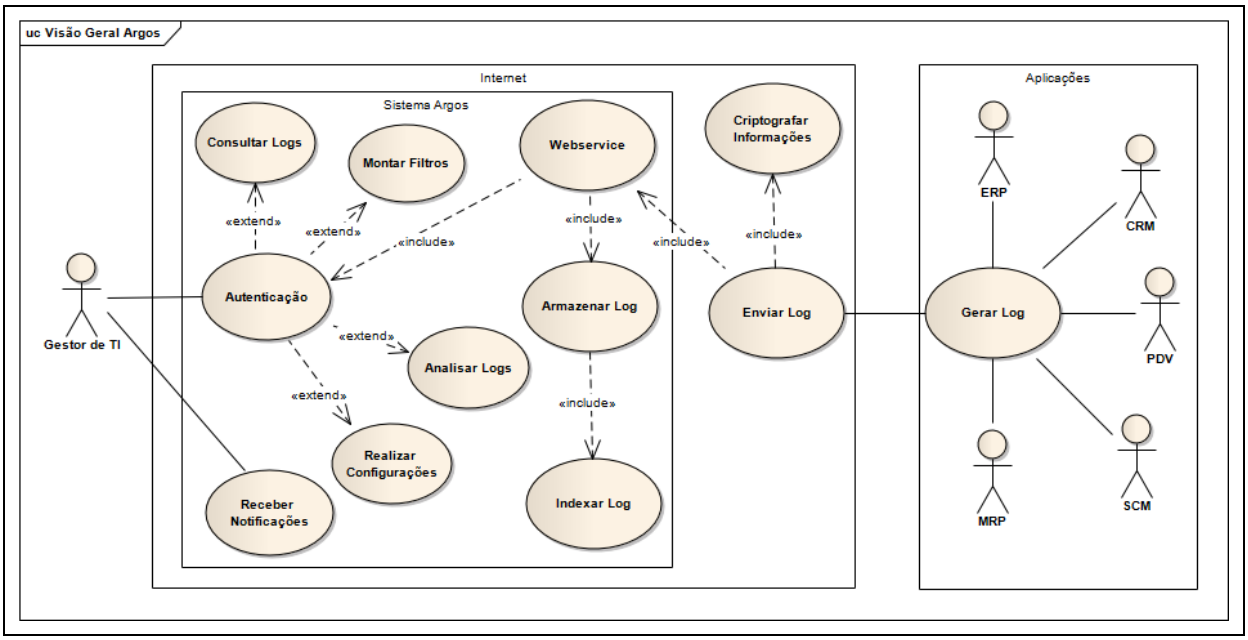
Sobre este diagrama Lima (2007, p.58) argumenta:

> Um caso de uso bem escrito é fácil de ler, pois consiste em sentenças escritas em passos de ação simples, em que um ator alcança um resultado ou transmite informação ao sistema. De modo geral, a leitura de um caso de uso toma poucos minutos e pode ser compreendida por qualquer leitor envolvido com o negócio, não sendo exigido nenhum conhecimento de informática – o foco é o problema e não a solução informatizada.

Desta forma, quanto mais claros e objetivos forem os casos de usos, maiores são as chances de sucesso da aplicação no momento de se obter um *feedback* do usuário. Isto se deve ao fato do diagrama de caso uso oferecer uma visão mais simplificada do funcionamento de um *software*. No Argos este diagrama facilitou a compreensão do sistema, pois ofereceu uma visão mais macro de sua estrutura, ambiente e funcionamento.

O diagrama de caso de uso, apresentado na Figura 4, demonstra que o Argos é um *software* que utiliza a plataforma *web*, ou seja, é executado na nuvem (SaaS - *Software as a servisse* ou *Software* como serviço) oferecendo um mecanismo para envio de registros de *logs* com um padrão pré-definido. A aplicação que integra ao Argos deve gerar os *logs* e enviar os mesmos por meio de requisições HTTP (*Hypertext Transfer Protocol*). Este serviço se dá por

meio de um *WebService*[8] com o padrão de troca de mensagens REST (*Representational State Transfer*). Estes *logs* podem conter atividades de usuários, exceções do sistema ou qualquer outro evento de segurança da informação. Após o armazenamento e indexação, o gestor de TI pode analisar e filtrar os *logs* através de uma interface *web*. Além disso, outros aspectos cruciais foram levados em consideração no desenvolvimento desta solução, como acesso seguro com autenticação para envio e análise de *logs*, transferência de dados de forma criptografada, possibilidade de montagem de filtros, além do envio automático de alertas quando houver *logs* críticos.



Fonte: O Autor (2013).

Figura 4 – Caso de uso representando a visão geral do Argos

O diagrama de caso de uso do Argos representa os requisitos funcionais do sistema e tem como objetivo permitir uma compreensão mais simples do seu ambiente, estrutura e funcionamento.

[8] API (Interface de Programação de Aplicativos) *web* que pode ser acessada por meio de uma rede.

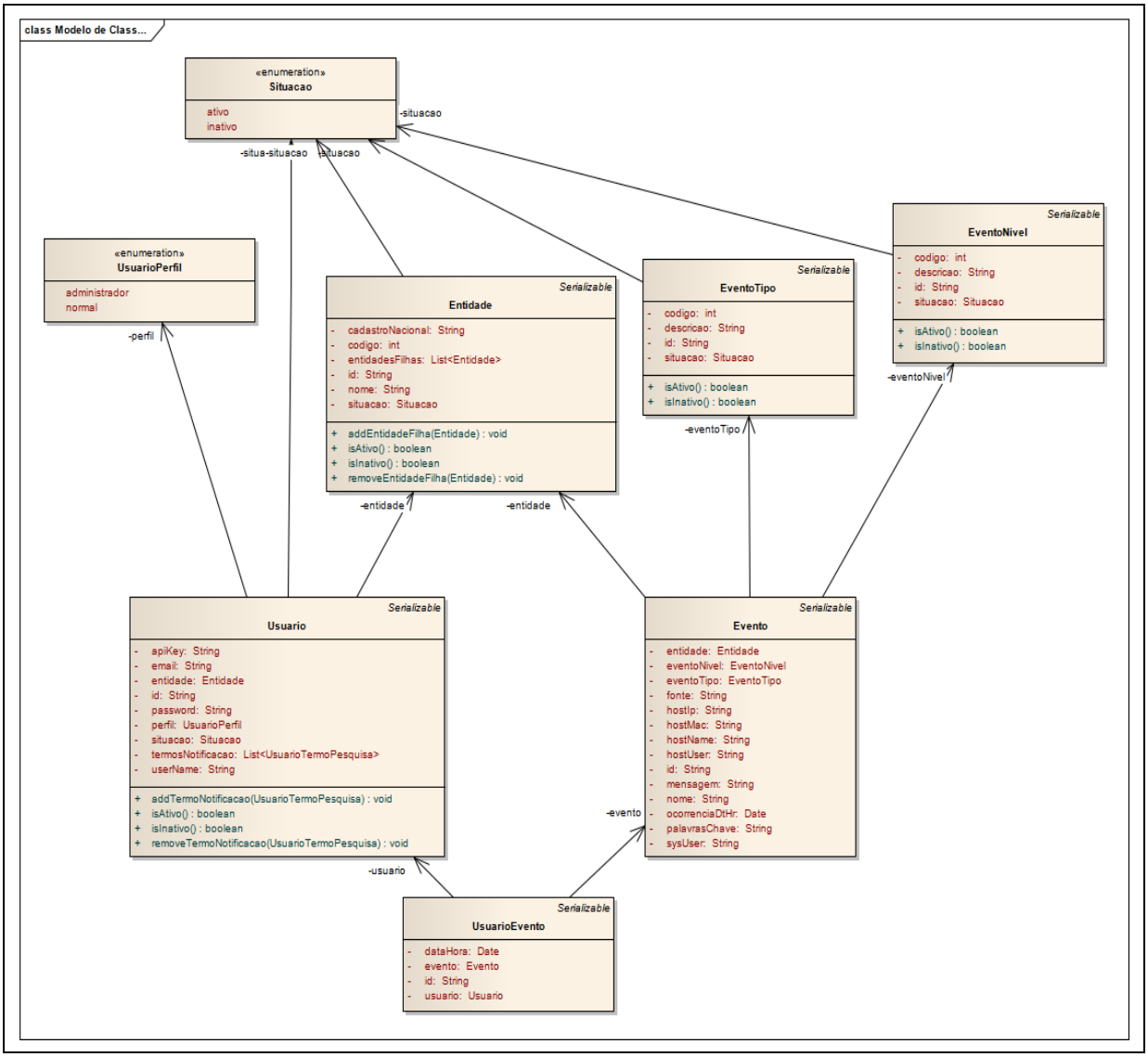
3.2.1.2.2 Diagrama de Classes

Tem como característica definir os recursos essenciais para a correta operação do sistema, modelando cada recurso em termos de sua estrutura, relacionamento e comportamento, definindo a geração de código, oferecendo um foco para os demais diagramas e possibilitando a engenharia reversa (PENDER, 2004).

Sobre este diagrama Lima (2007, p.174) argumenta:

> Um diagrama de classe mostra a estrutura estática do modelo, em que os elementos são representados por classes, com sua estrutura interna e seus relacionamentos. Os diagramas de classes também podem ser organizados em pacotes, mostrando somente o que é relevante em um pacote específico, considerando, por exemplo, o contexto que um grupo de classes têm em comum [...], ou os atores que utilizam seus serviços [...].

O diagrama de classes do Argos foi desenvolvido focando nas classes de domínio, conforme Figura 5, pois estas servem como base para toda a estrutura do *software*. Este diagrama define os nomes, atributos, métodos e relacionamentos das classes implementadas no sistema em uma notação visual de fácil compreensão. Desta forma, é possível identificar como são compostos os objetos, quais são suas ligações e qual a dependência entre eles no momento de salvar e indexar os registros de *logs*.



Fonte: O Autor (2013).

Figura 5 – Diagrama de classes de domínio do Argos

O diagrama das classes de domínio teve por objetivo demonstrar a estrutura e o relacionamento ao nível de programação dos objetos que compõem o sistema Argos, sendo uma ferramenta indispensável na construção do *software*.

3.2.1.3 Padrões de Projetos (*Design Patterns*)

Durante o processo de construção de um *software*, desenvolvedores se deparam com vários tipos de problemas durante a codificação. Quando se fala em desenvolvimento orientado a objetos estes problemas são bastante evidentes e comuns. Na tentativa de evitar estes problemas constantes, foram criados alguns padrões, definidos como padrões de projetos ou *design patterns*. Os padrões de projetos servem para solucionar problemas que são comuns em linguagens orientadas a objetos, agilizando o desenvolvimento e criando um padrão de desenvolvimento (ZEMEL, 2009a). Braude (2005, p.158) define padrões de projetos como: "[...] uma combinação de classes e algoritmos associados que cumprem com propósitos comuns de projeto". Ainda de acordo com Braude (2005), os padrões de projetos são classificados em tipos, onde variam entre padrões criacionais, estruturais e comportamentais.

Os padrões de projetos que são caracterizados como criacionais trabalham com a instanciação dos objetos, tornando a aplicação mais flexível, pois possibilitam que o *software* seja independente da forma como são instanciados os objetos, delegando esta responsabilidade ao padrão empregado (DESTRO, 2004).

Os padrões de projetos estruturais segundo Braude (2005, p. 242), devem "[...] representar objetos complexos (o ponto de vista estático) e obter funcionalidades a partir deles de maneira a utilizar seus objetos agregados (o ponto de vista dinâmico)". Ao que diz respeito aos padrões de projetos comportamentais, estes podem ser caracterizados por serem responsáveis por captar o comportamento dos objetos durante a utilização das funcionalidades estabelecidas (BRAUDE, 2005).

No desenvolvimento do Argos a utilização de padrões de projetos trouxe vários benefícios. O sistema pode ser desenvolvido de forma mais clara, melhor documentado e com maior possibilidade de exploração de bibliotecas alternativas. Houve uma melhora geral no *software*, pois a complexidade do código foi reduzida, oferecendo abstrações de classes e instâncias, além da redução do tempo de aprendizado com novos componentes ou métodos. Com a utilização dos padrões de projetos foi possível agilizar trabalhos repetitivos e custosos.

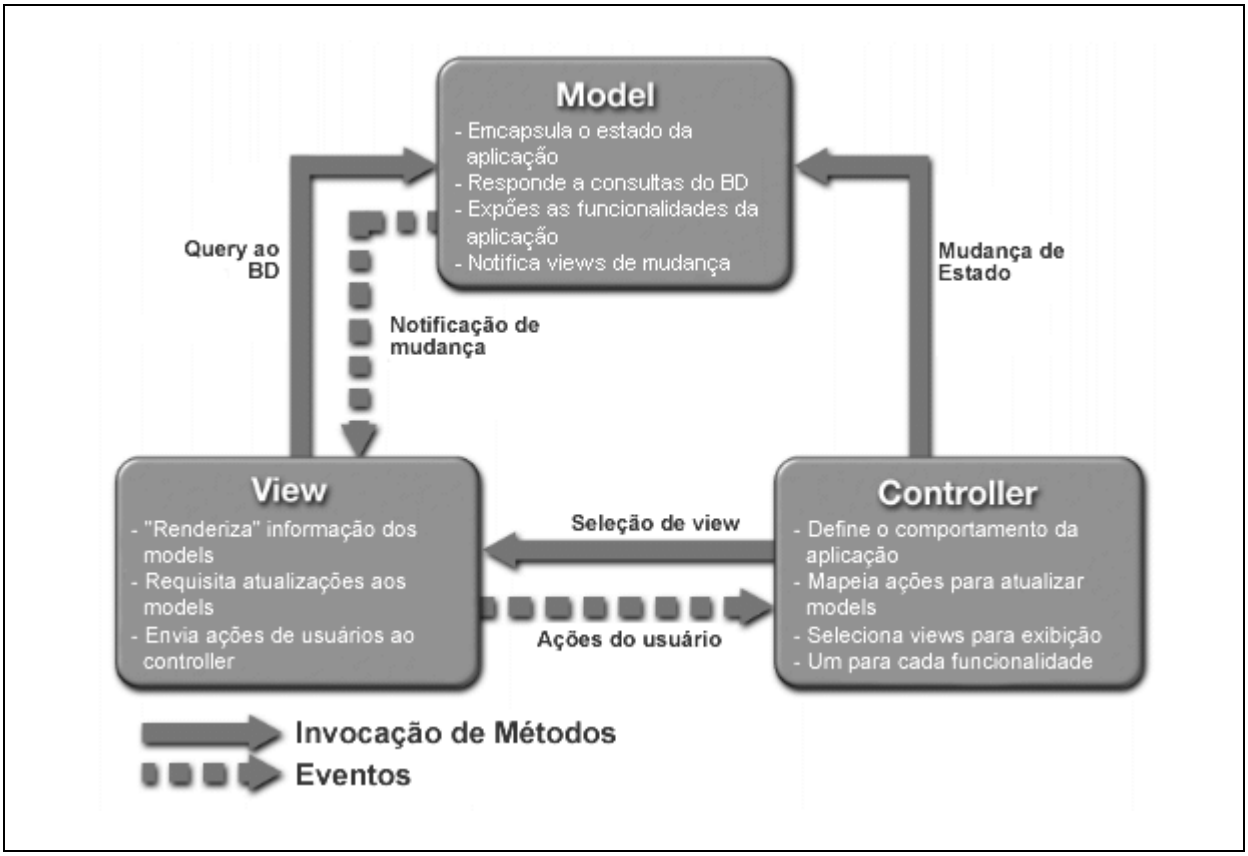
Dentre os padrões de projetos utilizados durante o desenvolvimento do *software* o de maior destaque foi o MVC (*Model-View-Controller*), pois as elaborações de tarefas foram divididas entre *models*, *views* e *controllers* fazendo a aplicação ficar leve e independente. Desta forma, ficou fácil desenvolver novos módulos ou alterar funcionalidades existentes, pois a

separação em camadas que o MVC oferece permite que a alteração de uma parte da aplicação não afete outras partes.

3.2.1.3.1 MVC (*Model-View-Controller*)

MVC é um padrão arquitetural que separa uma aplicação em várias camadas. Devido ao fato das aplicações terem aumentado a sua complexidade no desenvolvimento, é indispensável que ocorra esta divisão. Desta forma, quando é feita uma alteração em uma camada esta não afeta o funcionamento da outra. O MVC define que cada camada realiza um tipo de tarefa na aplicação, sendo dividas em *Model* (Modelo), *View* (Visão) e *Controller* (Controle) (ZEMEL, 2009b).

Como pode ser observado na Figura 6, a camada denominada modelo tem a responsabilidade de manipular os dados e fazer a aplicação das modificações que são solicitadas pela camada de controle. A camada de controle gerencia as operações que são realizadas no sistema tendo um relacionamento com a camada de visão e modelo. Neste sentido, a camada de controle tem como responsabilidade receber as solicitações que são enviadas pela camada de visão e fazer um gerenciamento das operações que provêm da camada de modelo, o contrário também acontece depois de realizado as operações na camada de modelo, o controle retorna as alterações ou as mensagens para a camada de visão. E finalmente a camada de visão que faz uma representação visual dos dados que provêm da camada de modelo e são administrados pelo controle (GONÇALVES, 2007).



Fonte: Zemel (2009b).

Figura 6 – As camadas do MVC (*Model-View-Controller*)

Segundo Rinaldi (2009), sistemas baseados em camadas, possuem como características "[...] a facilidade de manutenção, redução de código e maior acessibilidade". Este padrão tornou-se muito importante nos últimos anos no desenvolvimento de aplicações, em especial as aplicações *Web*. É fortemente empregado em linguagens como *Java* e C#, sendo a linguagem *Java* a principal difusora com os seus diversos *frameworks* (RINALDI, 2009).

Na construção do Argos ao fazer uso do padrão de arquitetura MVC (*Model-View-Controller*) foi possível perceber uma maior facilidade quanto à manutenção da aplicação, possibilitando adicionar, alterar ou retirar funcionalidades do sistema de forma mais simples e eficaz, sem afetar o sistema como um todo.

3.2.1.4 Definição de Tecnologias

Por se tratar de um *software* ambientado na *web* e capaz de armazenar, indexar e processar grandes quantidades de dados fez-se necessário a utilização de tecnologias capazes de suprir toda esta demanda.

Em um primeiro momento foi necessário definir uma ferramenta de engenharia de *software* para contextualização dos requisitos e modelagem do sistema, além de uma IDE (*Integrated Development Environment*) de desenvolvimento e programação. Desta forma, as ferramentas escolhidas foram: *Enterprise Architect* [9] (ambiente de modelagem que contempla todo o ciclo de desenvolvimento do sistema como modelagem de negócios, engenharia, arquitetura corporativa, gerenciamento de requisitos, projeto de *software*, geração de código etc.) e o *NetBeans*[10] (ambiente de desenvolvimento gratuito e de código aberto para desenvolvedores de *software* nas linguagens *Java*, C, C++, PHP, *Groovy*, *Ruby* etc.).

Após a definição da IDE foram levantadas as tecnologias envolvidas na construção do *software* e as de maior destaque são abordadas nos próximos tópicos.

3.2.1.4.1 *Java*

É considerada uma linguagem computacional completa, largamente utilizada no desenvolvimento de aplicações baseadas na *Internet*, redes fechadas ou ainda programas *stand-alone*. Surgiu em meados dos anos 90 nos laboratórios da *Sun Microsystems* com o objetivo de ser mais simples e eficiente do que suas predecessoras. De início seu foco era a produção de *software* para produtos eletrônicos de consumo, mas em virtude de seu grande sucesso, ocasionado principalmente pelo código compacto, arquitetura neutra e o cumprimento dos requisitos de sua especificação, passou a ser utilizada no desenvolvimento de aplicações comerciais (INDRUSIAK, 1996).

Atualmente, o *Java* está presente em vários ramos da tecnologia, sendo uma das plataformas mais importantes do mundo corporativo (*Java* EE) com soluções robustas para *Web* e aplicações distribuídas. Além disso, é também uma das principais plataformas para o

[9] http://www.sparxsystems.com/products/ea/
[10] https://netbeans.org/

desenvolvimento em dispositivos móveis (*Java* ME), com milhões de celulares, *palms*, *set-top boxes* e até aparelhos *blueray* (SILVEIRA et al., 2012).

Apesar de sua disseminação ocorrer principalmente pela utilização em dispositivos móveis, o *Java* ganhou força através dos *Applets*[11] e das aplicações corporativas no *server-side*[12]. Dada sua portabilidade e segurança, a plataforma foi a escolha feita por grandes bancos e empresas que necessitavam de maior flexibilidade. Esta flexibilidade evita o *vendor lock-in*[13] permitindo que não se fique dependente de um único fabricante, um único sistema operacional ou um único banco de dados (SILVEIRA et al., 2012).

O *Java* oferece uma maior liberdade para as empresas, e muito mais que uma linguagem de programação, *Java* é uma completa plataforma de desenvolvimento e execução. Esta plataforma é caracterizada por três pilares: a máquina virtual *Java* (JVM), um grande conjunto de APIs[14] e a linguagem *Java*. A JVM é uma *Application Virtual Machine* que abstrai não só a camada de *hardware* como a comunicação com o sistema operacional. Ao usar o conceito de máquina virtual, o *Java* elimina o problema de portabilidade executando o mesmo código em diversos sistemas operacionais (SILVEIRA et al., 2012).

No desenvolvimento do Argos a utilização da plataforma *Java* trouxe inúmeros benefícios para a aplicação. A portabilidade possibilita que o *software* seja executado em qualquer sistema operacional ou equipamento sem a necessidade de alteração de código. A robustez garante que os recursos da aplicação permanecem estáveis, mesmo tendo grande quantidade de requisições. A segurança protege o sistema contra ataques mal-intencionados. A orientação a objetos dinamiza a aplicação facilitando sua manutenibilidade. Os recursos da linguagem como *multi-threading* e compilação *just-in-time* garantem um alto desempenho da aplicação aproveitando o tempo ocioso do processador para antecipar a compilação de *bytecode* para código nativo.

3.2.1.4.2 *Frameworks*

Quando a reusabilidade se torna eminente no desenvolvimento de aplicações é preciso se habituar e trabalhar com a utilização de *frameworks*, segundo Braude (2005, p. 567)

[11] Pequenas aplicações embutidas em páginas *Web* executadas no navegador.
[12] Operações realizadas no lado servidor das aplicações.
[13] Aprisionamento tecnológico.
[14] API's - Interfaces de Programação de Aplicativos.

“Criamos *frameworks* porque queremos reutilizar grupos de classes e algoritmos entre eles”. Quando se fala em *frameworks* surge a ideia de um conjunto de classes. Macias (2008) conceitua *framework* “como sendo um projeto formado por um conjunto de classes que cooperam entre si e é reutilizável por um domínio de software determinado.” Macias (2008) argumenta que *frameworks* ou “arcabouços” são “uma estrutura muito importante na criação de arquiteturas de software de aplicações corporativas.”.

É importante diferenciar *frameworks* de *design patterns* (padrões de projetos), os *frameworks* possuem como característica serem mais concretos e de mais alto nível, geralmente são utilizados para domínios ou tecnologias específicas, um *framework* pode englobar vários *design patterns*. Já *design patterns* são dispostos em uma forma mais conceitual e podem ser aplicados em uma gama maior de problemas podendo ser usado em praticamente qualquer aplicação (JOHNSON, 2003).

Como os *frameworks* possuem características de serem reutilizáveis extensíveis e com funcionalidades abstratas que podem ser completadas, o tempo de desenvolvimento é reduzido de forma considerável e permite que problemas mais difíceis se resolvam com base nas melhores práticas já empregadas por eles (JOHNSON, 2003).

Quando se utiliza *frameworks* tem se o objetivo de padronizar o projeto e o desenvolvimento. Esses parâmetros definem como é realizada a divisão entre classes e objetos, suas responsabilidades, como cooperam entre si e seu fluxo de informação ditando assim a arquitetura da aplicação que os utilizam (JOHNSON, 2003).

A possibilidade de se utilizar funcionalidades comuns entre diferentes aplicativos caracterizam a criação ou a utilização de *frameworks*. Com a sua utilização é possível se beneficiar do conhecimento aplicado no desenvolvimento de *softwares* que compartilham uma família de problemas comuns entre eles (BRAUDE, 2005).

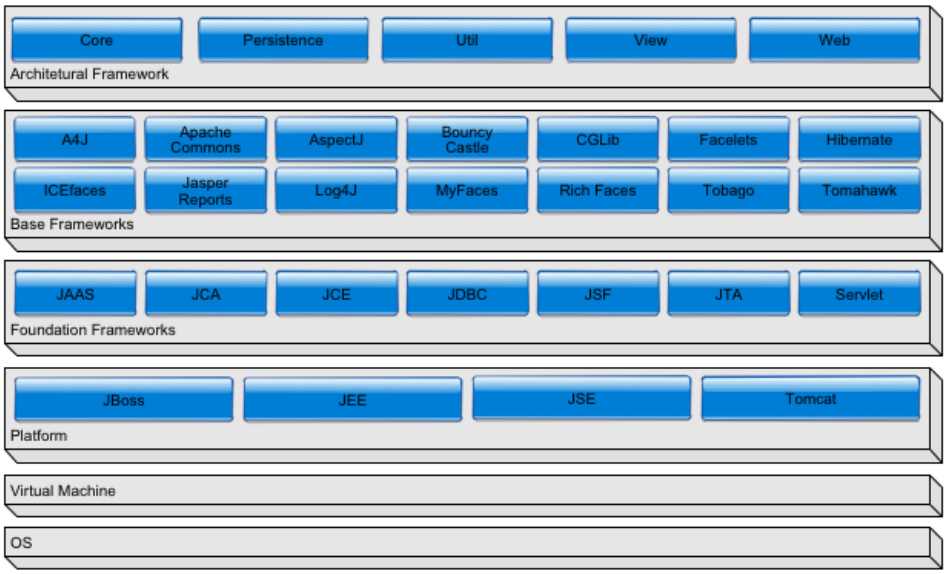
3.2.1.4.3 *Demoiselle Framework*

O Governo Federal por meio da SERPRO (Serviço Federal de Processamento de Dados) desenvolveu um *framework* visando uma maior padronização e reuso de aplicações em sistemas utilizados em órgãos federais. Esta plataforma foi denominada *Demoiselle* e segundo Lisboa (2009a) “[...] é essencialmente uma biblioteca central de módulos que atende às necessidades de infra-estrutura básica de uma aplicação web não distribuída.” Sobre as

principais características do *framework* Lisboa (2009a) analisa: “A adoção do Demoiselle pretende automatizar e acelerar a integração de sistemas, aumentar a produtividade e eliminar o retrabalho.”

A plataforma foi denominada em homenagem a um modelo de avião idealizado por Santos Dumont. Segundo Brasil (2009) “[...] Santos Dumont permitia a utilização, adaptação e cópia de seu trabalho.” Em função deste pensamento que segue os conceitos atuais do *software* livre, *Demoiselle* foi o nome mais apropriado para batizar este *framework*. É importante destacar que mesmo sendo concebido inicialmente para aplicações do governo, esta plataforma não se limita apenas a isso podendo ser usada livremente por qualquer entidade ou pessoa.

A arquitetura do *Demoiselle Framework* é divida em módulos que podem ser observados na Figura 7. O módulo *Core* possibilita fazer padronização, extensão e integração de camadas. O módulo *Persistence* é responsável pelo armazenamento e tratamento das informações. O módulo *Util* possui componentes que facilitam o trabalho do *framework*. O módulo *View* é responsável pela interface com o usuário. E finalmente o módulo *Web* que tem como responsabilidade prover o tratamento de sessões e requisições do usuário (LISBOA, 2009a).



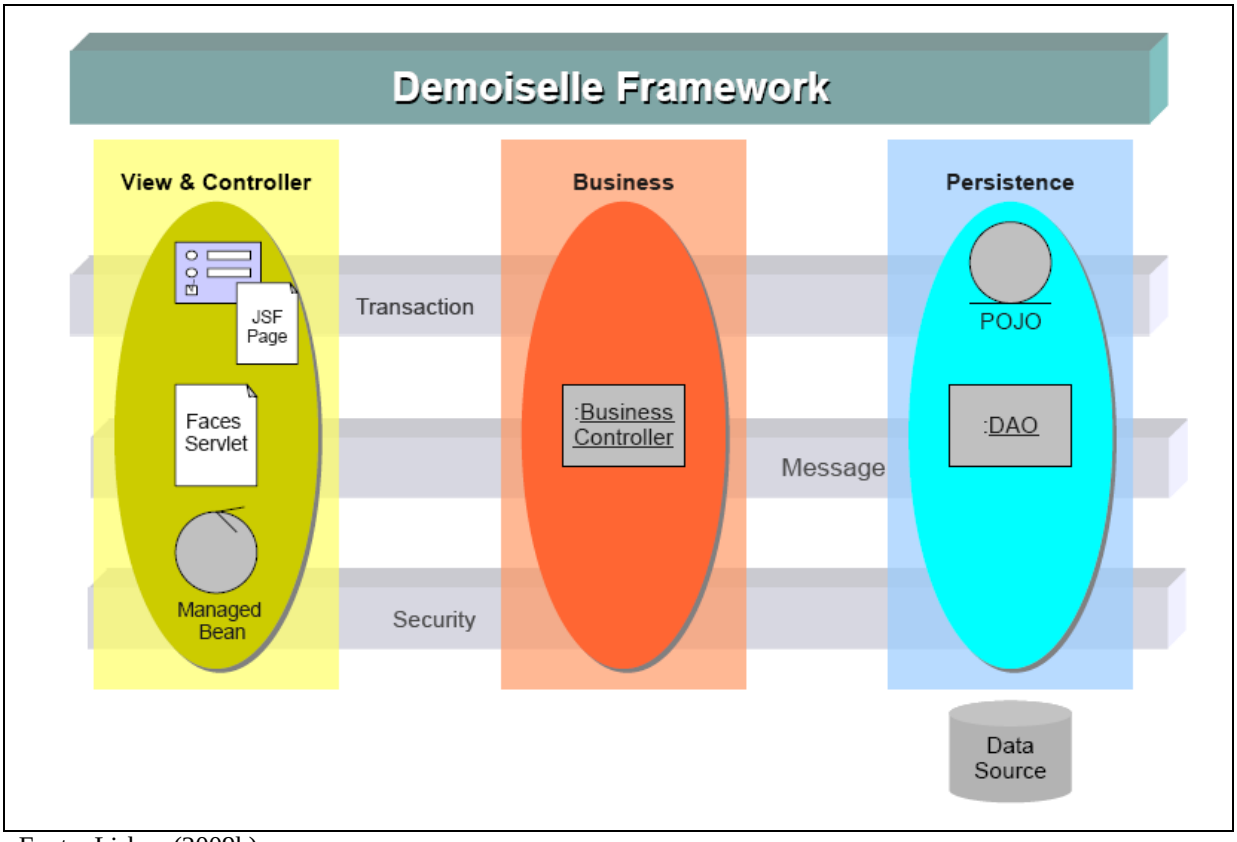
Fonte: Lisboa (2009b).

Figura 7 – Arquitetura do *Demoiselle Framework*

O *Demoiselle* foca a ideia de reaproveitamento e construção de *software* modular e escalável. O *framework* separa as suas principais características em camadas onde é possível separar responsabilidades e dar maior manutenibilidade ao código, sendo assim, há uma grande diminuição no tempo de aprendizagem, os processos se tornam mais simples, tornando fácil a reutilização de código, além de facilitar a manutenção dos sistemas (BRASIL, 2009).

Além de possuir as camadas clássicas do modelo MVC (Modelo, Visão e Controle), o *framework* ainda possui camadas de persistência, transação, segurança, injeção de dependência e mensagens (LISBOA, 2009a).

Segundo Lisboa (2009a), "Toda essa infraestrutura para as camadas tradicionais constitui o que chamamos de contextos". Na Figura 8 podem ser observados os contextos em camadas horizontais e a implementação MVC do *Demoiselle* em camadas transversais.



Fonte: Lisboa (2009b).

Figura 8 – Camadas verticais e horizontais do *Demoiselle Framework*

No ciclo de desenvolvimento do Argos, foi possível explorar as potencialidades do *Demoiselle Framework*, tais como padronização, reuso de código e métodos, baixo acoplamento, divisão em camadas e padrões de projetos. Esta estrutura em camadas facilitou a manutenção e a reutilização de códigos, diminuindo o impacto quando da necessidade do aumento do escopo de funcionalidades. Devido ao fato de incorporar em sua concepção diversos padrões e tecnologias especialistas, a utilização do *Demoiselle* proporcionou ganhos de produtividade e agilidade na execução do projeto.

Em vias gerais é possível afirmar que o *Demoiselle* se torna uma ótima opção para o desenvolvimento de aplicações voltadas para a *Web*, pois integra um conjunto de tecnologias que cooperam entre si buscando uma maior padronização, aumento da produtividade e diminuição do retrabalho.

3.2.1.4.4 *Apache Lucene*

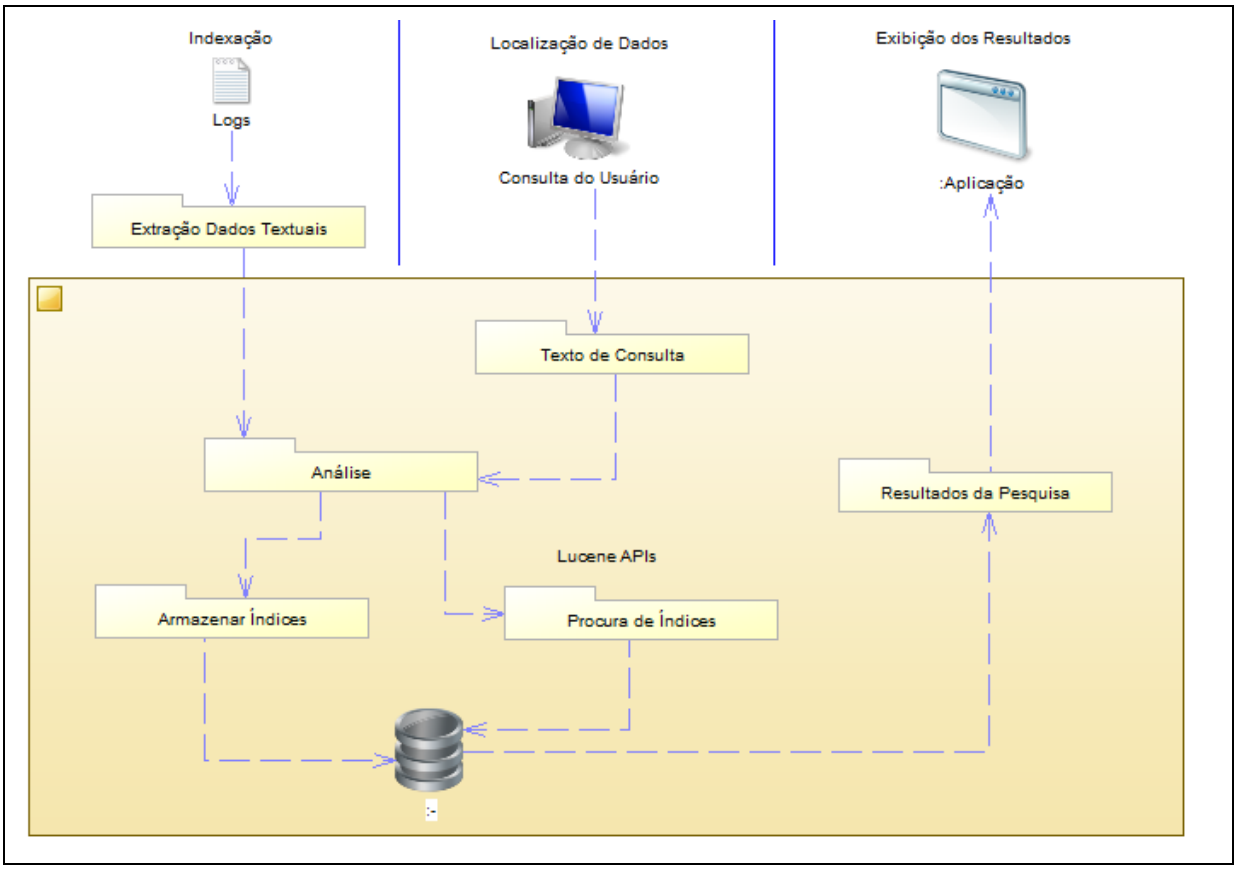
O *Lucene* trata-se de uma biblioteca de mecanismo de indexação e procura de texto altamente escalável escrito na linguagem de programação *Java*. Caracteriza-se por ser um *software* de código aberto da *Apache Software Foundation*[15] licenciado através da licença *Apache* que possibilita a sua utilização tanto para aplicativos comerciais como de *software* livre. O uso do *Lucene* é adequado para qualquer aplicação que requer pesquisa de texto completo, especialmente multi-plataforma (LUCENE, 2013).

As suas poderosas *APIs* (*Application Programming Interface* ou Interface de Programação de Aplicativos) podem ser usadas para criar recursos de procura para aplicações, como clientes de e-mail, listas de correspondências, procuras da *Web*, procuras de banco de dados, etc. Alguns *web sites* como *Wikipedia*[16], *TheServerSide*[17], *jGuru*[18] e *LinkedIn*[19] foram desenvolvidos com o *Lucene* (SONAWANE, 2013).

O *Lucene* dispõe de uma vasta quantidade de recursos que enriquecem uma aplicação. Seus algoritmos de procura são poderosos, precisos e eficientes. É capaz calcular uma pontuação para cada documento que corresponda a uma determinada consulta e retorna a

[15] http://www.apache.org/
[16] http://pt.wikipedia.org/
[17] http://www.theserverside.com/
[18] http://www.jguru.com/
[19] http://br.linkedin.com/

maioria dos documentos relevantes classificados por essas pontuações. Suporta a análise de expressões de consulta completas digitadas pelo usuário. Permite que os usuários estendam o comportamento da procura usando classificação, filtragem e análise de expressão de consulta. Usa um mecanismo de bloqueio baseado em arquivo para impedir modificações de índices simultâneos. Além de permitir a procura e a indexação simultânea (SONAWANE, 2013).



Fonte: O Autor (2013).

Figura 9 – Funcionamento do *Apache Lucene* no Argos

No Argos o *Lucene* é de fundamental importância, pois todos os registros de *logs* enviados são indexados e buscados por meio de suas *APIs*. Como mostra a Figura 9, o mecanismo de procura de *logs* do Argos envolve, primeiramente, a indexação dos registros de *logs*, posteriormente a procura de informações referente aos *logs* e por fim a exibição dos resultados da procura. No momento da indexação o *Lucene* faz uma análise dos dados para facilitar a procura que é realizada baseada em termos fornecidos pelo usuário.

3.2.1.4.5 *Spring Data*

Spring Data é um projeto da *Spring Source* e possui como objetivo unificar e facilitar o acesso a diferentes tecnologias de armazenamento de dados. Ele facilita a criação de aplicativos que usam novas tecnologias de acesso a dados, tais como banco de dados não relacionais (*NoSQL*), *Map-Reduce frameworks* (modelo de programação para grandes conjuntos de dados) , e serviços de dados baseados em nuvem, bem como fornece suporte aperfeiçoado para tecnologias de banco de dados relacionais (SPRINGSOURCE, 2013).

O *Spring Data* se caracteriza por possuir uma estrutura independente da solução de armazenamento, estas classes de "repositório" (também conhecidas como *Data Access Objects* ou DAOs) disponibilizam operações CRUD (*Create-Read-Update-Delete*) para um determinado objeto de domínio. Além disso, disponibiliza métodos de consulta (*Criteria* API que possibilita construir de forma programática, objetos *query expression* em *Java*) e funcionalidades de ordenação e paginação. Outros aspectos são suportados pelo *Spring Data* como o uso de *templates* e mapeamento de objetos para a estrutura de armazenamento do banco de dados (TRELLE, 2013).

Na construção do Argos o *Spring Data* possibilitou o mapeamento dos objetos de domínios para realização da persistência com o banco de dados não relacional *MongoDB*. Ele faz esta integração com bancos *NoSQL* que possuem estrutura similares a objetos, como é o caso do *MongoDB*. Além disso, a possibilidade de montagem de *query expression* em *Java* para consultas no banco de dados facilitou bastante o desenvolvimento, dando legibilidade ao código e desempenho a aplicação.

3.2.1.4.6 *MongoDB*

Atualmente percebe-se um crescente interesse em sistemas de gerenciamento de bancos de dados que diferem do modelo relacional tradicional. Neste novo contexto surge o conceito *NoSQL*, um termo usado para classificar um *software* de banco de dados que não utiliza SQL (*Structured Query Language*) para interagir com as informações do banco de dados. Um dos projetos *NoSQL* mais notáveis e quem vem sendo utilizado em larga escala até o momento é o *MongoDB* (LENNON, 2013).

O *MongoDB* é um banco de dados não relacional, de código aberto e de alta performance. Diferencia-se dos demais bancos de dados pelo fato de não possuir esquemas e armazenar os dados em coleções de documentos semelhantes a JSON (*JavaScript Object Notation*) que retém os dados usando pares de chave/valor. É escrito na linguagem C++ e possui como premissa ser ágil e altamente escalável (MONGODB, 2013).

Um recurso de destaque do *MongoDB* é a facilidade em converter instruções SQL (utilizadas em banco de dados relacionais) em chamadas de função de consulta específicas do *MongoDB*. Isso facilita a migração de organizações que atualmente usam bancos de dados relacionais, além de facilitar bastante o desenvolvimento, principalmente quando o programador já detém um bom conhecimento de SQL. Além disso, ele é muito simples de instalar e usar, com binários e *drivers* disponíveis para os principais sistemas operacionais e linguagens de programação (LENNON, 2013).

No Argos o *MongoDB* se mostrou um ótimo *software* gerenciador de banco de dados. Foi possível montar consultas que permitem localizar dados usando qualquer critério de qualquer atributo de um documento (objeto de domínio). Os resultados destas consultas são armazenados em cursores que fornecem diversas funções de filtragem, agregação e classificação. Além disso, o seu suporte a indexação permitiu criar índices em atributos específicos tornando as consultas mais rápidas.

3.2.1.4.7 *Glassfish*

GlassFish é um servidor de aplicações baseado na plataforma *Java* e tecnologia *Enterprise Edition* (*Java* EE) para o desenvolvimento e execução de aplicações e serviços *Web*. Seu uso vem crescendo de forma exponencial devido a alguns aspectos positivos como: desempenho, confiabilidade, produtividade e facilidade. É um servidor de aplicação de código aberto (*open source*) disponível gratuitamente e está licenciado sob *Common Development and Distribution License* (CDDL) (PELEGRI-LLOPART et al., 2007);

No Argos, o *Glassfish* ofereceu toda a infraestrutura de serviços para a execução da aplicação. Estes serviços diminuíram a complexidade do desenvolvimento, controlaram o fluxo de dados, aprimoraram o desempenho e gerenciaram a segurança. Além disso, por meio do *Glassfish* foi possível oferecer conexão segura HTTPS (*HyperText Transfer Protocol Secure*) para recebimento de *logs* de forma criptografada.

3.2.2 Arquitetura do Argos

O Argos foi projetado e desenvolvido tendo em vista um baixo acoplamento, desta forma, a manutenção e a adição de novas funcionalidades é realizada de forma mais simplificada e menos impactante. Neste sentido, como ponto de partida aconteceu uma separação do projeto em três camadas distintas: camada de visão (*View*), camada de negócio (*Controller*) e camada de persistência (*Model*). Estas camadas são apresentadas na forma de pacotes com funcionalidades em comum. Esta estrutura de pacotes tem o objetivo de manter a integração entre as camadas do sistema com um baixo acoplamento cujo resultado é maior facilidade no momento de efetuar a manutenção nas classes, pois elas estão organizadas tanto em sua estrutura como em seu código fonte.

Estrutura do Argos – Função de Cada Pacote
br.com.ezequieljuliano.argos.app
Pacote onde estão localizadas as classes de *Startup* (inicialização) da aplicação.
br.com.ezequieljuliano.argos.business
Pacote onde estão localizadas as classes de regras de negócio da aplicação.
br.com.ezequieljuliano.argos.config
Pacote onde estão localizadas as classes de configurações da aplicação.
br.com.ezequieljuliano.argos.constant
Pacote onde estão localizadas as classes de constantes da aplicação.
br.com.ezequieljuliano.argos.domain
Pacote onde estão localizadas as classes de domínios (POJOs - *Plain Old Java Objects*) da aplicação.
br.com.ezequieljuliano.argos.exception
Pacote onde estão localizadas as classes de exceções da aplicação.
br.com.ezequieljuliano.argos.message
Pacote onde estão localizadas as classes de mensagens da aplicação.
br.com.ezequieljuliano.argos.persistence
Pacote onde estão localizadas as classes de persistência (salvar e recuperar informações do banco de dados) da aplicação.
br.com.ezequieljuliano.argos.security
Pacote onde estão localizadas as classes que implementam os mecanismos de segurança da aplicação.
br.com.ezequieljuliano.argos.service
Pacote onde estão localizadas as classes que provém serviços HTTP e *RESTful* da aplicação.

br.com.ezequieljuliano.argos.service.to
Pacote onde estão localizadas as classes utilizadas para transferência de dados da aplicação.
br.com.ezequieljuliano.argos.statistics
Pacote onde estão localizadas as classes que retornam os dados para os gráficos do *dashboard*.
br.com.ezequieljuliano.argos.util
Pacote onde estão localizadas as classes que contém bibliotecas auxiliares da aplicação.
br.com.ezequieljuliano.argos.view
Pacote onde estão localizadas as classes que controlam a interface da aplicação.
br.com.ezequieljuliano.argos.view.converter
Pacote onde estão localizadas as classes que realizam conversão de objetos para a camada de visão da aplicação.

Fonte: O Autor (2013).

Quadro 2 - Função de cada pacote da estrutura do Argos

Como pode ser observado no Quadro 2, cada pacote possui uma função específica dentro do sistema e são separados por responsabilidades em comum. Toda essa estruturação fornece inúmeros benefícios, pois a partir do momento que o projeto começa a ganhar corpo, é extremamente importante manter uma organização entre as classes, para não haver trechos de códigos ilegíveis e de difícil entendimento.

3.2.2.1 Camada de Persistência (*Model*)

Esta camada tem por objetivo garantir transparência às demais camadas da aplicação na manipulação e tratamento das informações providas do Sistema Gerenciador de Banco de Dados. Nela acontecem as principais operações de um DAO (*Data Access Object*), as implementações JDBC (*Java Database Connectivity*) e o mapeamento dos objetos de domínios para que assim seja possível persisti-los junto ao *MongoDB*.

A camada de persistência engloba os seguintes pacotes:

- br.com.ezequieljuliano.argos.*domain*;
- br.com.ezequieljuliano.argos.*persistence*;

O pacote *Domain* representa as classes do tipo POJO (*Plain Old Java Object*), que são classes simples independentes em relação ao container da aplicação. Utilizando o *Spring Data*

(*Spring Framework)* foi realizado o mapeamento destes objetos via *Annotation* (Anotação), conforme demonstra a Figura 10, desta forma, é possível salvar e recuperar as informações dos objetos existentes no banco de dados.

```
@Document //Anotação que indica ser um documento do MongoDB
public class Usuario implements Serializable {

    @Id //Anotação que identifica a chave primária
    private String id;

    @Indexed //Anotação que define um índice para o campo
    private String userName;

    private String password;
    private UsuarioPerfil perfil = UsuarioPerfil.normal;
    private Situacao situacao = Situacao.ativo;

    @Indexed
    private String email;

    @Indexed
    private String apiKey;

    @DBRef //Anotaçãoq que faz referência a outro documento do MongoDB
    private Entidade entidade;

    private List<UsuarioTermoPesquisa> termosNotificacao = new ArrayList<UsuarioTermoPesquisa>();

}
```

Fonte: O Autor (2013).

Figura 10 – Mapeamento do *domain* usuário com *annotation*

Conforme pode ser acompanhado na Figura 11, o pacote *Persistence* contém classes que implementam métodos que realizam o tratamento das informações junto ao banco de dados *NoSQL* (banco de dados não relacional) *MongoDB*. O *Spring Data* fornece bibliotecas (*Criteria*, *Query* entre outras) que facilitam o trabalho de consultar e manipular informações providas do banco de dados.

```
@Repository
public class UsuarioDAO extends GenericDAO<Usuario, String> {

    private static final long serialVersionUID = 1L;

    public Usuario findByUserName(String userName) {
        Query query = new Query(Criteria.where("userName").is(userName));
        return getMongoOperations().findOne(query, Usuario.class);
    }

    public List<Usuario> findListByUserName(String userName) {
        Query query = new Query(Criteria.where("userName").regex(userName, "i"));
        return getMongoOperations().find(query, Usuario.class);
    }

    public Usuario findByEmail(String email) {
        Query query = new Query(Criteria.where("email").is(email));
        return getMongoOperations().findOne(query, Usuario.class);
    }

    public Usuario findByApiKey(String apiKey) {
        Query query = new Query(Criteria.where("apiKey").is(apiKey));
        return getMongoOperations().findOne(query, Usuario.class);
    }

    public Usuario login(String userName, String password) {
        Query query = new Query(Criteria.where("userName").is(userName).and("password").is(password));
        return getMongoOperations().findOne(query, Usuario.class);
    }
}
```

Fonte: O Autor (2013).

Figura 11 - Classe do tipo DAO (*Data Access Object*) do usuário

É possível perceber na Figura 11 que a classe UsuarioDAO estende a classe *Generic*DAO. Esta classe genérica implementa todos os métodos comuns para as classes do tipo DAO (*Data Access Object*) como inserção, alteração ou exclusão de informações referente aos objetos de domínio. Com este tipo de implementação é possível reduzir a quantidade de código gerado, bem como obter uma maior facilidade na manutenção do escopo de funcionalidades.

3.2.2.2 Camada de Negócio (*Controller*)

Esta camada é responsável por implementar as regras de negócio definidas para o sistema especificando os processos de integração entre os módulos. Esta integração com as demais camadas ocorre por meio do mecanismo de injeção de dependência. No Argos a injeção de dependência ocorre de duas formas: pelo *Spring Data* e pelo *Demoiselle*.

A injeção de dependência pelo *Spring Data* é feita na camada de persistência integrando todas as classes deste container. Pelo *Demoiselle* a injeção de dependência é controlada na camada de negócio e visão. Na arquitetura do *Demoiselle* é o *Core*[20] que especifica quem trata a injeção de dependência, os módulos que implementam o *Core* definem como a injeção será realizada e o módulo *Web* realiza a sua implementação.

O pacote br.com.ezequieljuliano.argos.*business* é o repositório base da camada de negócio da aplicação. Ela é responsável por implementar as regras de negócio do sistema integrando os processos entre os módulos. Nesta camada o tratamento das informações possibilita que os dados sejam enviados para a camada de persistência de forma mais consistente, diminuindo a possibilidade de armazenamento de informações incoerentes.

```
//CAMADA DE PERSISTÊNCIA
@Repository
public class EventoDAO extends GenericLuceneDAO<Evento, String> {
    @Autowired
    private EntidadeDAO entidadeDAO;
}
//CAMADA DE NEGÓCIO
@BusinessController
public class EventoBC extends GenericBC<Evento, String, EventoDAO> {
    public void saveOrUpdate(Evento evento, Usuario usuario) {
        if (evento.getId() == null) {
            getDAO().insert(evento);
        } else {
            getDAO().save(evento); //INTEGRAÇÃO COM A CAMADA DE PERSISTÊNCIA
        }
        //Insere Usuário relacionado ao Evento
        gravarUsuarioEvento(evento, usuario);
        //Envia notificação caso esteja configurado para isto
        enviarEmailComNotificacaoParaUsuario(evento, usuario);
    }
}
//CAMADA DE VISÃO
@ViewController
public class EventoPesquisaMB {
    @Inject
    private EventoBC eventoBC; //INTEGRAÇÃO COM A CAMADA DE NEGÓCIO
}
```

Fonte: O Autor (2013).

Figura 12 – Integração entre as camadas MVC do Argos

[20] O *Core* do *Demoiselle* contém aquelas funcionalidades que são comuns a todas as extensões e aplicações do *framework*.

Conforme pode ser acompanhado na Figura 12, a camada de negócio é o grande controlador do sistema, pois ela recebe as informações providas da camada de visão, realiza o tratamento destas informações e caso haja a necessidade de persistir ou buscar dados repassa esta tarefa a camada de persistência. A camada de negócio, da mesma forma que a camada de persistência, implementa uma classe genérica responsável por executar métodos comuns entre todas as classes do tipo BC (*Bussiness Controller*).

3.2.2.2.1 Segurança

A segurança sempre é um ponto muito importante para qualquer sistema, no caso de um sistema de gestão de *logs* não é diferente, pois muitas informações confidenciais se encontram nestes arquivos e, desta forma, é extremamente importante dar a devida relevância a este assunto. Sendo assim, no Argos o controle de segurança é baseado no componente *Demoiselle Security* que fornece mecanismos de autenticação e autorização de acesso ao escopo de funcionalidades do *software*.

Este componente de segurança do *Demoiselle* fornece mecanismos de autorização baseado em especificações JAAS (*Java Authentication and Authorization Service*). O JAAS é uma API que permite que aplicações escritas em J2EE usem serviços de autenticação e autorização sem a necessidade de estarem diretamente dependentes desses serviços. Para ter acesso às demais funcionalidades do sistema este componente pode ser utilizado via injeção de dependência possibilitando a sua utilização em qualquer camada da aplicação. Para garantir o acesso seguro do Argos, foram reimplementadas duas interfaces do *Demoiselle*: a *Authenticator* e a *Authorizer*.

```
public class Autenticador implements Authenticator {
    @Inject
    private UsuarioBC usuarioBC;
    @Inject
    private SessionAttributes sessionAttributes;
    @Override
    public boolean authenticate() {
        Usuario retorno = usuarioBC.login(loginMB.getUsuario().getUserName(), loginMB.getUsuario().getPassword());
        if (retorno == null) {return false;}
        sessionAttributes.setUsuario(retorno);
        return true;
    }
    @Override
    public void unAuthenticate() {sessionAttributes = null;}
}
public class Autorizador implements Authorizer {
    @Inject
    private SessionAttributes sessionAttributes;
    @Override
    public boolean hasRole(String requiredRole) {
        Usuario usuarioLogado = sessionAttributes.getUsuario();
        UsuarioPerfil perfil = usuarioLogado.getPerfil();
        Boolean autorizado = false;
        if (perfil.equals(UsuarioPerfil.administrador)) {autorizado = true;}
        else if (getUsuarioPerfil(requiredRole).equals(UsuarioPerfil.normal) && perfil.equals(UsuarioPerfil.normal)) {
            autorizado = true;
        }
        if (requiredRole.startsWith("@") && autorizado == false) {
            throw new AcessoNegadoException("Você não está autorizado a acessar este recurso!");
        }
        return autorizado;
    }
```

Fonte: O Autor (2013).

Figura 13 - Classes de autenticação e autorização da camada de segurança do Argos

Conforme pode ser visto na Figura 13, a classe Autenticador que implementa a interface *Authenticator*, possui métodos para garantir o acesso seguro via *login* e senha, além de retornar o usuário da sessão ativa e efetuar *logout* do sistema. A classe Autorizador que implementa a interface *Authorizer*, é responsável por verificar as permissões do usuário *logado*, definindo seu acesso as funcionalidades do sistema.

Quando ocorre uma violação de uma regra de autorização o Argos levanta uma exceção do tipo *AcessoNegadoException*. As exceções levantadas quando utilizado o *Demoiselle* podem ser do tipo "*Checked*" (necessário o tratamento) e "*Un-Checked*" (não força o tratamento). Cada componente pode gerar suas exceções, a mais comum delas é a *ApplicationRuntimeException*. Ao realizar tratamento das exceções é interessante mostrar mensagens mais amigáveis aos usuários, isto é possível através dos contextos de mensagens. Estes contextos possuem como característica efetuar a troca de mensagens entre as camadas da aplicação utilizando a interface *MessageContext*.

3.2.2.2.2 Recebimento de *Logs*

O *software* Argos é composto por diversos módulos importantes que auxiliam na busca e análise de *logs*, mas o módulo de maior relevância trata-se do *Webservice* usando *RESTful*/*JSON* API que é responsável por receber as chamadas HTTP contendo os eventos de *logs* que devem ser armazenados e indexados.

Os *Webservices* se caracterizam por possuir padrões que garantem a interoperabilidade de um sistema e podem ser aplicados a diversas plataformas de integração, desde estruturas mais simples até sistemas distribuídos mais complexos. Figueiredo (2008) argumenta que "os Web Services são por vezes denominados de serviços de aplicações, sendo serviços que foram disponibilizados por servidores Web para utilizadores Web ou para programas ligados à Web, sendo os seus fornecedores denominados, na generalidade, de Application Service Providers".

O termo REST (*Representacional State Transfer)* vem da dissertação de doutorado de Roy Fielding, publicada no ano de 2000. Para Fielding (apud OLIVEIRA, 2009), REST é "um conjunto de princípios arquiteturais que quando aplicadas como um todo enfatiza a escalabilidade da interação entre componentes para reduzir a latência de interação, garantir segurança e encapsular sistemas legados". Fielding também cunhou o termo *RESTful* e o classificou com os seguintes princípios: ser cliente servidor; apoiar sistemas de cache; possuir estado nulo (cada requisição de um cliente ao servidor deve conter todas as informações necessárias para entender a requisição); ser *stateless* (comunicação sem controle de estado); além de ser um sistema uniforme composto por quatro diretivas padronizadas: *GET*, *PUT*, *POST*, *DELETE*.

```
@POST
@Path("/{apiKey}")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public EventoLogReturnTO efetuarLogging(@PathParam("apiKey") String apiKey, EventoLogTO log) {
    try {
        //Pega o usuário
        Usuario usuario = getUsuarioByApiKey(apiKey);
        //Pega a Entidade
        Entidade entidade = getEntidadeByUsuarioAndCadastroNacional(usuario, log.getCadastroNacional());
        //Pega o Evento Nível
        EventoNivel eventoNivel = getEventoNivelByCodigo(log.getEventoNivelCodigo());
        //Pega Evento Tipo
        EventoTipo eventoTipo = getEventoTipoByCodigo(log.getEventoTipoCodigo());
        //Insere os logs no sistema
        armazenarEventoLog(log, entidade, eventoNivel, eventoTipo, usuario);
    } catch (LogServiceException ex) {
        return new EventoLogReturnTO(ex.getLogEx().getCodigo(), ex.getLogEx().getDescricao());
    }
    //Se tudo ocorrer de forma correta retorna OK
    return new EventoLogReturnTO(0, "Logging Efetuado com Sucesso!");
}
```

Fonte: O Autor (2013).

Figura 14 - Método responsável por receber os eventos de *logs* de sistemas de terceiros

Conforme pode ser visto na Figura 14 o Argos disponibiliza um método em seu *Webservice* denominado *efetuarLogging* que recebe a chave de identificação da entidade e o evento de *log* que deve ser armazenado e indexado. Ao receber o *log* o sistema efetua algumas validações e caso não ocorra erros retorna a mensagem "*Logging* Efetuado com Sucesso!". Caso aconteça algum problema na inserção do *log* é levantada uma exceção retornando um erro específico ao sistema que o está enviando. Ao utilizar esta estrutura foi possível atribuir identificação as unidades de recurso, utilizar métodos padronizados e obter uma comunicação sem controle de estado, ou seja, sem haver dados da sessão do cliente armazenados no servidor.

3.2.2.3 Camada de Visão (*View*)

No Argos a camada de visão é representada pelo pacote br.com.ezequieljuliano.argos.*view* e pelas páginas *Web* da aplicação. A principal responsabilidade desta camada é apresentar as informações aos usuários e controlar suas interações com o sistema.

Com a premissa de possuir uma interface agradável, intuitiva e que apresentasse os dados de forma rápida e compatível com a maioria dos navegadores, optou-se pela utilização do *Java Server Faces* (JSF) na montagem das páginas. O JSF é um *framework* que visa facilitar o desenvolvimento de interfaces de usuário baseadas em ambiente *web*.

Para que o JSF funcione de forma correta, é necessário configurar os arquivos *web.xml* e *faces-config.xml* da aplicação. Além disso, é necessário criar as páginas com os componentes JSF mapeados (Figura 15) para as classes que fazem esta interação. Estas classes são denominadas *ManagedBeans*.

```
<p:fieldset legend="Nível de Evento">
    <h:panelGrid columns="2">
        <h:outputLabel value="Código*" for="codigo"/>
        <p:inputText value="#{eventoNivelMB.bean.codigo}" id="codigo" label="Código"
                     required="true" size="40"/>

        <h:outputLabel value="Descrição*" for="descricao"/>
        <p:inputText value="#{eventoNivelMB.bean.descricao}" id="descricao"
                     label="Descrição" required="true" size="40"/>
    </h:panelGrid>
</p:fieldset>
```

Fonte: O Autor (2013).

Figura 15 – Mapeamento de componente JSF com o *ManagedBean*

O JSF permite ao usuário interagir com a página no navegador executando alguma ação, consequentemente esta ação gera uma requisição HTTP, esta requisição é tratada pelo *ManagedBean* que esta mapeado no componente de página e repassada para as demais camadas da aplicação se houver necessidade. Posteriormente os dados processados são formatados e a página *Web* é construída exibindo a informação ao usuário.

3.2.2.3.1 Interface

Oferecer um *software* de qualidade capaz de realizar de forma centralizada, em nuvem e em tempo real o gerenciamento e análise de eventos de *logs* gerados por aplicações de

terceiros sempre foi o objetivo vislumbrado. Mas é extremamente importante estender este conceito de qualidade para a forma de como os dados são apresentados ao usuário gestor de TI. Preferencialmente isto deve ser feito por meio de uma interface simples e intuitiva.

Devido a grande interação com o usuário, o Argos possui uma interface enxuta e de fácil manipulação, contendo toda uma estrutura de cadastros auxiliares, além de telas específicas de consultas de eventos de *logs* e *dashboard* com gráficos trazendo informações cruciais ao usuário gestor.

Para acessar o sistema, obrigatoriamente o usuário necessita estar *logado*, isto é possível por meio da tela de acesso, conforme mostra a Figura 16. É importante destacar que o *software* conta com uma área de gerenciamento e controle de usuários, onde pode ser definido o perfil (Normal ou Administrador), sua entidade (usuários só podem ver *logs* referentes a sua entidade e entidades filhas), *API Key* (equivalente ao usuário senha para envio de *logs*) e termos considerados relevantes para o envio de notificações a cerca de eventos de *logs* recebidos.

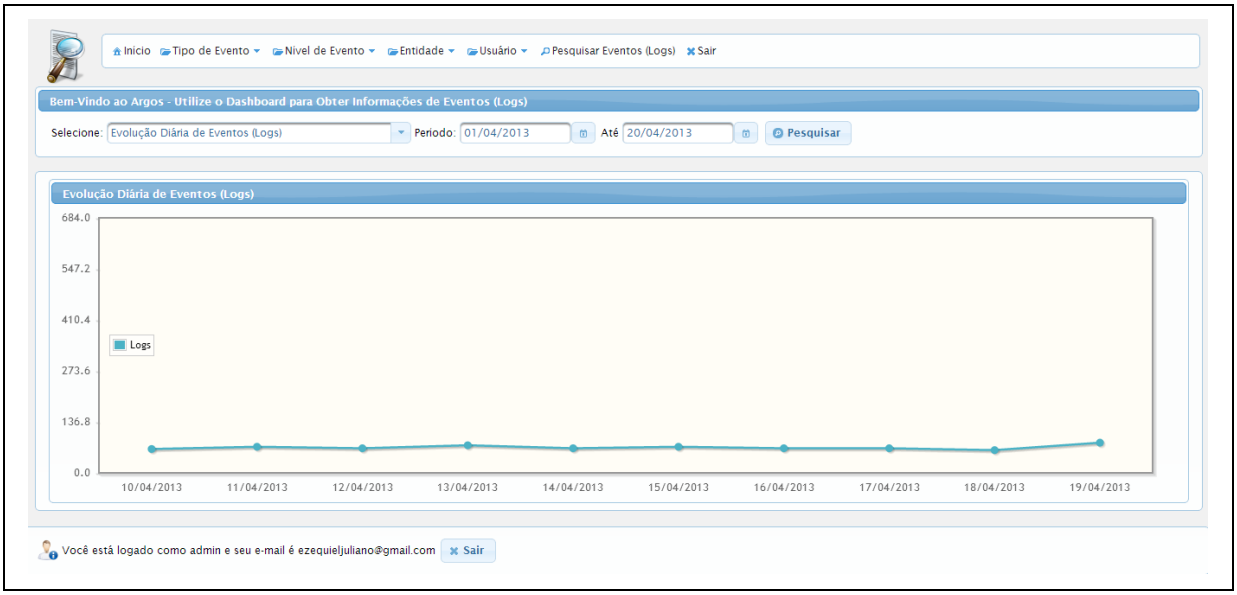
Fonte: O Autor (2013).

Figura 16 - *Login* de acesso ao sistema Argos

Na Figura 17 é apresentada a visualização do usuário após o *login*. O "Menu" superior fica sempre disponível, pois as páginas são carregados de forma dinâmica na parte central da tela, desta forma, o usuário tem mais facilidade para localizar as demais funcionalidades. Na tela inicial é disponibilizado um *dashboard* com gráficos contendo

informações cruciais ao usuário gestor de TI. Neste *dashboard* é possível pesquisar informações por um intervalo de datas e analisar estes dados sobre as seguintes perspectivas:

- Últimos Eventos (*Logs*) Recebidos;
- Evolução Diária de Eventos (*Logs*);
- Percentual de Eventos (*Logs*) por Níveis;
- Percentual de Eventos (*Logs*) por Tipos;
- Percentual de Eventos (*Logs*) por *Hosts*;
- Percentual de Eventos (*Logs*) por Usuários do Sistema;
- Percentual de Eventos (*Logs*) por Fontes;



Fonte: O Autor (2013).

Figura 17 - Tela inicial do Argos com *dashboard*

Conforme pode ser observado no Quadro 3, o Argos disponibiliza toda a infraestrutura de cadastros bases para o funcionamento das principais funcionalidades do sistema, além da tela de consulta de eventos (*logs*).

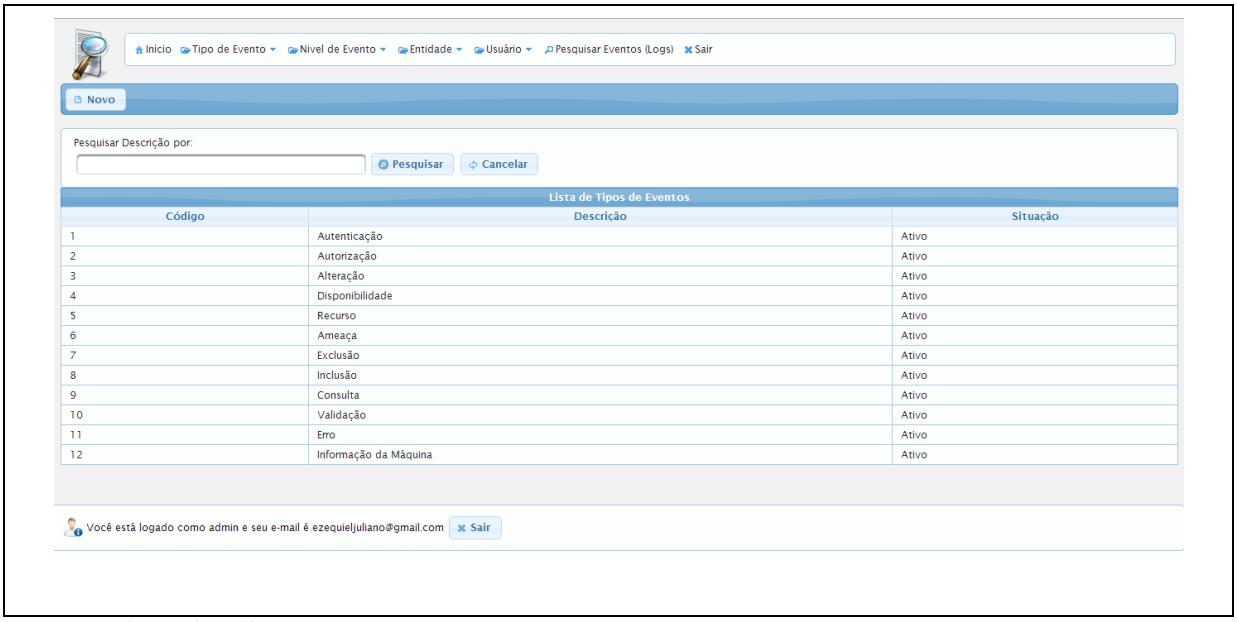
Estrutura do Menu Principal		
Argos	Início	
	Tipo de Evento	Novo

		Listar
	Nível de Evento	Novo
		Listar
	Entidade	Novo
		Listar
	Usuário	Novo
		Listar
	Pesquisar Eventos (Logs)	
	Sair	

Fonte: O Autor (2013).

Quadro 3 - Estrutura do menu principal do Argos

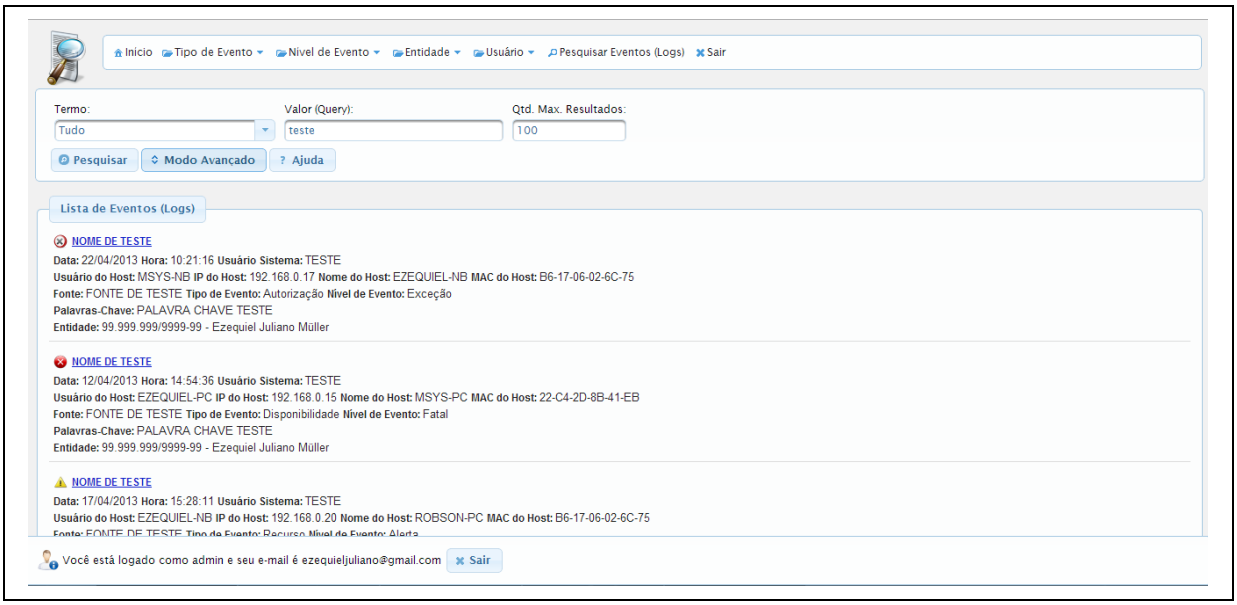
Exceto a tela do *dashboard* inicial e da pesquisa de eventos (*logs*), as demais telas do sistema seguem um mesmo padrão de manipulação de informações conforme mostra a Figura 18. Na parte superior esta situado o "Menu" de ações, possuindo procedimentos que servem para inserir, alterar, excluir ou executar alguma ação específica sobre um registro do sistema. Estas ações podem variar de acordo com a tela (listagem ou detalhamento). Na tela de listagem abaixo do "Menu" de ações existe uma funcionalidade de filtragem das informações e mais abaixo são apresentados os dados em forma de *grid* de dados. Na tela de detalhamento, abaixo do "Menu" de ações ficam os campos para inserção/alteração de dados.



Fonte: O Autor (2013).

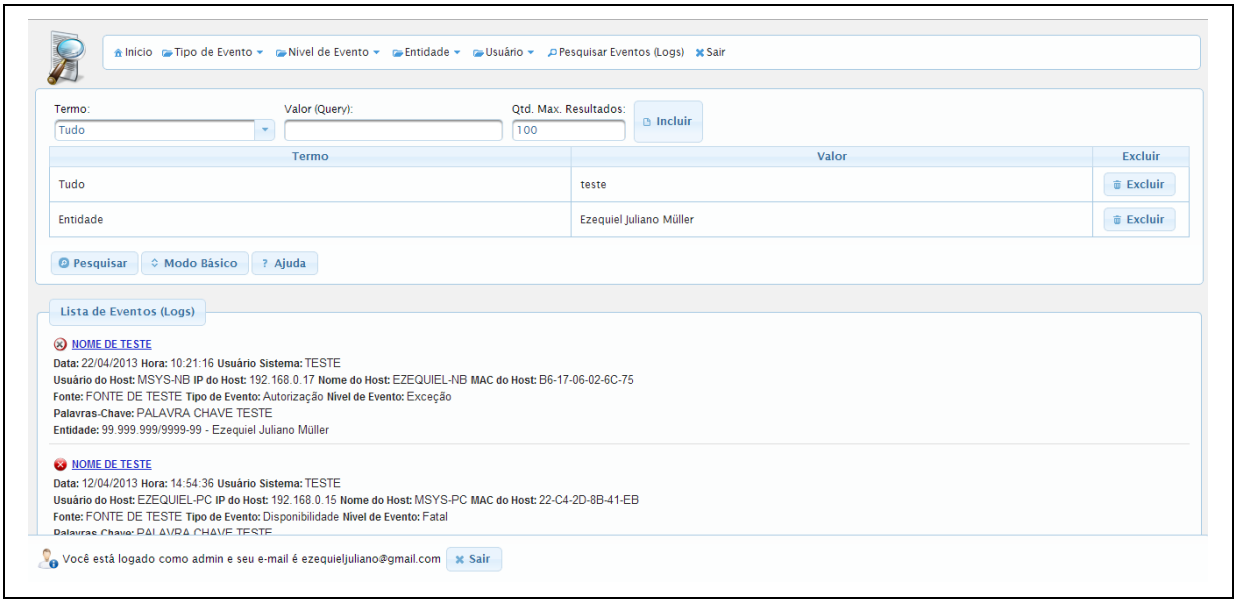
Figura 18 - Estrutura padrão de interface para manipulação de dados do Argos

O processo de pesquisa de eventos de *logs* é feito de duas formas: básico e avançado. No modo básico (Figura 19) é possível selecionar um termo (ao utilizar a opção “Tudo” todos os termos serão considerados), informar o valor da pesquisa e especificar a quantidade de retornos possíveis para melhorar o desempenho da consulta. No modo avançado (Figura 20) é possível especificar vários termos para uma mesma consulta minerando ainda mais os resultados obtidos. Os eventos de *logs* encontrados são exibidos em uma lista contendo informações resumidas e caso se deseje visualizar os dados em sua totalidade basta clicar sobre algum registro.



Fonte: O Autor (2013).

Figura 19 - Tela de pesquisa de eventos (*logs*) em modo básico



Fonte: O Autor (2013).

Figura 20 - Tela de pesquisa de eventos (*logs*) em modo avançado

Além da possibilidade de informar um texto livre no valor da pesquisa é possível ainda definir sintaxes de consultas mesclando vários valores. Estas sintaxes possibilitam minerar mais detalhadamente os resultados da pesquisa. Alguns exemplos de sintaxes que podem ser utilizadas:

- Sintaxe de consulta com todos os valores (*AND*): +Valor1 +Valor2 +Valor3;
- Sintaxe de consulta com frase exata: "Valor1 esta sendo usado";
- Sintaxe de consulta com pelo menos uma dos valores (*OR*): Valor1 Valor2 Valor3;
- Sintaxe de consulta sem os valores (*NOT*): -Valor1 -Valor2 -Valor3;
- Sintaxe de consulta para valor aproximado: "Valor1"~;
- Sintaxe de consulta entre um e outro valor: [Valor1 *TO* Valor2].

A padronização de telas, desde a definição de imagens, localização de componentes e tamanho de fontes, visa melhorar a experiência do usuário na utilização do sistema. Desta forma, o impacto da inserção de uma nova forma de trabalho é minimizada, tornando o aprendizado e uso do sistema mais simples e com uma boa taxa de aceitação.

3.2.3 Estudo de Caso do Argos

O estudo de caso do sistema foi realizado de maneira gradual, em um primeiro momento foi montado um ambiente de testes em uma rede local com o objetivo de identificar possíveis falhas na estrutura interna do *software*. Foram montados alguns casos de testes (apresentados na sequência) com a premissa de verificar se os requisitos propostos pela aplicação desenvolvida foram plenamente atendidos, possibilitando assim, avaliar se a aplicação estava apta à utilização em produção.

Os Quadros 4, 5 e 6 ilustram três casos de testes mostrando exemplos de envio de *logs* para o *Webservice* do Argos. O objetivo do primeiro caso de teste era verificar o envio de um evento de *log* válido com usuário existente e ativo, ou seja, um fluxo normal do sistema. O segundo e o terceiro caso de teste objetivaram analisar o tratamento do sistema quando um evento de *log* é enviado com um usuário inexistente ou inativo.

Operação	Envio de eventos de logs com usuário existente e ativo
Propósito	Testar o envio de eventos de logs em um fluxo normal do sistema
Descrição	Este teste verificará o fluxo normal do envio de eventos de logs com um usuário devidamente cadastrado e ativo e uma mensagem de log válida
Entrada	
API Key Usuário = bc79d79a9387fb563f6cd65a1b80241b *Log* = {"hostName":"EZEQUIEL-NB","hostIp":"192.168.56.1","hostUser":"Ezequiel","hostMac":"08-00-27-00-2C-A0","sysUser":"EZEQUIEL","cadastroNacional":"99.999.999/9999-99","mensagem":"UPDATE CLIENTES SET CLIENTES.CLI_NOME = "MULLER SA", CLIENTES.CLI_CNPJ = "99.999.999/9999-99" WHERE CLIENTES.CLI_CODIGO = 10; OLD_VALUE.CLI_NOME = "EMPRESA X' OLD_VALUE.CLI_CNPJ = "11.111.111/1111-11"","fonte":"FormClientes","nome":"Alteração no Cliente 10","ocorrenciaDtHr":"7/6/2013 10:10:15","palavrasChave":"UPDATE;CLIENTES;EZEQUIEL","eventoNivelCodigo":1,"eventoTipoCodigo":1}	
Retorno Esperado	
0 - Logging Efetuado com Sucesso!	
Retorno Obtido	
0 - Logging Efetuado com Sucesso!	
Resultado	
Sucesso	

Fonte: O Autor (2013).

Quadro 4 – Caso de teste envio de *logs* com fluxo normal

Operação	Envio de eventos de logs com usuário inexistente
Propósito	Testar o envio de eventos de logs utilizando um usuário inexistente para o sistema
Descrição	Este teste verificará o envio de eventos de logs com a utilização de um usuário inexistente para verificar o tratamento que é realizado pelo sistema neste caso
Entrada	
API Key Usuário = 11111111111111111111111 *Log* = {"hostName":"EZEQUIEL-NB","hostIp":"192.168.56.1","hostUser":"Ezequiel","hostMac":"08-00-27-00-2C-A0","sysUser":"EZEQUIEL","cadastroNacional":"11.111.111/1111-11","mensagem":"DELETE FROM CLIENTES WHERE CLIENTES.CLI_CODIGO = 100","fonte":"FormClientes","nome":"Exclusão do Cliente 100","ocorrenciaDtHr":"7/6/2013 10:08:33","palavrasChave":"EXCLUSAO;CLIENTE;100","eventoNivelCodigo":1,"eventoTipoCodigo":1}	
Retorno Esperado	
1 - Usuário Inválido	
Retorno Obtido	
1 - Usuário Inválido	
Resultado	
Sucesso	

Fonte: O Autor (2013).

Quadro 5 – Caso de teste envio de *logs* com usuário inexistente

Operação	Envio de eventos de logs com usuário inativo
Propósito	Testar o envio de eventos de logs utilizando um usuário inativo no sistema
Descrição	Este teste verificará o envio de eventos de logs com a utilização de um usuário inativo no sistema para verificar o tratamento que é realizado neste caso
Entrada	
API Key Usuário = 5f5e56853d894c5048637a40941fa9b6 *Log* = {"hostName":"EZEQUIEL-NB","hostIp":"192.168.56.1","hostUser":"Ezequiel","hostMac":"08-00-27-00-2C-A0","sysUser":"EZEQUIEL","cadastroNacional":"11.111.111/1111-11","mensagem":"DELETE FROM CLIENTES WHERE CLIENTES.CLI_CODIGO = 999","fonte":"FormClientes","nome":"Exclusão do Cliente 999","ocorrenciaDtHr":"7/6/2013 11:09:35","palavrasChave":"EXCLUSAO;CLIENTE;999","eventoNivelCodigo":1,"eventoTipoCodigo":1}	
Retorno Esperado	
6 - Usuário está Inativo	
Retorno Obtido	
6 - Usuário está Inativo	
Resultado	
Sucesso	

Fonte: O Autor (2013).

Quadro 6 – Caso de teste envio de *logs* com usuário inativo

O Quadro 7 ilustra um caso de teste de envio de um evento de *log* inválido para o *Webservice* do Argos. O objetivo deste caso de teste foi verificar o comportamento do sistema

quando um evento de *log* com formato inválido é recebido. Este tipo de validação tem grande valor para aumentar a confiabilidade do *software*.

Operação	Envio de evento de log inválido
Propósito	Testar o envio de um evento de log inválido
Descrição	Este teste verificará o envio de um evento de log inválido para o sistema para verificar o tratamento que é realizado neste caso
Entrada	
API Key Usuário = bc79d79a9387fb563f6cd65a1b80241b *Log* = {"HOST_ERRO":"EZEQUIEL-NB","IP_OLD":"192.168.56.1","USUÁRIO":"Ezequiel"}	
Retorno Esperado	
Log Inválido	
Retorno Obtido	
Log Inválido	
Resultado	
Sucesso	

Fonte: O Autor (2013).

Quadro 7 – Caso de teste envio de *log* inválido

Os Quadros 8, 9, 10 e 11 ilustram quatro casos de testes mostrando exemplos de tentativas de acesso a interface de consulta do Argos. O objetivo do primeiro caso de teste era verificar se o sistema está ativo e operante. O segundo e o terceiro casos de testes possuíam como objetivo verificar o tratamento do sistema quando um usuário inexistente ou inativo tentava acessar a interface de consulta. E o quarto caso de tese verificou se o sistema estava implementando mecanismos de segurança para impedir acesso indevido por meio de uma técnica comum de invasão: o SQL *Injection*.

Operação	Acesso a interface de consulta com usuário existente e ativo
Propósito	Testar o acesso ao sistema
Descrição	Este teste verificará se o acesso ao sistema está operante
Entrada	
Usuário = admin *Senha* = admin	
Retorno Esperado	
Acesso ao Sistema Liberado	
Retorno Obtido	
Acesso ao Sistema Liberado	
Resultado	

Sucesso

Fonte: O Autor (2013).

Quadro 8 – Caso de teste acesso a interface de consulta com usuário ativo

Operação	Acesso a interface de consulta com usuário inexistente
Propósito	Testar o acesso ao sistema com usuário não cadastrado
Descrição	Este teste verificará se o sistema está tratando usuários não cadastrados
Entrada	
Usuário = super *Senha* = super123	
Retorno Esperado	
Mensagem de Erro	
Retorno Obtido	
Mensagem de Erro	
Resultado	
Sucesso	

Fonte: O Autor (2013).

Quadro 9 – Caso de teste acesso a interface de consulta com usuário inexistente

Operação	Acesso a interface de consulta com usuário inativo
Propósito	Testar o acesso ao sistema com usuário inativo
Descrição	Este teste verificará se o sistema está tratando usuários inativos
Entrada	
Usuário = ezequiel *Senha* = pass123	
Retorno Esperado	
Mensagem de Erro	
Retorno Obtido	
Mensagem de Erro	
Resultado	
Sucesso	

Fonte: O Autor (2013).

Quadro 10 – Caso de teste acesso a interface de consulta com usuário inativo

Operação	Tentativa de acesso ao sistema via SQL Injection
Propósito	Testar se o sistema está implementando mecanismos de segurança para impedir acesso indevido
Descrição	Este teste verificará se o sistema está tratando tentativas de acesso indevido por meio de uma técnica comum para este fim
Entrada	
Usuário = admin *Senha* = ' or '1' = '1	

Retorno Esperado
Mensagem de Erro
Retorno Obtido
Mensagem de Erro
Resultado
Sucesso

Fonte: O Autor (2013).

Quadro 11 – Caso de teste tentativa de acesso a interface de consulta via SQL *Injection*

Os Quadros 12 e 13 ilustram dois casos de testes mostrando exemplos de consultas de eventos de *logs* no sistema Argos. O primeiro caso de teste teve por objetivo verificar se o sistema está possibilitando ao usuário gestor efetuar uma consulta de eventos de *logs* em modo básico (termo + valor de consulta). O segundo caso de teste teve por objetivo verificar se o sistema está possibilitando ao usuário gestor efetuar uma consulta de eventos de *logs* em modo avançado (junção de vários termos + valor de consulta).

Operação	Consulta de eventos de logs em modo básico
Propósito	Testar se o sistema está possibilitando a consulta de eventos de logs em modo básico e em todos os termos
Descrição	Este teste verificará se o sistema está oferecendo a possibilidade do usuário gestor efetuar uma consulta de eventos de logs em modo básico e em todos os termos possíveis. O teste vai procurar uma alteração feita no cliente de código 77807
Entrada	
Termo = TUDO *Valor de Consulta* = +UPDATE +CLIENTES +77807	
Retorno Esperado	
Localização e Listagem de Logs	
Retorno Obtido	
Localização e Listagem de Logs	
Resultado	
Sucesso	

Fonte: O Autor (2013).

Quadro 12 – Caso de teste consulta de eventos de *logs* em modo básico

Operação	Consulta de eventos de logs em modo avançado
Propósito	Testar se o sistema está possibilitando a consulta de eventos de logs em modo avançado possibilitando a junção de vários termos
Descrição	Este teste verificará se o sistema está oferecendo a possibilidade do usuário gestor efetuar uma consulta de eventos de logs em modo avançado com junção de vários termos. O teste vai procurar uma alteração realizada no produto de código 60416, feita pelo usuário EZEQUIEL e na empresa MULLER SA

Entrada
Termos: *Entidade* = MULLER AS *Usuário do Sistema* = EZEQUIEL *Mensagem* = +UPDATE +PRODUTOS +60416
Retorno Esperado
Localização e Listagem de Logs
Retorno Obtido
Localização e Listagem de Logs
Resultado
Sucesso

Fonte: O Autor (2013).

Quadro 13 – Caso de teste consulta de eventos de *logs* em modo avançado

Após a realização destes testes, verificou-se que a aplicação estava apta para a utilização em produção, desta forma, foi contratado um serviço de *cloud computing* (computação em nuvem) para possibilitar a implantação do sistema em modo SaaS (*Software* como serviço). Devido ao fato de utilizar o modelo SaaS, o Argos se responsabilizou por toda a estrutura necessária para a disponibilização do sistema (servidor, conectividade, cuidados com segurança) e o cliente preocupou-se somente com a integração e utilização do sistema via *internet*. Após realizar o *deploy*[21] da aplicação no ambiente *web* montado, foi desenvolvido um *Middleware*[22] responsável por receber eventos de *logs* de uma aplicação ERP[23] (*Enterprise Resource Planning*) e transmiti-los para o Argos. A integração foi feita pelo *WebService* (disponibilizado pelo Argos) com o padrão de troca de mensagens REST (*Representational State Transfer*). O *Middleware* realizava a autenticação e enviava *logs* de forma criptografada. O usuário gestor de TI teve acesso a interface *web* do Argos para consultar e analisar os *logs* gerados pela aplicação auditada.

[21] Instalação de uma aplicação em um servidor de aplicações.
[22] Programa de computador que faz uma intermediação em um *software* e demais aplicações.
[23] Sistema Integrado de Gestão Empresarial.

4 CONCLUSÃO

Esta monografia de especialização tratou do desenvolvimento de um *software* para a gestão centralizada e especializada de *logs* de aplicações. Para fundamentar e compreender a importância do desenvolvimento deste sistema foi elaborada uma revisão bibliográfica abordando assuntos relacionados à evolução e importância da informação, sistemas de informação, classificação, ciclo de vida e segurança da informação, administração estratégica, políticas e planos de segurança, normas, desenvolvimento seguro de *software*, registros de eventos (*logs*), além da apresentação de alguns trabalhos relacionados, possibilitando assim, efetuar um comparativo com a solução desenvolvida. Em seguida evidenciou-se todo o ciclo de desenvolvimento da aplicação, para posteriormente, passar aos testes funcionais que verificaram possíveis falhas na estrutura interna do *software*, além de oferecer uma garantia de que os requisitos propostos pelo sistema foram plenamente atendidos.

O *software* desenvolvido nesta monografia de especialização foi denominado de Argos, é ambientado na *web* e possui como objetivo oferecer funcionalidades que visam proporcionar segurança da informação com a detecção de atividades não autorizadas ou em não conformidade com as políticas de segurança da informação, armazenando os eventos de *logs* da aplicação auditada em um único repositório. Este *software* se caracteriza pelo desenvolvimento por meio de tecnologias livres e, em virtude disso, é disponibilizado na modalidade *Open Source*.

Ao adotar uma metodologia de desenvolvimento por meio de tecnologias livres, foi possível reduzir custos, pois não houve a necessidade de se adquirir licenças de *softwares*, além de possibilitar uma independência de fornecedores. Ao desenvolver o Argos seguindo a modalidade *Open Source*, é possível disseminar conhecimento, possibilitando que demais pessoas possam se beneficiar com o seu uso. Para que isso fosse possível, todo o controle de versionamento dos arquivos da aplicação é feito por meio do *Github*[24], onde foi criado um repositório público denominado de Argos (https://github.com/ezequieljuliano/argos).

Para possibilitar que registros de *logs* sejam armazenados, indexados e monitorados em um repositório central, o Argos oferece mecanismos de integração, em que por meio de um *Webservice*, recebe eventos de *logs* com um padrão pré-definido de outras aplicações. Quando estes *logs* chegam, são tratados, indexados e armazenados em um servidor central

[24] https://github.com/ - serviço de *Web Hosting* compartilhado para projetos que usam o método *Git* de versionamento.

(banco de dados e estrutura de diretórios), que disponibiliza uma interface de consulta para filtrar, analisar e utilizar os registros de *logs* na identificação ou na prevenção de problemas em sistemas de informação.

Para oferecer proteção dos registros de *logs* contra falsificação e acesso não autorizado, o Argos disponibiliza mecanismos de segurança via autenticação e autorização. A autenticação garante a identidade do usuário via *login* e senha no caso de acesso a interface de consulta, ou via API *Key* (código único que identifica o usuário) no caso de envio de eventos de *logs* por meio do *Webservice*. Já a autorização se encarrega de verificar se o usuário pode acessar ou executar determinada operação dentro do sistema.

Com o objetivo de garantir a proteção de privacidade de eventos de *logs* com dados pessoais ou confidenciais, o Argos implementa o HTTP com uma camada adicional de segurança via protocolo SSL/TLS, ou seja, o HTTPS (*HyperText Transfer Protocol Secure*). Desta forma, é possível garantir a autenticidade do servidor e trafegar os dados dos *logs* de forma criptografada até o *Webservice*, sem que terceiros possam interceptar estas informações.

Para possibilitar futuras investigações, inclusive análise forense computacional, o Argos é capaz de armazenar registros de *logs* contendo atividades dos usuários, exceções do sistema ou qualquer outro evento de segurança da informação por um longo período de tempo. Isto é possível pelo fato do Argos ser um *software* escalável, de alta performance, capaz de processar grande quantidade dados e possuir em sua essência uma vasta gama de tecnologias especialistas que fornecem mecanismos para suprir grandes demandas. Tecnologias como a linguagem de programação *Java*, o *Demoiselle Framework*, o banco de dados *NoSQL MongoDB* e a biblioteca de indexação e procura de texto *Apache Lucene* agregaram valor a aplicação ampliando seus recursos e tornando-a mais robusta.

Possuir os dados de registros *logs* é importante, mas extrair informações pontuais destes dados é um processo fundamental e o Argos possui grande atenção neste fato. Para possibilitar a análise de registro de *logs*, facilitando a compreensão de problemas encontrados, a maneira pela qual ele pode ocorrer e qual a melhor ação a se adotar, o Argos é dotado de uma interface cheia de recursos. O *software* disponibiliza um *dashboard* com gráficos que expressam visualmente informações de *logs* facilitando a compreensão dos mesmos. Além disso, sua consulta por texto é altamente escalável e possibilita montar diversas expressões e junções de termos garantindo uma minuciosa análise de *logs*, esta possui como principal saída, as informações necessárias para resolver ou prevenir problemas em sistemas computacionais.

De maneira geral, é possível concluir que o *software* desenvolvido neste trabalho de conclusão atende aos requisitos propostos, e pode ser considerado como uma ótima solução para o gerenciamento centralizado de registros de *logs*, além de ser um grande aliado do gestor de TI para a análise e utilização de informações de eventos de *logs* para solução ou prevenção de problemas em sistemas informatizados. Em virtude do rápido avanço das tecnologias, é importante estar atento as mudanças e manter o sistema em constante evolução. Desta forma, como possível trabalho futuro é possível citar pesquisas e implementações de algoritmos capazes de apreender padrões de interesse ou criar regras de correlação de eventos de *logs*.

REFERÊNCIAS

ABREU, Dimitri. **Melhores Práticas para Classificar as Informações**. Módulo e-Security Magazine. São Paulo, agosto 2001. Disponível em <http://www.modulo.com.br>. Acesso em: 12 fev. 2013.

ACADEMIA LATINO AMERICANA DE SEGURANÇA DA INFORMAÇÃO. **Segurança da Informação**. Disponível em: <https://www.microsoft.com/brasil/technet/>. Acesso em: 12 fev. 2013.

ADACHI, Tomi. **Gestão de Segurança em Internet Banking** – São Paulo: FGV, 2004. 121p. Mestrado. Fundação Getúlio Vargas – Administração. Orientador: Eduardo Henrique Diniz.

ALBUQUERQUE, Ricardo e RIBEIRO, Bruno. **Segurança no Desenvolvimento de Software** – Como desenvolver sistemas seguros e avaliar a segurança de aplicações desenvolvidas com base na ISO 15.408. Editora Campus. Rio de Janeiro, 2002.

ALVES, Marcelo Pitanga. et al. **Middlog**: Uma infraestrutura de serviços de log de aplicações baseada em tecnologias de middleware. 2007. Disponível em: <http://sbrc2007.ufpa.br/anais/2005/ST%2010-04%207301.pdf>. Acesso em: 29 set. 2012.

APACHE LOG4J. **Apache Log4j 2**. Disponível em: <http://logging.apache.org/log4j/2.x/>. Acesso em: 16 fev. 2013.

BEAL, Adriana. **Segurança da Informação:** princípios e melhores práticas para a proteção dos ativos de informação nas organizações – São Paulo: Atlas, 2005.

BIANCHI, Wagner. **Como realizar o monitoramento de TI com a ferramenta Splunk**. 2012. Disponível em <http://imasters.com.br/banco-de-dados/monitorando-a-ti-com-splunk/> Acesso em: 22 jul. 2013.

BORAN, Sean. **IT Security Cookbook**, 1996. Disponível em <http://www.boran.com/security/>. Acesso em: 12 fev. 2013.

BORGES, M.. **A informação como recurso gerencial das organizações na sociedade do conhecimento**. Ciência da Informação, Brasília, DF, Brasil, 24, ago. 1995. Disponível em: http://revista.ibict.br/ciinf/index.php/ciinf/article/view/551/500. Acesso em: 07 Fev. 2013.

BRASIL. Serviço Federal de Processamento de Dados (Serpro). **Segurança no Desenvolvimento**. 2013a. Disponível em: <http://www.softwarelivre.serpro.gov.br/recife/download-plaestras/Apresentacao%20Seguranca%20Desenvolvimento.pdf>. Acesso em: 12 fev. 2013.

BRASIL. Núcleo de Informação e Coordenação do Ponto Br. **Cartilha de Segurança para Internet**. 2013b. Disponível em: <http://cartilha.cert.br/mecanismos/>. Acesso em: 12 fev. 2013.

BRASIL. Núcleo de Informação e Coordenação do Ponto Br. **Análise e Interpretação de Logs**. 2003a. Disponível em: <http://www.cert.br/docs/palestras/nbso-gter15-tutorial2003.pdf>. Acesso em: 12 fev. 2013.

BRASIL. Núcleo de Informação e Coordenação do Ponto Br. **Práticas de Segurança para Administradores de Redes Internet**. 2003b. Disponível em: < http://www.cert.br/docs/seg-adm-redes/seg-adm-redes.html#subsec4.4>. Acesso em: 12 fev. 2013.

BRASIL. Tribunal de Contas da União (TCU). S. **Boas Práticas em Segurança da Informação**. 2008. Disponível em: <http://portal2.tcu.gov.br/portal/pls/portal/docs/2059160.PDF>. Acesso em: 12 fev. 2013.

BRASIL. Ministério da Fazenda. Serviço Federal de Processamento de Dados. **Demoiselle Framework**. 2009. Disponível em: <http://www.frameworkdemoiselle.gov.br/>. Acesso em: 25 fev. 2013.

BRAUDE, Eric. **Projeto de Software:** Da programação à arquitetura: uma abordagem baseada em Java. Porto Alegre: Bookman, 2005. 619 p.

BSI BRASIL. **ISO/IEC 27001 Segurança da Informação**. Disponível em: <http://www.bsibrasil.com.br/certificacao/sistemas_gestao/normas/iso_iec27001/>. Acesso em: 12 fev. 2013.

CABRAL, Bill Carlos. **Rsyslog** - Gerenciamento Centralizado de Logs. 2012. Disponível em: <http://www.vivaolinux.com.br/artigo/Rsyslog-Gerenciamento-centralizado-de-logs>. Acesso em: 12 fev. 2013.

CADERMAN, Alexandre. **Segurança em Tecnologia das Informações**: Log´s - Como interpretá-los?. Disponível em: <http://www.viaseg.com.br/artigos/seguranca_logs_alexandre.htm>. Acesso em: 12 fev. 2013.

CANSIAN, Adriano Mauro. **Conceitos para perícia forense computacional**. Anais VI Escola Regional de Informática da SBC, Instituto de Ciências Matemáticas e Computação de São Carlos, USP (ICMC/USP), São Carlos, SP, 30 de abril a 02 de maio de 2001. ISBN 85-87837-05-2, p.141-156, 2001.

CARNEIRO, Alberto. **Auditoria de Sistemas de Informação**. 2004. 2ª edição. Editora FCA, Lisboa – Portugal.

CARUSO, Carlos A. A.; STEFFEN, Flávio Deny. **Segurança em Informática e de Informações** – São Paulo: Editora SENAC São Paulo, 1999.

CAVALCANTI, E. P. (1995) **Revolução da informação**: algumas reflexões. Caderno de Pesquisa em Administração, v. 1, n. 1, p. 40-46, 2º sem.

CLEMENTE, Ricardo Gomes. **Uma Arquitetura para Processamento de Eventos de Log em Tempo Real**. 2008. 72 f. Tese (Mestrado) - Curso de Informática, Puc-rio - Pontifícia Universidade Católica Do Rio De Janeiro, Rio de Janeiro, RJ, 2008.

DALLMANN, Marciano D. **Protótipo de Uma Ferramenta para Análise do Log de Eventos de Um Firewall**. Universidade Regional de Blumenau – FURB. Centro de Ciências Exatas e Naturais. Curso de Pós-graduação em Tecnologias para o Desenvolvimento de Aplicações Web, 2005.

DESTRO, Daniel. **Implementando Design Patterns com Java.** 2004. Disponível em: <http://www.guj.com.br/article.show.logic?id=137>. Acesso em: 05 mai. 2013.

DIAS, Cláudia. **Segurança e Auditoria da Tecnologia da Informação**. Axcel Books. Rio de Janeiro, 2000.

FACEBOOK SCRIBE. **Scribe**. Disponível em: <https://github.com/facebook/scribe/wiki>. Acesso em: 16 fev. 2013.

FIGUEIREDO, Fernando. **XML E Web Service**. 2008. Disponível em: http://www.ticua.net/Apontamentos/TRABALHO1_FINAL_ASES_E_REIS.pdf. Acesso em: 30 mai. 2013.

FONSECA, Paula Fernanda. **Gestão de Segurança da Informação**: O Fator Humano. 2009. Disponível em: <http://www.ppgia.pucpr.br/~jamhour/RSS/TCCRSS08A/Paula%20Fernanda%20Fonseca%20-%20Artigo.pdf>. Acesso em: 12 fev. 2013.

FONSECA, Fernando Sérgio Santos. **Sistema de Tratamento de Arquivos de Logs**. 2005. Disponível em: <http://segurancaobjetiva.files.wordpress.com/2010/10/mono-fernandofonseca.pdf>. Acesso em: 12 fev. 2013.

FONTES, Edison. **Segurança da Informação**: O usuário faz a diferença. São Paulo: Saraiva, 2006. 172 p.

GIL, Antônio de Loureiro. **Sistema de Informações Contábil/Financeiros**. 3. ed. São Paulo: Atlas, 1999.

GONÇALVES, Edson. **Desenvolvendo Aplicações Web com JSP, Servlets, Javaserver Faces, Hibernate, EJB 3 Persistence e Ajax.** Rio de Janeiro: Editora Ciência Moderna, 2007. 736 p.

HOLANDA, Roosevelt de. **O estado da arte em sistemas de gestão da segurança da Informação**: Norma ISO/IEC 27001:2005 – São Paulo: Módulo Security Magazine, 19 jan 2006. Disponível em <http://www.modulo.com.br>. Acesso em: 12 fev. 2013.

INDRUSIAK, Leandro Soares. Linguagem Java. **Grupo JavaRS JUG Rio Grande do Sul**, 1996.

ISO 17799. ABNT NBR ISO/IEC 17799:2003 – Tecnologia da Informação. **Código de Prática para Gestão da Segurança da Informação**. Associação Brasileira de Normas Técnicas. Rio de Janeiro, 2003.

ISO 17799. ABNT NBR ISO/IEC 17799:2005 – Tecnologia da Informação – Técnicas de segurança. **Código de prática para a gestão da segurança da informação**. Associação Brasileira de Normas Técnicas – Rio de Janeiro: ABNT, 2005.

JLOG. **JLog**. Disponível em: <http://jlog.org>. Acesso em: 16 fev. 2013.

JOHNSON, Rod. **Expert one-on-one:** J2EE desing and development. Indianápolis - USA: Wrox, 2003. 742p.

KRAUSE, Micki e TIPTON, Harold F. **Handbook of Information Security Management**. Auerbach Publications, 1999.

LAUDON, Kenneth C.; LAUDON, Jane Price. **Sistemas de informação**. 4. ed. LTC: Rio de Janeiro,1999.

LAUREANO, Marcos Aurélio Pchek; MORAES, Paulo Eduardo Sobreira. **Segurança como estratégia de gestão da informação**. In: Revista Economia & Tecnologia, v. 8, fasc. 3, p. 38-44, 2005. Disponível em: <http://www.ppgia.pucpr.br/~euclidesfjr/SEGURANCA_DA_INFORMACAO/economia_tecnologia_seguranca_2005.pdf>. Acesso em: 12 fev. 2013.

LE COADIC, Yves-François. **A ciência da informação**. Brasília: Briquet de Lemos Livros, 1996.

LENNON, Joe. **Explore MongoDB:** Learn why this database management system is so popular. Disponível em: <http://www.ibm.com/developerworks/library/os-mongodb4/>. Acesso em: 13 maio 2013.

LESCA, H.; ALMEIDA, F. C. de. **Administração estratégica da informação**. Revista de Administração de Empresas, v.29, n.3, p.66-75, jul./set. 1994.

LIMA, Adilson da Silva. **UML 2.0**: Do Requisito a Solução. 2. ed. São Paulo: Editora Érica, 2007. 326 p.

LISBOA, Flávio Gomes da Silva. Made in Brazil: Padronização e Reuso de Aplicações Web com Demoiselle Framework. **Revista Mundo Java**: jruby da Web ao Desktop, Curitiba, n. 36, p.8-17, 01 jul. 2009a. Bimestral.

LISBOA, Flávio Gomes da Silva. **Demoiselle Framework. 2009b.** Disponível em: <http://www.frameworkdemoiselle.gov.br/menu/framework/apresentacoes/apresentacao-conip-2009>. Acesso em: 05 mai. 2013.

LOGKIT. **LogKit**. Disponível em: <http://mvnrepository.com/artifact/logkit/LogKit/1.2>. Acesso em: 16 fev. 2013.

LOGZILLA. **LogZilla**. Disponível em: <http://www.logzilla.pro/>. Acesso em: 16 fev. 2013.

LYRA, Maurício Rocha. **Segurança e auditoria em sistemas de informação**. Rio de Janeiro: Ciência Moderna, 2008. 253 p.

LUCENE. **Apache Lucene Core**. 2013. Disponível em: <http://lucene.apache.org/core/>. Acesso em: 25 fev. 2013.

MACIAS, Ananda De Medeiros. **Framework de desenvolvimento:** visão geral. 2008. Disponível em: <http://www.serpro.gov.br/clientes/serpro/serpro/imprensa/publicacoes/tematec/2008/ttec92_a>. Acesso em: 05 mai. 2013.

MONGODB. **MongoDB Agile and Scalable**. 2013. Disponível em: <http://www.mongodb.org/ >. Acesso em: 25 fev. 2013.

OLIVA, Rodrigo Polydoro; OLIVEIRA, Mírian. **Elaboração, Implantação e Manutenção de Política de Segurança por Empresas no Rio Grande do Sul em relação às recomendações da NBR/ISO17799 – ENANPAD**. 2003.

OLIVEIRA, Leonardo Eloy. **Estado da arte de banco de dados orientados a documento**. 2009. Monografia (Conclusão do Curso de Graduação em Ciências Tecnológicas) – Universidade de Fortaleza – UNIFOR, Ceará, 2009.

PARRA, Gislaine. **Metodologia para análise de segurança aplicada em uma infraestrutura de chave pública**. 2002. 99f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina, Florianópolis.

PELEGRI-LLOPART, Eduardo; Yoshida, Yutaka; Moussine-Pouchkine, Aléxis. **The GlassFish Community**: Delivering a Java EE Application Server. 2007. Disponível em: http://glassfish.java.net/faq/v2/GlassFishOverview.pdf. Acessado em 24 de maio de 2013.

PENDER, Tom. **UML a Bíblia**. Rio de Janeiro: Editora Campus, 2004. 711 p.

PEREIRA, Maria José Lara de Bretãs; FONSECA, João Gabriel Marques. **Faces da Decisão**: as mudanças de paradigmas e o poder da decisão. São Paulo: Makron Books, 1997.

REGO, Bruno M.; BROSSO, Inês**. Segurança no Desenvolvimento de Sistemas com Metodologia Ágil SCRUM**. Disponível em: <http://www.slideshare.net/BrunoMottaRego/segurana-no-desenvolvimento-de-sistemas-com-metodologia-gil-scrum>. Acesso em: 12 fev. 2013.

RINALDI. **Porque usar MVC.** 2009. Disponível em: <http://grifemidia.com.br/noticias-grife/2009/02/17/porque-usar-mvc/>. Acesso em: 05 mai. 2013.

SACCONI, Luiz Antonio. 1998. **Dicionário da Língua Portuguesa**, São Paulo: Atual.

SANDHU, Ravi S. e SAMARATI, Pierangela. **Authentication, Acess Control, and Intrusion Detection**. IEEE Communications, 1994.

SÊMOLA, Marcos. **Gestão da Segurança da Informação**: uma visão executiva – Rio de Janeiro: Campus, 2003.

SHIREY, R. **RFC 2828 – Internet Security Glossary**. The Internet Society, 2000. Disponível em: <http://www.ietf.org/rfc/rfc2828.txt?number=2828>. Acesso em: 12 fev. 2013.

SILVA, Pedro Tavares; CARVALHO, Hugo; TORRES, Catarina Botelho. **Segurança dos Sistemas de Informação**: Gestão Estratégica da Segurança Empresarial. Lisboa, Portugal: Centro Atlântico, 2003. 254 p.

SILVEIRA, Paulo et al. **Introdução à Arquitetura e Design de Software:** Uma Visão Sobre a Plataforma Java. Rio de Janeiro: Elsevier, 2012. 257 p.

SPLUNK. **Splunk**. Disponível em: <http://www.splunk.com/>. Acesso em: 16 fev. 2013.

SPRINGSOURCE, S. **Spring Data.** Disponível em: <http://www.springsource.org/spring-data/>. Acesso em: 24 maio 2013.

STAIR, Ralph M. **Princípios de sistemas de informação**. Rio de Janeiro: LTC, 1998.

SOLHA , Liliana Esther Velásquez Alegre. Os Logs como Ferramenta de Detecção de Intrusão. 1999. Disponível em: <http://www.rnp.br/newsgen/9905/logs.html>. Acesso em: 16 fev. 2013.

SONAWANE, Amol. **Using Apache Lucene to search text:** Easily build search and index capabilities into your applications. Disponível em: <http://www.ibm.com/developerworks/library/os-apache-lucenesearch/>. Acesso em: 12 maio 2013.

THE ISO 27000 DIRECTORY. **An Introduction to ISO 27001, ISO 27002....ISO 27008**. Disponível em: <http://www.27000.org/index.htm>. Acesso em: 12 fev. 2013.

TRELLE, Tobias. **Spring Data**: A solução mais geral para persistência?. Disponível em: <http://www.infoq.com/br/articles/spring-data-intro>. Acesso em: 24 maio 2013.

WADLOW, Thomas. **Segurança de Redes**. Editora Campus. Rio de Janeiro, 2000.

ZEMEL, Tárcio. **Padrões de projetos (ou design patterns): o que são e para que servem e qual a sua implicação de uso.** 2009a. Disponível em: <http://codeigniterbrasil.com/passos-iniciais/padroes-de-projeto-ou-design-patterns-o-que-sao-para-que-servem-e-qual-sua-implicacao-de-uso/>. Acesso em: 05 mai. 2013.

ZEMEL, Tárcio. **MVC(Model-View-Control).** 2009b. Disponível em: <http://codeigniterbrasil.com/passos-iniciais/mvc-model-view-controller/>. Acesso em: 05 mai. 2013.

ANEXOS

ANEXO A – Executando o sistema Argos

Passo a Passo para Criação do Ambiente de Execução do Argos	
Passo	**Tarefa a Executar**
1	Efetue o *download* do *NetBeans* IDE 7.3 através do endereço https://netbeans.org/downloads/7.3/ e realize sua instalação em seu sistema operacional.
2	Efetue o *download* o *MongoDB* em sua versão 2.4.1 através do endereço http://www.mongodb.org/downloads e realize sua instalação e configuração.
3	Faça o *download* do projeto Argos por meio de seu repositório no *Github* que está disponível no seguinte endereço: https://github.com/ezequieljuliano/argos.
4	Coloque em execução o *MongoDB* e abra o *NetBeans* IDE 7.3.
5	Abra o projeto Argos no *NetBeans* IDE 7.3 utilizando a opção do menu Arquivo > Abrir Projeto.
6	Após o projeto ser carregado clique com o botão direito sobre ele e selecione a opção "Limpar e Construir". Aguarde todas as dependências serem baixadas para sua máquina, este processo é um pouco demorado e necessita de conexão com a internet.
7	Nas configurações do projeto ative ou crie o perfil "*glassfish3*".
8	Agora execute o projeto, ao final da compilação o navegador será aberto e o sistema executado.