Airton Carlos Nunes Raimundo
Luiz Fernando Yamaoka

Agentes Autônomos para Segurança Residencial

Brasil
2013

Airton Carlos Nunes Raimundo
Luiz Fernando Yamaoka

Agentes Autônomos para Segurança Residencial

Trabalho de formatura

Universidade de São Paulo
Escola Politécnica
Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos

Orientador: Prof. Dr. José Reinaldo Silva
Coorientador: Prof. Marco Antonio Poli Jr

Brasil
2013

Nunes Raimundo, Airton Carlos; Yamaoka, Luiz Fernando
Agentes Autônomos para Segurança Residencial/ Airton Carlos Nunes Raimundo e Luiz Fernando Yamaoka. – Brasil, 2013-
131 p. : il. (algumas color.) ; 30 cm.

Trabalho de Formatura – Universidade de São Paulo
Escola Politécnica
Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos, 2013.

1. Edifícios residenciais (Segurança). 2. Sistemas autônomos. 3. Arquitetura e organização de computadores. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos

Resumo

A evolução dos sistemas automatizados aponta hoje para uma tendência a utilizar sistemas mais pervasivos e distribuídos, eventualmente portadores de algum nível de inteligência, conferindo aos respectivos sistemas uma abordagem cognitiva. Os sistemas domóticos em especial partilham desta tendência, mas estão ainda muito atrelados, no que tange ao desenvolvimento de sistemas mecatrônicos, a uma dependência dos sistemas industriais correspondentes. Um dos problemas acarretado por isso é a falta de flexibilidade, especialmente dos sistemas de vigilância. Neste caso o ideal seria ter sistemas distribuídos e autônomos que pudessem ser colocados em qualquer ponto do ambiente, sem a necessidade de instalação sofisticada causando mudanças - as vezes drásticas - neste ambiente legado. Naturalmente estes elementos distribuídos poderiam ser mapeados compondo um sistema integrado regido por um servidor. Neste trabalho desenvolveremos um projeto de sistema mecatrônico que na verdade é um sistema composto de agentes autônomos independentes funcionando cooperativamente e coordenados por um sistema central que implementaria as funções coordenadas de vigilância (como "seguir"um alvo móvel). A arquitetura distribuída é expansível e pode ser futuramente utilizada, inclusive em um sistema multi-agente, e será baseada em elementos autônomos que usam o Robot Operating System (ROS) como base de processamento, faremos ainda as tomadas do ambiente alvo com diferentes níveis de resolução, dependendo do estado do ambiente. Isto permite uma maior agilidade e a concentração do sistema supervisório somente em dados que realmente interessam para o processo de vigilância, em um determinado intervalo de tempo. Neste documento mostraremos a fase inicial do desenvolvimento, seguindo o método de prototipagem rápida e virtual (e não um processo de eliciação e análise de requisitos convencional), de modo a se adaptar ao atraso ocorrido no início do projeto. Para o futuro (segundo semestre) ficarão o desenvolvimento de alguns protótipos do agente independente, seu controle e processamento local e a integração com o software supervisório do servidor. Uma prova de conceito será desenvolvida baseada na implementação de um número reduzido de agentes.

Palavras-chaves: Sistema de segurança, agentes independentes, sistema distribuído, multi-agente

Abstract

Nowadays the evolution of automated systems points to a tendency to use more pervasive and distributed systems, possibly containing some level of intelligence, giving the respective systems a cognitive approach. Home automation systems, in particular, share this trend, but are still very linked to the dependence of the corresponding industrial systems, regarding the development of mechatronic systems. One of the problems entailed by this is the lack of flexibility, especially surveillance systems. In this case the ideal would be to have autonomous distributed systems, that could be placed anywhere in the room, without the need of sophisticated installation processes causing change in the environmente, sometimes dramatic. Naturally these distributed elements could be mapped composing an integrated system governed by a server. In this paper we develop a project of mechatronic system which is actually a system composed of autonomous agents working independently, but cooperatively and coordinated by a central system that would implement the coordinated functions of surveillance (f.i. "follow" a moving target). The distributed architecture is scalable and can be used in the future, even in a multi-agent system, and will be based on autonomous elements that use the Robot Operating System (ROS) as a processing base, the pictures of the environmente can be taken with different levels of resolution depending on the state of the environment. This benefits the system with a greater agility and the supervisory system can be concentrated on data that really matter to the vigilance procedure in a given time interval. This document show the initial phase of development, following the method of rapid prototyping and virtual (and not a process of elicitation and conventional analysis of requirements), in order to adapt to delay in the start of the project. In the future (second half of the year) some prototypes of the independent agent will be developted, its control and local processing and integration with supervisory software server. A proof of concept will be developed based on the implementation of a small number of agents.

Keywords: Security system, independent agents, distributed system, multi-agent

Lista de ilustrações

Lista de abreviaturas e siglas

AVAC	Aquecimento, Ventilação e Ar Condicionado
OpenCV	Open Source Computer Vision Library
ROS	Robot Operating System
VPN	Virtual Private Network

Sumário

1 Introdução

Os sistemas de segurança para domótica geralmente apresentam uma arquitetura fixa, embora distribuída, onde sensores e câmeras são colocados em postos fixos. Isso depõe contra o próprio requisito de segurança, dado que é possível ter conhecimento prévio da colocação dos sensores e câmeras, na maior parte dos casos.

Outro problema é que os sistema de domótica atualmente são isolados e não se intercomunicam. Por exemplo o sistema de segurança observado em (VERISURE, 2013) não permite a integração com outros sistemas presentes na residência. Essa característica dos sistemas de segurança domótica atuais restringe que novas funcionalidades possam ser implementadas e sistemas mais completos possam ser criados. Além disso, nos sistemas de segurança atuais também não há a possibilidade de expansão em escala como observado em (VERISURE, 2013) há uma limitação de 32 detectores, em (SMSOLUTION, 2013) existe uma limitação de 16 câmeras e em (SU; LEE; WU, 2006), onde foi desenvolvido um sistema de home automation a partir de um microcontrolador, que possui apenas 6 entradas para receber sinais dos sensores e 4 saídas, que controlam os aparelhos. Se houver necessidade de expansão do sistema, esta é geralmente feita a custa de retrabalho e adaptações significativas no sistema original – embora já existam algumas propostas de sistema já modularizados (GOMASA, 2011).

Outra característica é que os diversos sistemas de segurança do mercado podem utilizar diversos sensores, como sensores de contato, de presença, infravermelho, de movimento, câmeras e também atuadores como luzes, alto-falantes e até robôs (HSU; YANG; WU, 2009), (SONG et al., 2009), (LUO, 2007). No entanto, um fato em comum deles é que possuem uma central de monitoramento e/ou central de controle. Com isso, se esta central possui algum problema o sistema inteiro perde sua funcionalidade e, portanto, o ambiente se torna inseguro. Por isso, objetiva-se neste projeto desenvolver um sistema de agentes distribuídos, ou seja, um sistema de agentes que funcionem independentemente. Desta forma, não haveria uma central de controle e se um agente é, por exemplo, danificado, os outros agentes continuam operando sem problemas.

Locais públicos possuem ainda requisitos próprios quando se trata de sistemas de segurança porque, diferente de um ambiente privado, um ladrão tem o total acesso ao ambiente do furto quantas vezes ele desejar. Isso permite que planos complexos sejam arquitetados para vencer um sistema de segurança convencional. Partindo dessa particularidade, esse trabalho propõe o desenvolvimento um sistema de segurança flexível contra intrusão que permita a realocação de seus sensores a qualquer momento. Outro exemplo, no qual este problema é abordado seria um condomínio de casas, onde o condomínio ofe-

reça um sistema de segurança interno para as casas. No entanto, cada casa possui suas particularidades. Deste modo, algumas casas poderiam optar pela instalação das câmeras enquanto outros não por questões de privacidade; a disposição dos móveis e a construção das casas são diferentes em cada casa, então as câmeras precisam ser dispostas em locais diferentes; após alguns meses de teste, o usuário pode não gostar do sistema e optar por retirá-lo de sua residência; entre outros exemplos. Em todos estes casos, um sistema flexível seria muito útil e facilitaria a vida dos usuários.

Neste trabalho investigaremos a possibilidade de implementação de um sistema de segurança com uma arquitetura multi-agente flexível, que de fato pode ser usada em vários sistemas de automação, seja em domótica ou em outros ramos de aplicação. Como demonstrado em (ABREU et al., 2000) em uma aplicação para monitoramento de tráfego de veículos em uma rodovia.

O sistema proposto pode ser caracterizado como uma arquitetura de duas camadas para um sistema multi-agente (não-inteligente, no caso de estudo proposto), colaborativo, flexível e aberta – onde novos elementos podem entrar a qualquer momento, bastando ter o seu identificador registrado no banco de dados, juntamente com a sua posição atual. Isso permite que a qualquer momento a posição dos elementos sensor/camera possam ser alterados e atualizados novamente no banco de dados sem nenhuma alteração na funcionalidade ou na programação dos processos do sistema.

Para garantir a generalidade do sistema e sua possível aplicação a outros aplicativos de domótica ou de automação em geral, optou-se por ter dispositivos genéricos, que seguem a atual tendência de modularização: cada agente será então implementado em um Raspberry Pi e conectados sem fio a um servidor (onde fica a camada de processamento principal, onde eventualmente se incluiria inteligência); em cada agente tem seu processamento local baseado em ROS (Robotic Operating System).

Apesar de ter como base uma implementação específica para sistemas domóticos o objetivo principal é na verdade discutir e investigar a arquitetura proposta e sua aplicabilidade a outros sistemas similares.

2 Estado da Arte

Como dito anteriormente já existem no mercado vários sistemas de domótica, que, no entanto, possuem diversas limitações referentes a intercomunicação com sistemas com funções diferentes (VERISURE, 2013) e também a impossibilidade de expansão em escala no número de sensores, câmeras, detectores e outros (SMSOLUTION, 2013).

Para solucionar esses problemas supracitados uma possibilidade seria a implantação de sistemas multi-agentes como mostrado em (ABREU et al., 2000) no desenvolvimento de um sistema multi-agente para monitorar o tráfego de veículos em uma rodovia. Este trabalho foi realizado por um consórcio dos laboratórios ADDETTI (Lisboa, Portugal), EPFL (Lausane, Suiça), LEP (Paris, França) e UCL (Louvain, Bélgica) e mostrou a possibilidade de expansão em escopo, possibilidade de adicionar novas funcionalidades ao sistema; e também a possibilidade de expansão em escala com a adição de mais agentes ao sistema. Contudo, apesar deste sistema implementar um sistema com uma arquitetura multi-agente como também é objetivado aqui, o projeto em questão possui requisitos diferentes dos nossos, principalmente requisitos referentes ao sistema de segurança.

No âmbito da segurança residencial, um grande problema existente é a central de comando, que é o ponto fraco do sistema, pois uma vez desativada a central de comando, o sistema inteiro perde a sua funcionalidade. Por isso, uma solução para este problema é a implementação de sistemas multi-agentes com agentes distribuídos como demonstrado em (SHARPLES; CALLAGHAN; CLARKE, 1999). Neste sistema os agentes possuem sua própria inteligência e, portanto, tem autonomia para funcionar sozinho, mas também podem realizar tarefas em conjunto.

Outros trabalhos foram realizados na área do monitoramento de tráfego e também mostraram sistemas multi-agentes com agentes possuindo inteligência artificial (VALLEJO et al., 2011). Neste trabalho cada agente é um computador normal e, por isso, é mostrado que cada gente possui uma capacidade de processamento muito alta. No entanto, o sistema perde em flexibilidade, ou seja, na possibilidade de fácil realocação dos agentes e posicionamento em locais sem fontes de energia, como será deverá ser plausível para um sistema de segurança.

3 Requisitos de projeto

Este projeto não foi realizado através metodologia convencional, onde os requisitos funcionais e não funcionais devem primeiro ser definidos e posteriormente da análise destes. Ao invés disso, este projeto foi desenvolvido com base na aplicação da metodologia de prototipagem rápida e virtual (seção 3.1.1); e de um projeto exploratório (seção 3.1.2). Em termos metodológicos e de projeto isto significa que o projeto já partiu de um modelo de sistema de vigilância baseado em agentes canônicos (seção 3.2). Esta estratégia permitiu trabalhar diretamente na prova de conceito do modelo proposto estudando a viabilidade da arquitetura.

3.1 Metodologia de projeto

3.1.1 Prototipagem rápida e virtual

Prototipagem rápida e virtual é uma metodologia ágil de projeto que possui foco em atingir rapidamente o produto final para submetê-lo a testes. Essa metodologia é empregada em projetos exploratórios e que podem ter alguma restrição tecnológica para serem realizados.

Por esse motivo e diferentemente do projeto de metodologia convencional, não tem um foco muito grande na definição dos requisitos de projeto. Por isso, é necessária a definição de um modelo de referência (seção 3.2), que servirá como base para o desenvolvimento do projeto. Nele serão definidos detalhes de como o projeto deverá ser apesar de não haver um estudo de requisitos.

3.1.2 Projeto exploratório

Um projeto exploratório visa a pesquisa e desenvolvimento de um conceito ou uma ideia que é possível na teoria mas que pode possuir uma barreira tecnológica para ser realizado. Com a implementação desta ideia, será testada se esta é realmente plausível de ser aplicada na prática.

Por isso, é possível que a tecnologia que dispomos hoje não seja suficiente para criarmos um sistema de segurança expansível, multi-agente e flexível, no entanto, novas aplicações para a tecnologia testada podem surgir desse trabalho.

3.2 Modelo de referência

O modelo de referência é uma solução abstrata para uma classe de problemas. Assim esta solução omite os detalhes de implementação de uma instância particular do problema para se concentrar nos aspectos gerais.

O modelo de referência aqui definido tem como base as ideologias do Prof. Dr. José Reinaldo Silva e do Prof. Marco Poli. Ele é definido por uma arquitetura multi-agente onde a distribuição de tarefas é baseada na arquitetura e em uma definição funcional do papel destes agentes. Assim, o modelo explorado neste trabalho não é baseado em agentes inteligentes, e portanto estes agentes não usam a troca de mensagem para definir o que fazer. Em caráter especial uma situação de emergência foi considerada onde a coordenação local de atividades pode ser repassada a outro agente, quando o coordenador estiver impossibilitado ou estiver sendo atacado. Mesmo nesse caso a decisão é unilateral do agente coordenador.

No caso geral, um sistema multi-agente clássico é um sistema formado com base em uma arquitetura de agentes autônomos, que possuem certa inteligência (SHARPLES; CALLAGHAN; CLARKE, 1999), (VALLEJO et al., 2011) para execução de tarefas visando objetivos específicos. Uma decorrência direta do caráter autônomo é que o não funcionamento de um agente não impede outros agentes de efetuarem suas atividades. Logo, os agentes estariam todos em uma mesma camada de nível. Desta maneira, o sistema seria facilmente expansível, pois não existe uma hierarquia entre os agentes (neste nível).

Além disso, existe o conceito por trás da arquitetura que um agente ativo (KUBERA; MATHIEU; PICAULT, 2010) não precisa ser capaz de realizar toda a inteligência do sistema, mas que um comportamento global inteligente pode ser obtido através da soma das atividade individual e independente de cada um dos agentes.

No entanto, o modelo de referência utilizado define uma arquitetura multi-agente modificada para um sistema em duas camadas, onde a camada superior é um supervisor rodando sobre um sistema de informação e a segunda camada é composta pelos demais agentes, como pode ser observado na Figura 1. Neste sistema o agente superior, que é o sistema de informação não sabe da existência dos agentes do sistema multi-agente, mas os agentes da segunda camada conhecem o endereço do sistema para que possam estabelecer uma comunicação com o servidor, estabelecendo um comunicação “one way”, o que reduz ainda mais a necessidade de comunicação.

Outra ideia do modelo de referência é que o sistema seja aberto, e altamente configurável e modificável, ou seja, os agentes devem ser implementados em um hardware que permita a fácil realocação dos agentes, e o sistema global deve admitir sempre a entrada de novos agentes ou a retirada de algum.

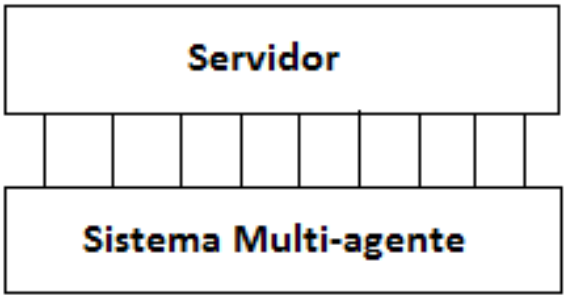


Figura 1 – Diagrama do sistema multi-agente modificado

4 Design do sistema

Para que o sistema desenvolvido atenda ao modelo de referência, algumas especificações foram realizadas para este projeto nos seguintes pontos: Hardware (seção 4.1), software (seção 4.2), aquisição de imagens (seção 4.3) e sistema de segurança (seção 4.4)

4.1 Hardware

Com o intuito de montar um agente para a aquisição das imagens das câmeras e realizar possivelmente um tratamento inicial da imagens, diversos hardwares poderiam ser utilizados, como por exemplo:

- Desktop
- Laptop
- Arduino
- *Raspberry Pi*

Desktops e Laptops apresentam grande poder de processamento e seriam ótimos para a construção de agentes independentes, que fariam todo o processamento de imagens neles mesmo, mas não seriam uma boa opção para este projeto, pois é objetivo deste projeto construir agentes pequenos e de fácil realocação. Conta ainda contra estas opções o alto custo do equipamento.

O Arduino seria uma ótima solução para o desenvolvimento de um agente pequeno, leve e muito flexível, no entanto, seu poder de processamento é muito baixo (o Arduino Uno, por exemplo, conta com apenas 2Kb de memória RAM estática) e seria muito difícil realizar um processamento da imagem local.

Uma opção para suprir as necessidades de realizar um agente flexível com poder de processamento razoável seria o *Raspberry Pi*. Mais detalhes sobre esse hardware são dados na seção seguinte.

4.1.1 *Raspberry Pi*

O Raspberry Pi é um computador de baixo custo desenvolvido pela Raspberry Pi Foundation no Reino Unido com o intuito de promover o ensino de ciência da computação em escolas. O O modelo B deste computador possui as dimensões de 85.60mm x 53.98mm, duas portas USB, adaptador Ethernet para conexão a rede, fonte de energia via MicroUSB,

512Mb de memória RAM e outros. Além disso, pode rodar uma versão adaptada da distribuição para Linux Debian, o Raspbian.

No Brasil, ele pode ser adquirido a um custo de R$150,00 por unidade.

Devido as suas pequenas dimensões pode ser utilizado neste projeto para controlar a câmera, adquirir as imagens e realizar o tratamento inicial destas. Deste modo, poderíamos desenvolver um sistema pequeno, móvel e flexível, que poderá ser movido e reinstalado pelo usuário sem perda de funcionalidades e qualidade.

4.2 Software

Para atender aos requisitos poderíamos desenvolver uma linguagem própria de programação, que possuiria uma adaptabilidade imensa, pois seria desenvolvida por nós mesmos. No entanto, seu desenvolvimento seria muito trabalhoso. Uma segunda alternativa seria a utilização de *frameworks* já existentes, que proporcionariam uma solução mais robusta, pois são desenvolvidas por especialistas há anos. Duas opções de *frameworks* já existentes são *Erlang* e *ROS (Robot Operating System)*.

Erlang é uma linguagem de programação a principio desenvolvida pela empresa Ericsson, mas que agora é disponível *open source*. Esta é uma linguagem muito robusta, com funções altamente desenvolvidas.

Apesar do *Erlang* ser uma boa alternativa para este projeto optaremos pelo *ROS*. *ROS* não possui uma gama de funcionalidades como o *Erlang*, no entanto, possui um fórum altamente ativo e prontamente disposto a ajudar, fator que ajudará no desenvolvimento do projeto.

4.2.1 *ROS*

ROS é uma framework de software para o desenvolvimento de software para robôs, no entanto, possui as ferramentas necessárias para o desenvolvimento deste projeto, como transmissão de mensagens, captura de imagens da webcam e outros, além de possuir ferramentas de robótica, que poderão ser utilizadas futuramente para o desenvolvimento e aperfeiçoamento do projeto.

Esta framework funciona com uma arquitetura, que possui quatro tipos diferentes de agentes:

- Service
- Client
- Publisher

- Subscriber

Estes agentes funcionam pareados. O agente service trabalha em conjunto com o agente client, e o publisher com o subscriber.

Como pode ser observado na figura abaixo, os agentes service e client funcionam como um sistema de pergunta e resposta. O agente client pergunta ao agente service e obtêm a resposta deste. Deste modo, vários clientes podem interagir com o mesmo serviço e um único cliente também pode interagir com múltiplos serviços.

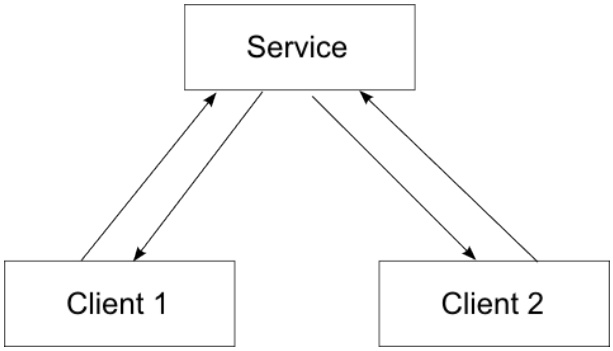


Figura 2 – Sistema client e service

Os agentes publisher e subscriber trabalham diferentemente. O agente publisher envia uma mensagem constantemente: um feed de mensagens, e o subscriber "ouve"este feed de mensagens. Assim sendo, vários subscribers podem aderir ao mesmo feed de um publisher e, por exemplo, realizar tarefas diferentes com este feed. Por exemplo, um publisher envia a imagem da sua câmera em um feed e há um subscriber detectando movimento e outro subscriber realizando o reconhecimento de faces. Este modo de funcionamento pode ser observado na figura abaixo.

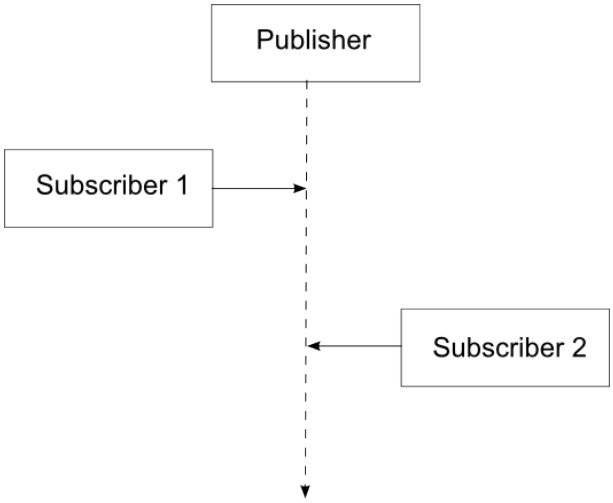


Figura 3 – Sistema publisher e subscriber

Com estes agentes é possível desenvolver o sistema expansível desejado, pois poderemos adicionar novos subscribers para realizar tarefas diferentes, aumentando as funcionalidades do projeto, assim como novas câmeras gerando novos feeds de mensagens, que serão aderidos por outros subscribers aumentando a escala do projeto.

No início do desenvolvimento do ROS, o sistema deveria conter sempre um único nó master para que o sistema pudesse funcionar. Deste modo, não seria possível desenvolver um sistema de agentes distribuídos, pois sem a existência do nó master, o sistema pararia de funcionar. No entanto, foram desenvolvidas recentemente soluções multi-master para sistemas ROS, portanto, este problema pode ser sanado e um sistema distribuído construído.

4.3 Aquisição de imagens

Para definir as 2 variáveis na aquisição das imagens: resolução e cadência, foram realizados alguns testes e comparações. Para estes testes foi utilizada a câmera embutida do laptop Asus U46E. Esta é uma webcam de resolução máxima de 0,3Mp.

4.3.1 Resolução da imagem

Para realizar a seleção da resolução da imagem adquirida, foi realizado um teste analisando a mesma imagem em diversas resoluções.

Para este teste utilizamos uma imagem em tamanho 4x3, pois este é o formato da imagem fornecida por grande parte das webcams atuais. Deste modo analisamos a mesma imagem de um corredor com uma pessoa em primeiro plano nas seguintes resoluções: 240x320, 360x480 e 480x640.

Observando as imagens obtidas, pudemos observar que a imagem com menor resolução não apresentará uma resolução mínima suficiente para processá-la e reconhecer uma intrusão de uma pessoa. Por exemplo, será reconhecer movimento de pessoas fora do primeiro plano, por isso acredita-se que uma imagem de resolução 320x240 não apresente os requisitos necessários para o reconhecimento de intrusões.

Analisando a imagem de resolução 480x360 já apresenta uma resolução suficiente para reconhecer a intrusão de pessoas em pelo menos primeiro e segundo plano. Naturalmente um movimento muito distante será muito difícil de reconhecer, mas realizar este reconhecimento seria necessário aumentar consideravelmente a resolução da imagem e, consequentemente, sobrecarregar a rede com a transmissão desses dados.

No entanto, para realizar processamentos mais complexos do que o reconhecimento de intrusão, como o reconhecimento de faces, seria necessário uma resolução maior como, por exemplo a de 640x480.

Para evitar o sobrecarregamento desnecessário da rede, uma solução viável seria o envio de imagens com resolução 480x360 enquanto não houvesse reconhecimento de intrusões e a partir do momento que um intruso fosse reconhecido, a resolução da imagem adquirida aumentasse para 640x480 ou superior para que os processamentos complexos sejam realizados.

4.3.2 Cadência

Para a filmagem e projeção de filmes em cinema a cadência padrão desde a década de 20 é de 24fps (BROWNLOW, 1980), portanto, esta frequência de quadros oferece uma qualidade visual muito boa e contínua. Para o desenvolvimento de um sistema de segurança as imagens seriam captadas idealmente em uma frequência que oferecesse uma imagem contínua, no entanto, com o intuito de não sobrecarregar a rede com a transmissão de dados, utilizaremos uma cadência menor, mas que não comprometa a segurança do local em questão, ou seja, que não perca informações importantes.

Para a escolha da cadência foram realizados três testes: a 2fps, 4fps e 6fps. Neste teste uma pessoa passou caminhando normalmente a cerca de meio metro de distância da câmera. Com isso pode ser analisado em quantos frames a pessoa seria captada.

O resultado destes testes pode ser observado nas figuras 4, 5 e 6.

Figura 4 – Captação de imagens em cadência de 2fps

Como observado nos testes, a pessoa pode ser observada em 2 frames na cadência de 2fps, em 4 frames na cadência de 4fps e em 7 frames na cadência de 6fps.

Dois pontos podem e devem ser citados aqui. Primeiramente as câmeras de segurança geralmente são fixadas em locais altos e, portanto, dificilmente uma pessoa passaria a uma distância de meio metro da câmera. Por estar mais distante da câmera, em uma sequência de imagens captada com a mesma cadência, a pessoa apareceria em mais frames do que nos testes realizados. O segundo ponto a ser levado em consideração é que o primeiro objetivo a ser atendido é o reconhecimento da presença de uma pessoa no local, portanto, não é necessário que ela seja captada em diversos frames.

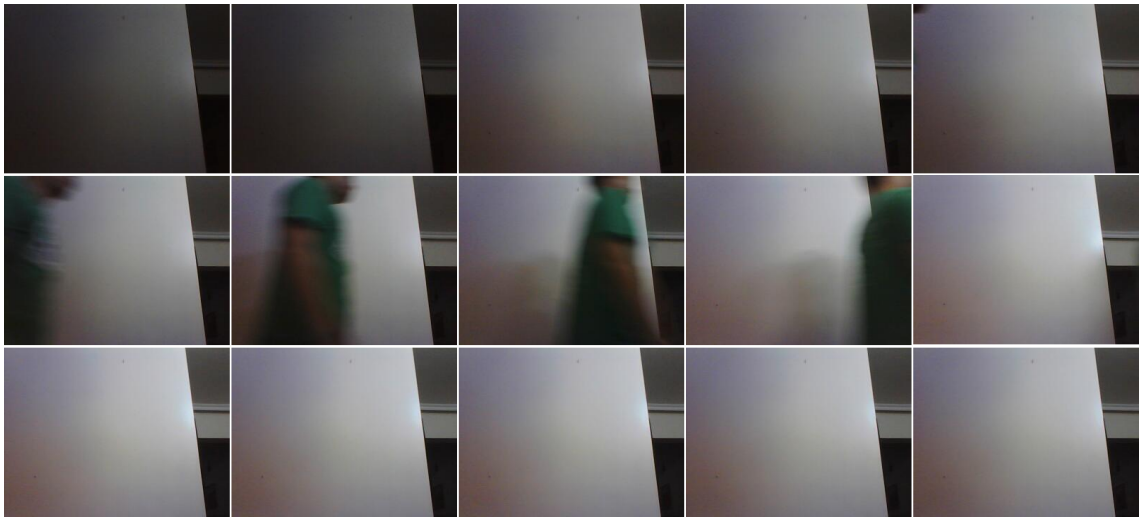


Figura 5 – Captação de imagens em cadência de 4fps

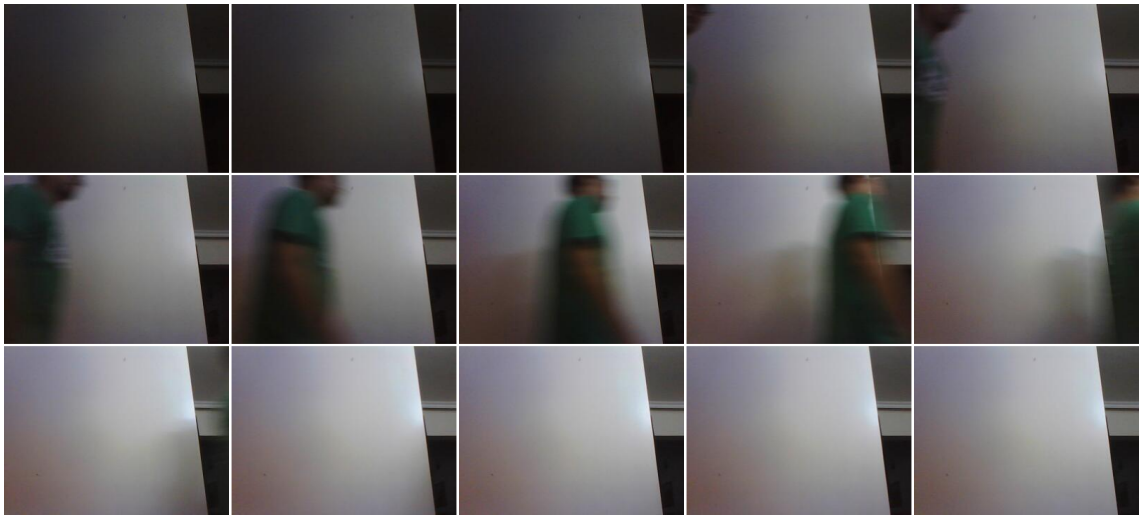


Figura 6 – Captação de imagens em cadência de 6fps

Considerando que um invasor poderia passar pelo recinto em alta velocidades e que o sistema de segurança precisa reconhecer a presença de todos os que passam em seu campo de visão, acredita-se que a cadência de 6fps é mais adequada para o projeto para que este também contenha uma certa margem de segurança.

Contudo, assim como ocorre na resolução da imagem, que pode ser aumentada quando um invasor é reconhecido, a cadência também pode ser elevada para aumentar a chance de obter um quadro com uma imagem clara.

4.4 Sistema de segurança

Ao falar sobre o sistema de segurança em si, foram realizadas as seguintes especificações com referência ao ambiente a ser monitorado.

4.4.1 As zonas do sistema

Com a função de tornar precisa a ação preventiva quando esta for acionada é útil dividir o ambiente monitorado em zonas de segurança, como no esquema da figura 7.

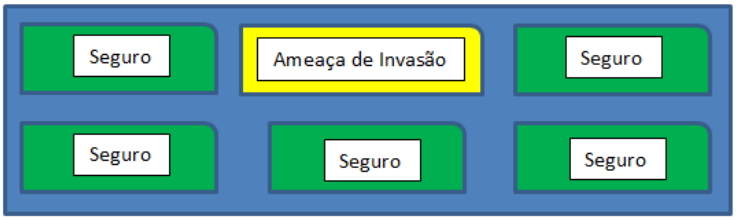


Figura 7 – Exemplo de divisão do ambiente em zonas de segurança

4.4.2 Os estados do sistema

Um sistema de segurança possui como produto de seu funcionamento um estado que diz se o ambiente está seguro ou se alguma ação deve ser tomada para garantir a segurança. Por exemplo:

- Seguro
- Ameaça de invasão
- Invasão

O sistema opera normalmente quando em estado seguro, verificando todas as variáveis periodicamente para validar esse estado, caso algo saia do normal o sistema pode passar para os estados ameaça de invasão ou invasão onde deverá encaminhar uma mensagem de alerta. Um diagrama destas transações pode ser observado na figura 8.

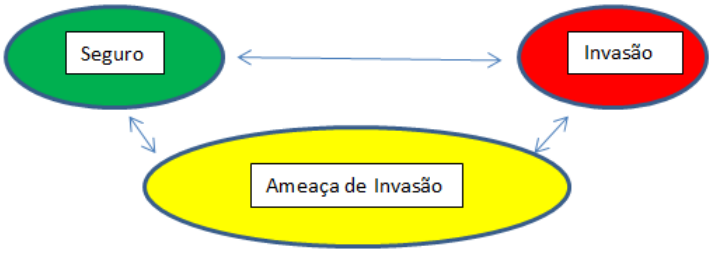


Figura 8 – Diagrama de transição de estados do sistema

4.4.3 Os atributos do sistema

Um conjunto de variáveis internas e externas será utilizado para se determinar o estado de segurança do sistema.

Sensores de movimento, câmeras de vigilância e sensores luminosos podem ser usados para captar informações do ambiente e são exemplos de variáveis externas.

Dia da semana e hora local são variáveis que podem ser monitoradas sem a necessidade de se instalar sensores no ambiente e são exemplos de variáveis internas.

A relação entre estes atributos do sistema com o estado de segurança do sistema pode ser observado na imagem 9

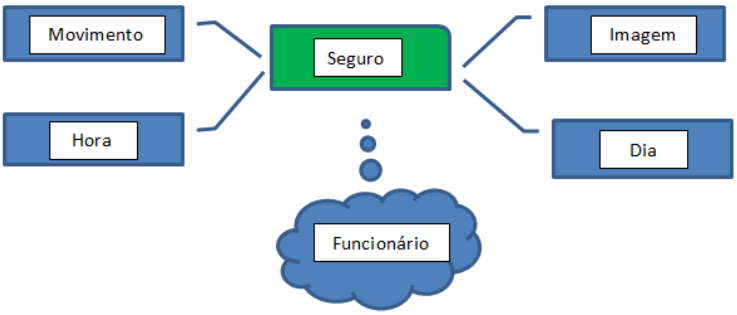


Figura 9 – Relação das variáveis do sistema com o estado de segurança do sistema

Uma vez que o sistema entre nos estados Ameaça de Invasão ou Invasão ele somente poderá voltar para o estado Seguro com a ação de um funcionário da segurança que deve ter feito a devida inspeção da zona ameaçada.

5 Projeto

Nesta seção vamos descrever o conjunto de softwares e hardwares adotados para colocar em teste a proposta de arquitetura para domótica que foi apresentada por esse trabalho.

A arquitetura multi-agente baseada em ROS que foi escolhida no capitulo referente aos requisitos de projeto será implementada num hardware *Raspberry Pi* que deverá suportar, além da arquitetura, o processamento da biblioteca *OpenCV*, o driver da câmera de vídeo e todos os agentes necessários para realizar a tarefa de supervisionamento.

Em paralelo a esse sistema, teremos um sistema de informações que será responsável por apresentar os dados coletados relativos à segurança, inclusive o vídeo captado pelas câmeras que estiverem transmitindo.

5.1 Estrutura do sistema

Este projeto implementa um sistema de segurança distribuído que pode monitorar em paralelo diversos locais fazendo uso de uma arquitetura multi-agente.

O diagrama da figura 10 ilustra o funcionamento do sistema.

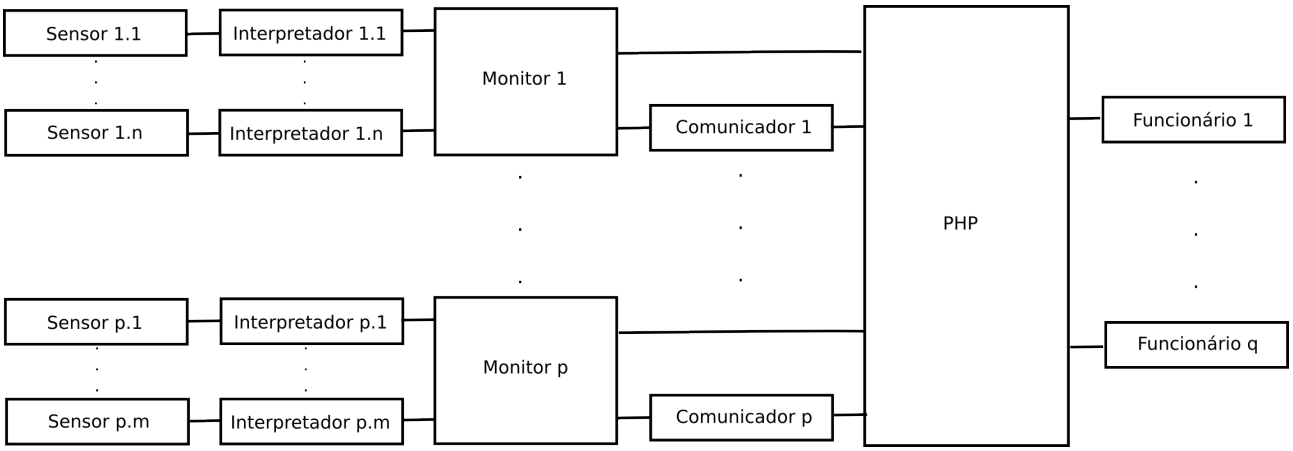


Figura 10 – Diagrama do funcionamento do sistema

Agente Sensor O agente sensor é o agente responsável pela captação das imagens através da câmera USB e a disponibilização destas imagens em um feed de mensagens do tipo publisher-subscriber. Deste modo, as imagens captadas pela câmera ficarão disponíveis para que outros agentes recebam essas mensagens e possam efetuar as tarefas atribuídas a eles.

Agente Interpretador O agente interpretador possui como funções receber o feed de mensagens contendo as imagens enviadas pelo agente sensor, interpretar estas imagens, ou seja, verificar se houve ou não uma invasão do ambiente e notificar os outros agentes sobre a análise das imagens para que as devidas providencias sejam tomadas.

Agente Monitor O agente monitor é responsável por convergir os sinais de status de vários agentes interpretadores de uma zona, as imagens enviadas pelos agentes sensores desta mesma zona e também uma comunicação com o servidor para saber o status desta zona no sistema de informações. Com estas informações este agente será capaz de determinar o estado da zona atual e salvar as imagens em disco.

Agente Comunicador O agente comunicador é responsável por fazer a transmissão das imagens salvas em disco pelo agente monitor para o sistema de informações.

Agente Funcionário O agente funcionário é responsável por voltar o sistema ao modo seguro após uma invasão. Vários funcionários podem ser responsáveis pela segurança da mesma zona.

Como é possível observar através da figura 10 cada zona possui apenas um agente monitor e um agente comunicador, no entanto, elas podem possuir inúmeros agentes sensores e interpretadores. Contudo, estes dois agentes precisam existir sempre aos pares, pois este conjunto será a representação de uma câmera. O agente sensor captando as imagens e o agente interpretador analisando estas imagens.

Também é possível observar que apenas os agentes monitor e comunicador tem conhecimento da existência do sistema de informação e que o funcionário nunca se comunicará diretamente com os agentes do sistema ROS, este utilizará a interface homem-máquina para se comunicar com o sistema de informação, que passará as informações para os agentes monitores.

5.2 Desempenho esperado

O sistema modelado para essa prova de conceito possui quatro funções principais que deverão ser executadas pelo conjunto. São elas: “Captação de imagens, Processamento e geração da informação, transmissão e apresentação dos dados”,

Para realizar a tarefa de captação de imagens o Raspberry Pi precisa ter um agente que suporte o driver *uvc_camera* (TOSSELL, 2013), que é capaz de captar as imagens de uma câmera USB. Após a captação das imagens, no entanto, elas não estão prontas para serem transmitidas para outros agentes, portanto, é necessário converter estas imagens para um formato transmissível pelo *ROS*. Isto se dá através do pacote *image_transport*

(MIHELICH, 2013), que permite a publicação e a subscrição do feed de imagens e oferece suporte para o transporte de imagens em formatos comprimidos e em conexões de baixa capacidade de transmissão.

Com as imagens captadas, o próximo passo é o processamento destas para geração de informações. Para o processamento de imagens é necessária a conversão das imagens para um formato que permita a manipulação destas. Isto se dá através do formato *Mat* da biblioteca OpenCV (Open Source Computer Vision Library) (OPENCV, 2013). OpenCV é uma biblioteca de funções voltadas para a visão computacional em tempo real. Ela foi desenvolvida pela Intel em 1999 e possui suporte para diversas linguagens de programação, inclusive C++. Com esta biblioteca é possível analisar as imagens pixel a pixel e assim realizar a detecção de intrusões.

Como uma invasão no ambiente monitorado pode ser detectado quando houver um movimento no ambiente, é necessário um algoritmo para detecção de movimento nas imagens. Diversos são os algoritmos para tal e neste projeto utilizaremos um algoritmo de comparação de frames subsequentes. Um maior detalhamento deste algoritmo é detalhado no apêndice D.1.

Outros dois algoritmos para detecção de invasões foram desenvolvidos e testados, no entanto, optou-se pela utilização do algoritmo de comparação de frames subsequentes. Mais detalhes sobre os outros algoritmos podem ser encontrados no apêndice D.

Com a informação de invasão de várias câmeras de uma mesma zona é possível determinar o estado desta zona, ou seja, se a zona se encontra em estado de segurança, alerta ou invasão. Este estado da zona é transmitido para o servidor através de uma chamada HTTP e casa a zona se encontre em estado de alerta ou invasão a imagem das câmeras da zona também são enviadas para o servidor através de uma chamada HTTP e codificando a imagem em base64 para transmitir informações binárias em forma de texto.

Como interface com o usuário nós escolhemos adotar a linguagem PHP por sua facilidade de acesso a dados, por ser multi-plataforma: funcionando tanto em Linux, Windows e Solaris, e também por possuir código-fonte aberto.

Esse sistema será desenvolvido utilizando um framework que utiliza o padrão MVC com o objetivo de proporcionar agilidade no desenvolvimento e estará armazenado num web server que pode ser hospedado na nuvem, essa é mais uma vantagem de sua utilização pois os dados não são armazenados nas dependências do local monitorado.

A comunicação com a arquitetura multi-agente será feita via chamada http-post tanto para transferência do estado do sistema quanto para a transferência das imagens, isso implica numa solução de compromisso entre resolução da imagem e velocidade de transferência de frames.

O sistema utilizará o projeto ffmpeg para trabalhar com os vídeos. Para acelerar

a interpretação dos dados pelo usuário a apresentação dos dados deve ser clara, direta e se aproximar ao máximo do tempo real para garantir a maior segurança.

6 Resultados

Para a realização da prova de conceito foram realizadas implementações do sistema completo em funcionamento com diferentes formações dos agentes como pode ser observado na seção 6.1. Posteriormente, na seção 6.2, são discutidos os resultados obtidos desta prova de conceito e a implementação das propriedades definidas pelo modelo de referência.

6.1 Implementação do sistema completo

Para testar o funcionamento do sistema completo foram utilizados duas unidades do *Raspberry Pi* com uma câmera USB conectada a cada um destes. Uma imagem do sistema pode ser observado na figura 11.

Figura 11 – Sistema ROS completo

Além disso foram realizados dois testes com duas formações diferentes dos agentes como será descrito nas próximas seções.

Ambas as configurações dos agentes funcionaram sem problemas e puderam cumprir com seu propósito: detectar uma invasão na zona, informar o usuário final e gravar as imagens das câmeras no banco de dados.

6.1.1 Teste 1

No primeiro teste realizado foram colocados os dois *Raspberry Pi*s para funcionar, sendo que cada um destes Hardwares representava uma zona. Desta maneira, cada um

dos *Raspberry Pi*s possui todos os agentes: sensor, interpretador, monitor e comunicador. Além disso, um deles ainda deve possuir um Master rodando nele.

Uma ilustração do sistema pode ser observado na figura 12.

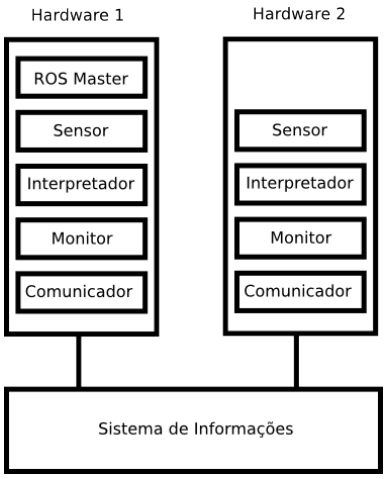


Figura 12 – Diagrama com a estrutura do sistema no teste 1

6.1.2 Teste 2

No segundo teste realizado foram colocados os dois *Raspberry Pi*s em uma mesma zona, ou seja, apenas um destes Hardwares deveria conter um agente monitor e uma agente comunicador. Como pode ser observado na figura 13 o Hardware 1 possui o ROS Master nele enquanto o Hardware 2 possui os agentes monitor e comunicador. Desta maneira, o Hardware 1 se comunica apenas com o Hardware 2, enquanto o Hardware 1 será responsável por toda a comunicação com o sistema de informações.

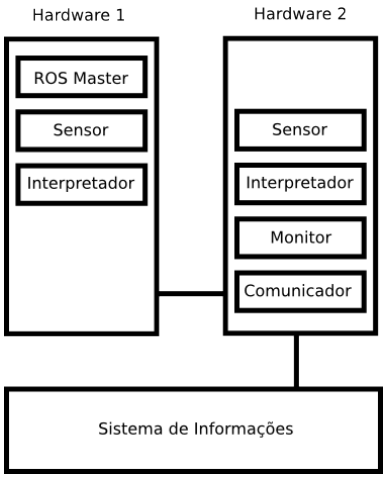


Figura 13 – Diagrama com a estrutura do sistema no teste 2

6.2 Discussão dos resultados

A implementação do sistema completo foi realizada com sucesso em ambos os testes feitos, com as diferentes formações dos agentes. Com isso foi possível observar as propriedades idealizadas pelo modelo de referência da seção 3.2, o sucesso da implementação destas propriedades e outros pontos julgados necessários de serem abordados.

6.2.1 Raspberry Pi

Durante a realização dos testes o Raspberry Pi se mostrou completamente capaz de suportar 5 agentes funcionando simultaneamente em seu sistema. Isso acarretou numa diminuição de rendimento dos agentes individuais, mas não afetou o funcionamento global do sistema.

A principio o sistema suportaria a adição de novos agentes neste mesmo Raspberry Pi, mas a real exequibilidade desta implementação precisa ser colocada a prova para confirmarmos sua plausibilidade.

6.2.2 Funcionamento da arquitetura

No modelo convencional de arquitetura multi-agente os agentes se comunicam entre si para determinar o que fazer, mas na arquitetura utilizada os agentes já possuem tarefas pré-definidas e possuem pouca ou nenhuma inteligência na execução de suas tarefas.

Funcionalmente esta pré-definição da ação dos agentes possibilitou os agentes a focarem 100% do tempo na execução das tarefas e não na comunicação com outros agentes para a definição do que fazer, por isso não encontraram o problema destacado em (SMITH, 1980), onde os agentes passam mais tempo combinando o que fazer do que efetivamente realizando a tarefa.

6.2.3 Algoritmo de detecção de invasão

O algoritmo utilizado para detecção de invasão, comparação de frames subsequentes, mostrou-se capaz e suficiente para realizar a tarefa designada. Obviamente o algoritmo possui suas limitações e pontos fracos, como a incapacidade de detecção de invasão em um plano muito distante; pessoas caminhando muito devagar à frente da câmera, pois a diferença entre os frames se torna muito pequena; entre outros. No entanto, o algoritmo se conseguiu detectar uma invasão em todas as ocasiões de pessoas andando normalmente em primeiro plano.

6.2.4 Arquitetura Mutli-agente

A implementação de um sistema com a arquitetura multi-agente possiblitou ao sistema uma fácil expansão em escala. Nos dois testes realizados foi possível observar a facilidade de adição de um novo agente sensor e interpretador no teste 2 e um agente sensor, interpretador, monitor e comunicador no teste 1. Este mesmo procedimento poderia ser aplicado mais vezes para expansão de número de câmeras por zonas e também de zonas no espaço.

No âmbito da expansão em escopo foi possível observar a facilidade de adição de uma nova funcionalidade ao sistema durante a execução do projeto. Por exemplo, o projeto inicial não previa a existência de um agente comunicador no sistema, no entanto, devido a problemas de velocidade de comunicação este agente foi implementado e a facilidade para adição de um novo tipo de agente ao sistema foi enorme devido a não necessidade de modificação nos agentes pré-existentes.

6.2.5 Flexibilidade

Após implementação dos dois testes na seção 6.1 foi possível verificar através da mudança de um teste para o seguinte que a flexibilidade do sistema é enorme, pois apenas é necessário modificar a atribuição de zona dos agentes sensor e interpretador do Hardware 1 para que o sistema já tivesse uma estrutura completamente diferente em funcionamento. Desta maneira se pode demonstrar que o sistema possui uma grande flexibilidade, pois ao modificar o posicionamento (zona) de uma câmera, é apenas necessário mudar a atribuição de zona deste agente.

Em outro caso, quando a criação de uma nova zona é necessário, também é necessária a criação de um agente monitor e um agente comunicador para esta zona. No entanto, apesar da necessidade de criar mais agentes, a dificuldade para realização de tal tarefa continua mínima.

6.2.6 Agentes leves

Os agentes desenvolvidos neste projeto realizam tarefas mínimas, mas somando as ações de todos eles é possível realizar uma tarefa mais complexa, detecção de invasão de uma zona. Estes agentes foram desenvolvidos desta maneira para torná-los leves e possibilitar a separação deles em agentes diferentes caso o hardware não suportasse o processamento requisitado por eles. No entanto, como pode ser observado no primeiro teste realizado, os agentes desenvolvidos são bastante leves e possibilitaram a execução de todos eles simultaneamente em um mesmo nó, Hardware 1.

Obviamente o processamento do Raspberry Pi não é infinito e, portanto, os próximos agentes desenvolvidos devem seguir a mesma linha de agentes leves e simples para que,

a partir do momento que o hardware não possua capacidade de processamento suficiente, os agentes possam ser separados em agentes diferentes.

6.2.7 Controle distribuído

O controle distribuído idealizado pelo modelo de referência visava a não necessidade de uma central de controle de todas as zonas e convergência de sinais. Este objetivo foi em parte atingido, pois já existe um controle distribuído por parte dos agentes monitores que possuem controle, cada um, sobre uma zona. No entanto, ainda existe um ponto de confluência de todos as informações das zonas que é o servidor e este obstáculo não pôde ser contornado, tornando-se este um ponto fraco do sistema.

Além disso, o sistema como um todo ainda necessita de um agente master em funcionamento para o sistema funcionar, sendo este também um ponto falho. No entanto, este não é um ponto tão crítico, pois o agente master pode estar localizado em qualquer um dos hardwares e para este problema já existem algumas soluções de multi-master que poderiam ser aplicadas a este sistema, mas que não foram escopo deste.

6.2.8 Segurança da rede

Como o sistema na plataforma ROS não oferece segurança, é necessário implementar uma camada de rede para assegurar a não invasão de terceiros à rede onde dados e comunicações sigilosas são transmitidos e armazenados. Isso pode ser obtido através da implementação de uma VPN. Com isso os agentes e o servidor conseguem se comunicar de maneira segura e mesmo que um possível invasor conheça, por exemplo, o endereço do servidor utilizado, ele não conseguirá acessá-lo a menos que esteja dentro da mesma VPN.

6.2.9 Velocidade de transmissão de dados

A velocidade de transmissão das imagens do agente comunicador para o servidos ainda é um ponto a ser melhorado. O sistema é capaz de enviar as imagens ao servidor, contudo ele não consegue realizar esta tarefa em tempo real e, portanto, existe um acúmulo de imagens a serem enviadas para o servidor mesmo após a atualização do estado de uma zona para seguro novamente. Desta maneira, as imagens serão disponíveis para o usuário final com um atraso que pode ser bastante significativo dependendo de quanto tempo de vídeo foi armazenado, porque o atraso é acumulativo e o agente apenas consegue recuperar este atraso quando a zona está em estado de segurança e não é necessário o envio de novas imagens para o servidos.

6.2.10 PHP

O sistema de informação mostrou-se eficaz nessa prova de conceito, atendendo as exigências de aparência e clareza. O projeto ffmpeg também se mostrou muito eficaz por ser capaz de criar os vídeos em tempo real.

Como previsto, a comunicação via http encontrou dificuldades de congestão e demora no ensaio dos frames mas que foi resolvido com a introdução do agente comunicador.

7 Conclusão

Como dito anteriormente o propósito deste trabalho é mostrar a implementabilidade de uma estrutura com arquitetura multi-agente e a partir do desenvolvimento de um sistema de segurança implementado segundo um modelo de referência cuja arquitetura se baseia em uma arquitetura multi-agente.

O primeiro claro ponto positivo da implementação de um sistema multi-agente é a fácil expansão do sistema tanto em escala. No sistema abordado uma expansão em escala seria a adição de novas câmeras ao sistema. Esta tarefa seria facilmente atingida colocando em operação mais agentes sensores, interpretadores e quando necessário agentes monitores e comunicadores (caso as novas câmeras estejam localizadas em novas zonas). Dada a arquitetura do sistema, uma ação de adição de novos agentes não teria influência sobre os outros agentes e, portanto, esta expansão é ilimitada diferentemente dos atuais sistemas de segurança que possuem limitações de número de câmeras, número de sensores, etc (VERISURE, 2013), (SMSOLUTION, 2013), (SU; LEE; WU, 2006).

Outro ponto a ser destacado é a facilidade da expansão em escopo do sistema. Devido a forma de comunicação entre os agentes, publisher e subscriber, este sistema pode ser, com facilidade, acrescido em novas funcionalidades. Por exemplo, o agente sensor já disponibiliza as imagens obtidas da câmera em um feed de mensagens, assim como o agente interpretador envia o status de invasão desta câmera em outro feed. Desta maneira é possível uma implementação de um agente que receba as imagens dos agente sensor e o status do agnete interpretador e caso exista uma invasão faça o reconhecimento de face no invasor. Esta seria uma funcionalidade que poderia ser acrescida ao sistema, mas como as imagens são publicadas em feeds de mensagens, ilimitados subscribers podem receber as mensagens e realizar diferentes tarefas, portanto, a expansão de funcionalidades do sistema se dá de uma maneira relativamente simples, pois não há a necessidade de modificação de agentes pré-existentes.

Com o desenvolvimento deste sistema de segurança se pode notar mais uma das vantagens da implementação de uma arquitetura multi-agente como a proposta: os agentes conseguem funcionar independentemente do funcionamentos dos outros agentes. Obviamente alguns agentes necessitam de informações fornecidas por outros agentes para executarem suas tarefas, mas um agente continua em modo de operação mesmo quando outro agente para de funcionar. Com isso, é possível que exista um agente monitor que seja capaz de recolocar um agente parado em funcionamento ou mesmo colocar um agente parado em modo de operação em outro hardware para que este volte a fornecer informações para um terceiro agente. Esta seria uma funcionalidade muito útil para o sistema,

pois o próprio sistema seria capaz de detectar mal funcionamentos e corrigí-los, tornando assim o sistema mais inteligente e independente, além de mais seguro. Contudo, esta funcionalidade não seria aplicável apenas ao sistema de segurança proposto. Esta seria uma característica da arquitetura dos agentes, que poderia ser utilizada para a implementação de diversos outros sistemas de automação e não somente à sistemas de segurança.

Obviamente a implementação deste sistema com esta arquitetura também possui os seus pontos negativos. O primeiro ponto a ser abordado é que a implementação dos agentes no Raspberry Pi tem as suas vantagens do ponto de vista de flexibilidade do sistema, ou seja, os hardwares conseguem ser facilmente trocados de posição, contudo este fato também impôs uma limitação na capacidade de processamento dos agentes, pois o processador deste hardware também é limitado e, portanto, é necessário construir os agentes de forma que estes efetuem tarefas simples e que conjuntamente consigam efetuar tarefas complexas. Desta maneira, é possível dividir agentes que necessitem de mais poder de processamento em hardwares diferentes.

Outro ponto a ser abordado é a segurança da rede, pois não existe segurança na camada dos agentes no sistema ROS, portanto, qualquer um poderia ler as mensagens publicadas em feeds pelos agentes bem como modificar os estados das zonas facilmente, por isso, é imprescindível que o sistema inteiro funciona embaixo de uma VPN para que esta realize o papel da camada de segurança do sistema.

O terceiro ponto falho deste sistema é a centralização de atividades. Para que o sistema desenvolvido funcione é necessário o funcionamento de um agente master, que comanda toda a comunicação entre os agentes e sem o funcionamento deste o sistema inteiro para de funcionar. Portanto, este é um ponto falho do sistema, mas já existem algumas soluções multi-master para sanar este problema. Além disso, existe a convergência de sinais de todas as zonas para um único ponto que é o servidor, logo, este também é um ponto falho do sistema que pode se tornar um alvo para possíveis invasores.

7.1 Trabalhos futuros

Com o desenvolvimento deste trabalho, muitos caminhos podem ser seguidos em trabalhos a serem desenvolvidos no futuro. Estas possibilidades foram divididas em duas categorias: melhorias na estrutura multi-agente (seção 7.1.1) e melhorias e implementação de novas funcionalidades no sistema (seção 7.1.2).

7.1.1 Estrutura multi-agente

Algumas melhorias no estrutura multi-agente do sistema poderiam ser desenvolvidas para melhorar a qualidade da estrutura. Um exemplo de uma melhoria nesta categoria seria a implementação de um sistema com múltiplos masters. Na atual configuração o sis-

tema conta com um único master e este é obrigatório para que o sistema funcione. Deste modo, se, em algum momento, o hardware contendo o master é desconectado ou desligado, o sistema inteiro perderá sua funcionalidade. Por isso, este é um ponto falho no sistema.

Outra melhoria seria a capacidade do próprio ser capaz de verificar se um agente possui um mal funcionamento ou sofreu um ataque e o próprio sistema conseguir de fazer o agente voltar a funcionar. Além disso, o sistema poderia detectar que um hardware não funciona e fazer os agentes deste hardware funcionarem em um hardware diferente para que o sistema não perca suas funcionalidades.

7.1.2 Funcionalidades do sistema

No âmbito de funcionalidades do sistema diversas novas funcionalidades poderiam ser desenvolvidas para o sistema. Como o sistema foi desenvolvido de forma a ser altamente expansível em escopo, novos agentes podem facilmente ser implementados para desenvolverem funções diferentes no sistema. Algumas ideias de agentes seriam um agente para travamento de portas e janelas, sensor de temperatura e fumaça para detecção de possíveis incêndios, entre outros.

Referências

ABREU, B. et al. Video-based multi-agent traffic surveillance system. In: *Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE*. [S.l.: s.n.], 2000. p. 457–462.

ADMIN, S. *Site Templates - Slate Admin | ThemeForest*. 2013. http://themeforest.net/item/slate-admin/133854.

BROWNLOW, K. Silent Films: What Was the Right Speed? *Sight and Sound*, p. 164–167, 1980.

GOMASA, S. P. *Modular Design and Implementation of a Low Cost Home Automation System using Web-Services*. Dissertação (Mestrado) — Massey University, Abany, New Zealand, 2011.

HSU, C.-L.; YANG, S.-Y.; WU, W.-B. Constructing intelligent home-security system design with combining phone-net and bluetooth mechanism. In: *Machine Learning and Cybernetics, 2009 International Conference on*. [S.l.: s.n.], 2009. v. 6, p. 3316–3323.

KUBERA, Y.; MATHIEU, P.; PICAULT, S. Everything can be agent! In: *AAMAS*. [S.l.: s.n.], 2010. p. 1547–1548.

Hongyue Luo. *Intelligent Home Security System*. 2007. US 2007/0182543 A1. Disponível em: <http://www.patentlens.net/patentlens/patent/US_2007_0182543_A1/en/>.

MIHELICH, P. *image_transport - ROS Wiki*. 2013. http://www.ros.org/wiki/image_transport.

NYFFENEGGER, R. *CMSSW_3_9_7 Reference Manual*. 2013. https://cmssdt.cern.ch/SDT/doxygen/CMSSW_3_9_7/doc/html/db/dab/PixelBase64_8cc.html.

OPENCV. *OpenCV | OpenCV*. 2013. http://www.opencv.org/.

SHARPLES, S.; CALLAGHAN, V.; CLARKE, G. A Multi-Agent Architecture for Intelligent Building Sensing and Control. *International Sensor Review Journal*, v. 19, p. 135–140, fev. 1999.

SMITH, R. G. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.*, IEEE Computer Society, Washington, DC, USA, v. 29, n. 12, p. 1104–1113, dez. 1980. ISSN 0018-9340. Disponível em: <http://dx.doi.org/10.1109/TC.1980.1675516>.

SMSOLUTION. *SMSolution - Sistemas de segurança - Alarmes - - Alarmes - Câmeras - Portões Automáticos - Interfones*. 2013. http://smsolution.com.br/2013/alarmes/.

SONG, G. et al. A surveillance robot with hopping capabilities for home security. *Consumer Electronics, IEEE Transactions on*, v. 55, n. 4, p. 2034–2039, 2009. ISSN 0098-3063.

SU, J.-H.; LEE, C.-S.; WU, W.-C. The design and implementation of a low-cost and programmable home automation module. *Consumer Electronics, IEEE Transactions on*, v. 52, n. 4, p. 1239–1244, 2006. ISSN 0098-3063.

TOSSELL, K. *uvc_camera - ROS Wiki*. 2013. http://www.ros.org/wiki/uvc_camera.

VALLEJO, D. et al. A multi-agent architecture for supporting distributed normality-based intelligent surveillance. *Eng. Appl. of AI*, v. 24, n. 2, p. 325–340, 2011.

VERISURE. *Alarmes para residências | Verisure Brasil*. 2013. http://www.verisure.com.br/alarme-residencias.

Apêndices

APÊNDICE A – Diagramas do sistema ROS

A.1 Casos de uso

A.1.1 Especificação de casos de uso

Nesta seção serão tratados os casos de uso do sistema ROS. Para tal pode ser observado na subseção A.1.1.1 o diagrama de casos de uso do sistema. Em seguida podem serão identificados os atores do sistema na subseção A.1.1.2 e os casos de uso na subseção A.1.1.3. E por fim o detalhamento de cada um dos casos de uso na subseção A.1.1.4.

A.1.1.1 Diagrama de casos de uso

A figura 14 mostra o diagrama de casos de uso desenhado para o sistema ROS.

A.1.1.2 Identificação dos atores

Como especificado na seção 4, o sistema *ROS* conta com quatro atores: sensor, interpretador, monitor e comunicador.

Ator-01 Sensor Agente responsável pela captação das imagens

Ator-02 Interpretador Agente responsável pela interpretação das imagens

Ator-03 Monitor Agente monitorador da zona

Ator-04 Comunicador Agente responsável pela comunicação com o sistema de informações PHP

A.1.1.3 Identificação dos casos de uso

O diagrama de casos de uso do projeto do sistema ROS pode ser observado na figura 14.

Como pode ser observado neste diagrama, o agente sensor será responsável captar a imagem da câmera e publicar esta imagem em um feed de mensagens. A partir deste feed com as imagens da câmera os agentes interpretador e monitor irão subscrever à essa publicação.

Figura 14 – Diagrama de casos de uso do sistema ROS

O agente interpretador deverá analisar esta imagem e detectar possíveis invasões. De acordo com essa análise deverá publicar um feed de mensagens com uma mensagem de status.

O agente monitor estará subscrevendo ao feed de mensagens com as imagens da câmera e com o status da zona. Como haverá mais de um agente em uma mesma zona

e apenas um agente monitor por zona, este deve concentrar as mensagens de status da zona e também receber atualizações do PHP sobre o status da zona e atualizar o status da zona real. Com a definição do status real da zona, as imagens dos agentes sensores da zona deverão ou não ser salvas no em arquivo.

O agente comunicador ficará todo o tempo observado se existe uma imagem salva em arquivo no disco. Em caso positivo, este agente transmitirá essa imagem para o sistema de informações.

Estes casos de uso são listados abaixo.

UC-01 Captar imagem Capturar a imagem de uma câmera USB conectada ao hardware

UC-02 Publicar imagem Publicar imagem em um feed de mensagens

UC-03 Subscrever imagem Subscrever ao feed de mensagens da imagem

UC-04 Analisar imagem Analisar a imagem recebida e definir se há ou não uma invasão na imagem

UC-05 Publicar status Publicar status da câmera em um feed de mensagens

UC-06 Subscrever status Subscrever ao feed de mensagens do status

UC-07 Receber status PHP Receber o status da zona proveniente do sistema de informações PHP

UC-08 Atualizar status Atualizar status da zona no ROS e PHP, se necessário

UC-09 Salvar imagem Salvar imagem no disco

UC-10 Abrir imagem Abrir imagem salva no disco

UC-11 Transmitir imagem Transmitir imagem para o sistema de informações PHP

A.1.1.4 Detalhamento dos casos de uso

UC-01 Captar imagem Este caso de uso especifica a ação de capturar a imagem de uma câmera conectada a entrada USB do *Raspberry Pi*.

Atores Sensor

Pré-condições Câmera deve estar conectada a entrada USB do *Raspberry Pi*

Pós-condições As imagens estarão dentro do agente sensor para que as tarefas designadas a este agente possam ser executadas

UC-02 Publicar imagem Este caso de uso especifica a ação de modificar o formato da imagem capturada do formato de imagem do ROS para image_transport e publicar esta imagem em uma feed de mensagens para que outros atores possam utilizar estas imagens captadas em suas respectivas tarefas.

Atores Sensor

Pré-condições Imagem capturada

Pós-condições Feed de mensagens com a imagem da câmera

UC-03 Subscrever imagem Este caso de uso especifica a ação de subscrever ao feed de mensagens contendo a imagem da câmera do agente sensor.

Atores Interpretador e Monitor

Pré-condições Feed de mensagens contendo a imagem publicado pelo agente sensor

Pós-condições Imagem disponível para agentes interpretador e monitor para que as tarefas designadas a estes agentes possam ser executadas

UC-04 Analisar imagem Este caso de uso especifica a ação de modificar o formato das imagens recebidos de image_transport para o formato de imagens do OpenCV, Mat, para que as imagens possam ser tratadas e analisadas. Também é aplicado as imagens um algoritmo de comparação de imagens subsequentes para detecção de intrusões e geração de uma variável booleana de status para designar a detecção ou não de um invasor neste sensor.

Atores Interpretador

Pré-condições Imagem disponível no agente interpretador

Pós-condições Status do invasão no agente sensor correspondente

UC-05 Publicar status Este caso de uso especifica a ação de publicar o status de invasão nas imagens da câmera em um feed de mensagens.

Atores Interpretador

Pré-condições Imagem analisada e variável de status setada

Pós-condições Feed de mensagens com status de invasão nas imagens da câmera

UC-06 Subscrever status Este caso de uso especifica a ação de subscrever ao feed de mensagens contendo o status de invasão nas imagens de todas as câmeras de uma determinada zona.

Atores Monitor

Pré-condições Feed de mensagens contendo os status de invasão das câmeras da zona

Pós-condições Status de invasão das câmeras de uma zona

UC-07 Receber status PHP Este caso de uso especifica a ação de receber o status de invasão de uma determinada zona no sistema de informações PHP.

Atores Monitor

Pré-condições Sistema de informações PHP rodando e zona cadastrada no sistema

Pós-condições Status de invasão da zona no sistema PHP

UC-08 Atualizar status Este caso de uso especifica a ação de analisar os status de invasão de todas a câmeras de uma determinada zona e também o status da zona no sistema de informações e determinar o atual status desta zona. Também é preciso atualizar este status de invasão da zona no sistema de informações através de uma chamada HTTP.

Atores Monitor

Pré-condições Status de invasão das câmeras da zona e da zona no sistema de informações

Pós-condições Status de invasão atual da zona

UC-09 Salvar imagem Este caso de uso especifica a ação de salvar a imagem captada pelas câmeras da zona em disco no caso de uma invasão ter sido detectada na zona.

Atores Monitor

Pré-condições Invasão detectada na zona

Pós-condições Imagem salva no disco

UC-10 Abrir imagem Este caso de uso especifica a ação de abrir a imagem salva em disco

Atores Comunicador

Pré-condições Existir uma imagem salva em disco

Pós-condições Imagem disponível no agente comunicador para que as tarefas designadas a este agente possam ser executadas

UC-11 Transmitir imagem Este caso de uso especifica a ação de transmitir a imagens para o sistema de informações PHP através de uma chamada HTTP.

Atores Comunicador

Pré-condições Imagem disponível no agente comunicador

Pós-condições Imagem enviada para o sistema de informações PHP

A.2 Diagrama de sequência

Para os casos de uso UC-04 Analisar Imagem e UC-11 Transmitir Imagem no diagrama de casos de uso 14 foram desenvolvidos os diagramas de sequência, figuras 15 e 16 respectivamente. Apenas estes casos de uso foram desenvolvidos devido a complexidade destes casos de uso e da importância deste no sistema.

No diagrama do caso analisar imagem 15 é possível observar que a imagem que é inicialmente no formato de image_transport será transformada para o formato Mat do OpenCV, a seguir a imagem deverá ser transformada em uma imagem binária e comparada com o frame anterior para a detecção de diferenças nos frames. Então é possível contar o número de pixels diferentes nestas duas imagens e setar o status de invasão para invasão caso existe diferença em mais de 2% dos pixels das imagens. Enfim salva-se a imagem para que o mesmo fluxo possa ser realizado com o próximo frame.

No diagrama do caso transmitir imagem 16 a imagem precisa ser codificada utilizando o padrão base64 para que possa ser enviada através de uma chamada HTTP, além disso é preciso aplicar o código percentagem para eliminar caracteres especiais. Feito isso, a chamada HTTP pode ser realizada e a imagem deve ser apagada do disco para que não seja enviada novamente ao servidor.

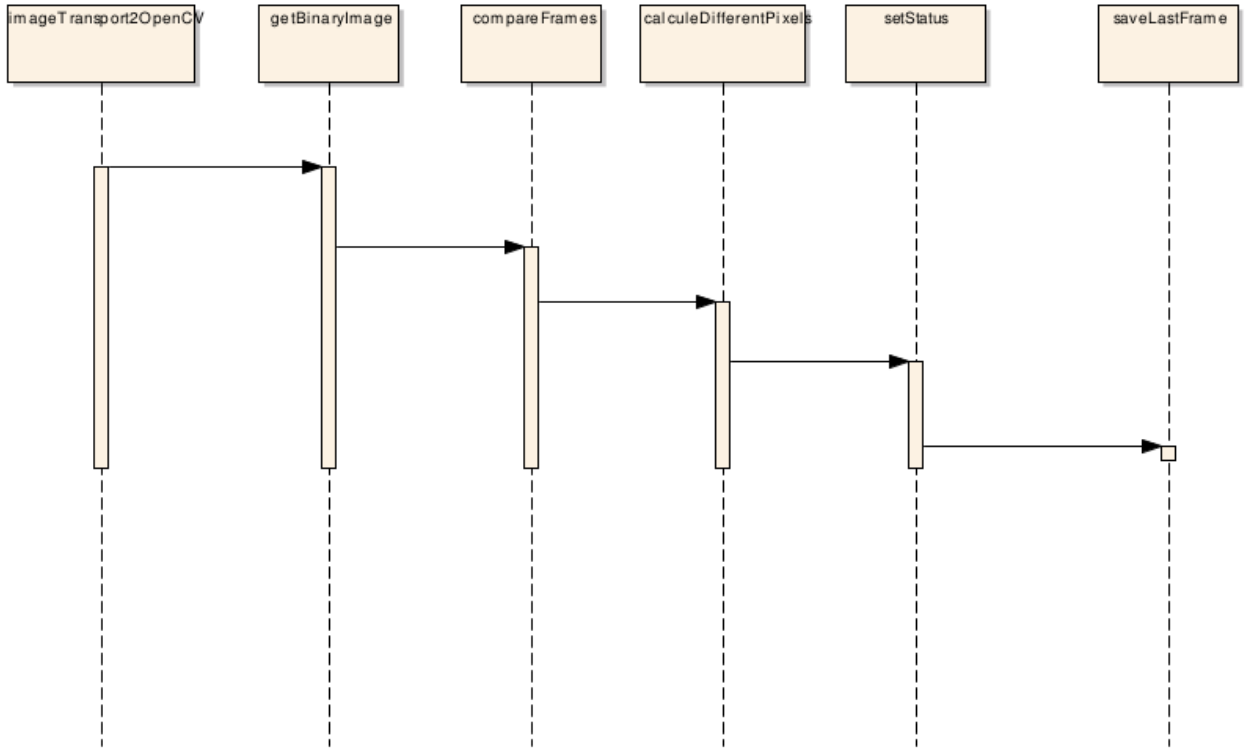


Figura 15 – Diagrama de sequência do caso analisar imagem

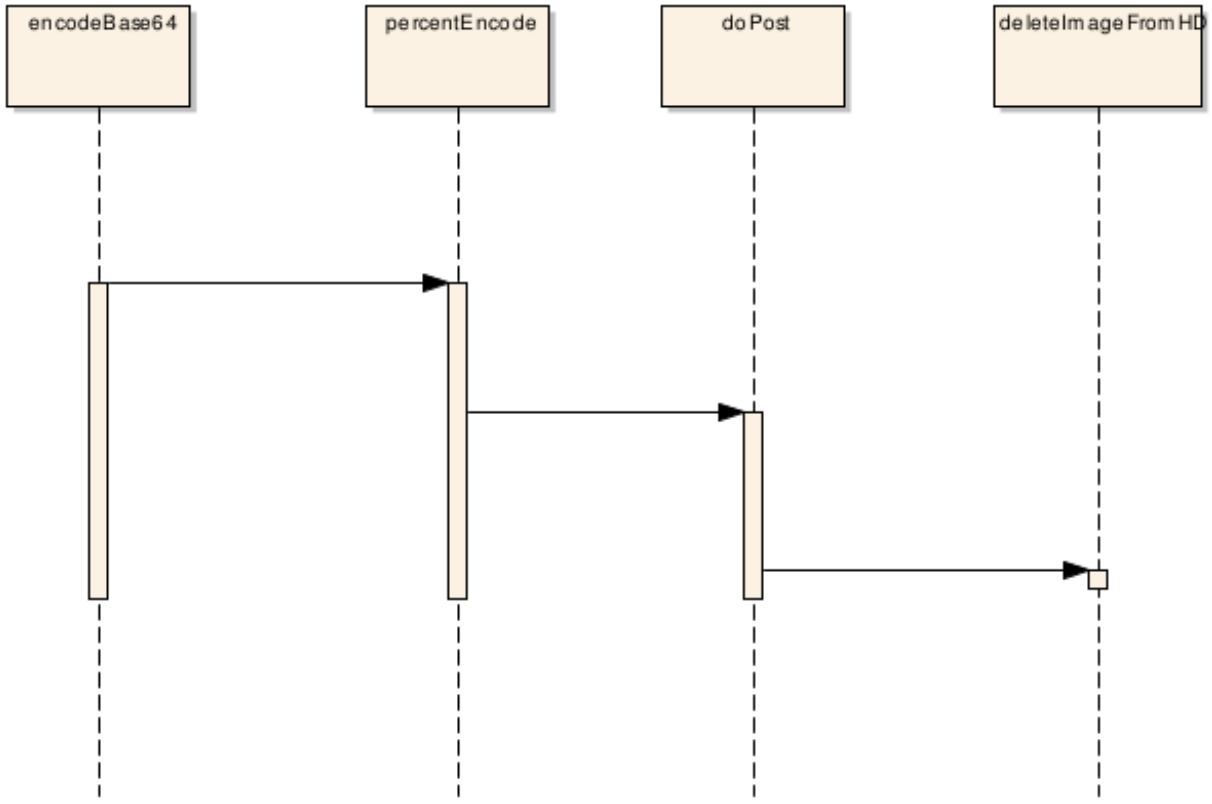


Figura 16 – Diagrama de sequência do caso enviar transmitir imagem

APÊNDICE B – Diagramas do sistema PHP

B.1 Casos de uso

B.1.1 Especificação de casos de uso

B.1.1.1 Diagrama de casos de uso

B.1.1.2 Identificação dos atores

Ator-01 Funcionário Pessoa responsável pela segurança do local, ele deve estar autenticado por login e senha.

Ator-02 ServerPHP Ator que realiza as ações internas do sistema.

B.1.1.3 Identificação dos casos de uso

UC-01 Fazer Login Permitir que um usuário possa ter acesso ao sistema para visualizar o estado do sistema, alterar o estado do sistema e baixar arquivos de vídeo da lista de logs do sistema.

UC-02 Fazer Logout Encerra a autenticação existente, assim impede o acesso ao sistema.

UC-03 Renomear Zona Altera o rótulo da zona.

UC-04 Visualizar Zona Exibe o rótulo da zona, o seu estado e a lista de eventos associada a essa zona.

UC-05 Listar Log de Eventos Exibe a lista de eventos conjunta de todas as zonas cadastradas no sistema.

UC-06 Fazer Download Força o download de um arquivo de vídeo.

UC-07 Visualizar Sistema Exibe os estados de todas as zonas cadastradas no sistema.

UC-08 Atualizar Status da Zona PHP Altera o estado da zona no sistema de informação.

UC-09 Enviar Imagem Permite que uma imagem seja enviada para o sistema de informação.

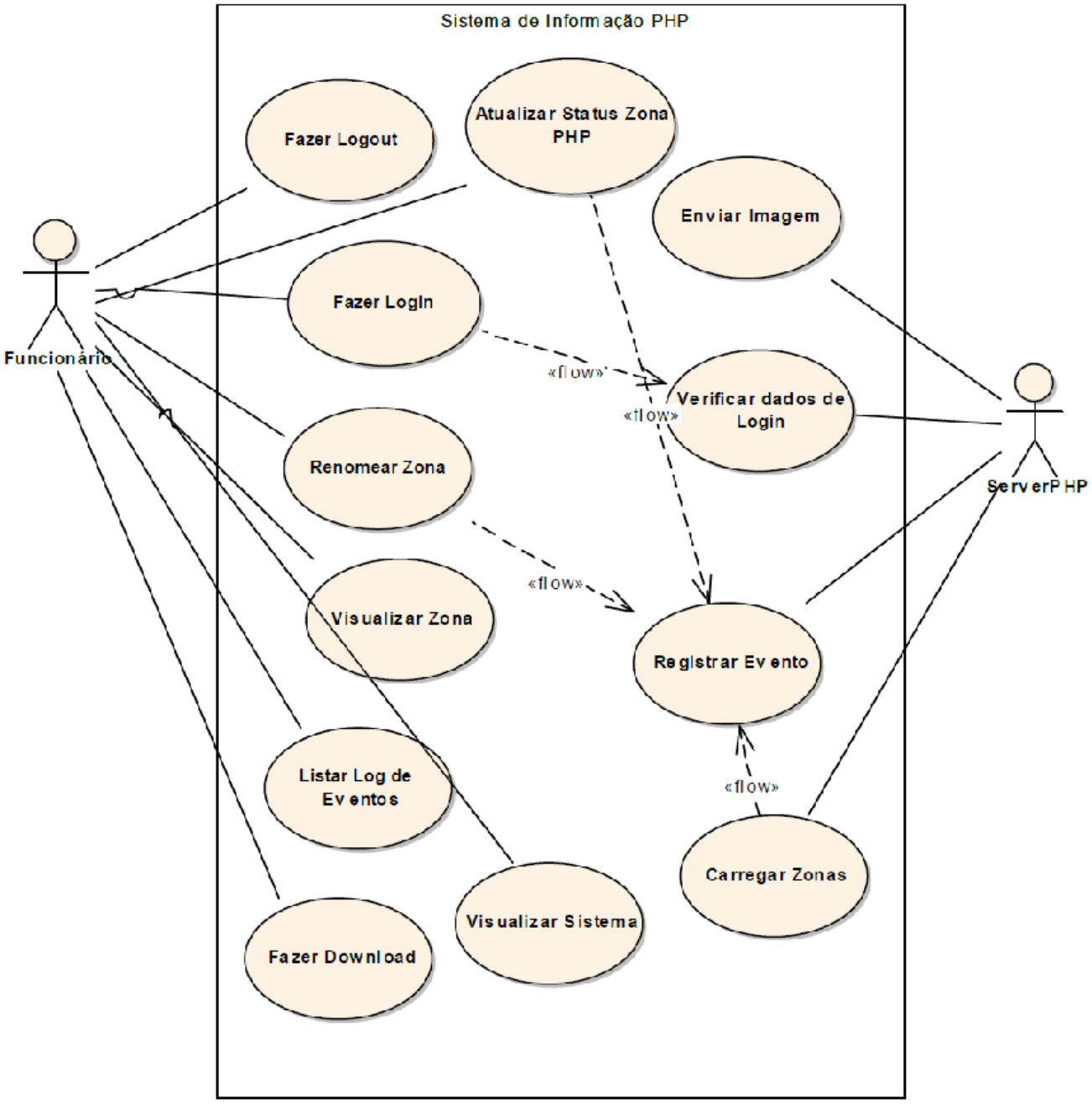


Figura 17 – Diagrama de casos de uso do sistema PHP

UC-10 Verificar dados de Login Faz a autenticação do funcionário por meio de usuário e senha.

UC-11 Registrar Evento Salva as informações de uma mudança de estado do sistema.

UC-12 Carregar Zonas Permite o cadastro de uma nova zona a ser monitorada ou altera o estado de uma zona existente.

B.1.1.4 Detalhamento dos casos de uso

UC-01 Fazer Login Este caso de uso especifica a ação de autenticação que um usuário executa no sistema, com o objetivo de conectar-se a ele. Apenas usuários cadastrados podem se conectar ao sistema. Devem ser passadas as informações de usuário e senha e, após a validação no sistema, o usuário recebe o acesso ao sistema de monitoramento. Só existe um tipo de usuário que é o funcionário da segurança e este possui permissão para: "Alterar o estado da zona, Renomear Zona, Fazer download do vídeo" e todas as visualizações.

Atores Funcionário

Pré-condições O ator deve estar cadastrado no sistema

Pós-condições O ator fica habilitado a interagir com o sistema

Fluxo básico

1. O ator decide se autenticar no sistema.
2. O sistema solicita as informações obrigatórias para autenticação: usuário e senha.
3. O ator informa os dados.
4. O sistema valida os dados.
5. O sistema habilita o acesso do ator.
6. A página de visualização do sistema é carregada

Fluxo alternativo A

1. No passo 4 do Fluxo Básico, caso haja erro na autenticação.
2. O sistema informa erro ao ator.
3. O sistema volta para o passo 2 do Fluxo Básico.

UC-02 Fazer Logout Este caso de uso especifica a ação de encerramento da sessão de trabalho. O objetivo é impedir invasões ao sistema por meio de uma sessão já aberta.

Atores Funcionário

Pré-condições O usuário precisa estar autenticado no sistema

Pós-condições O usuário não possui acesso ao sistema

UC-03 Renomear Zona Este caso de uso especifica a ação de editar o nome de uma zona. O objetivo é criar uma relação direta da zona que o funcionário está visualizando e o local que essa zona representa. Deve ser passados o código da zona e o novo nome dela.

Atores Funcionário

Pré-condições O usuário precisa estar autenticado no sistema e a zona precisa estar cadastrada

Pós-condições A zona está cadastrada com um novo rótulo

UC-04 Visualizar Zona Este caso de uso especifica a ação de visualização da zona. O objetivo é reunir toda a informação relativa a uma zona e exibi-la de forma clara e concisa. Deve ser passado o código da zona e serão exibidos o rótulo da zona, o seu estado e a lista de eventos associados a essa zona. Nessa tela o usuário deve ter a opção de alterar o estado da zona.

Atores Funcionário

Pré-condições O usuário precisa estar autenticado no sistema e a zona precisa estar cadastrada

Pós-condições

UC-05 Listar Log de Eventos Este caso de uso especifica a ação de listar o log de eventos do sistema, com o objetivo de passar ao usuário uma visão geral dos eventos que acontecem do sistema. O usuário precisa estar autenticado no sistema.

Atores Funcionário

Pré-condições O usuário precisa estar autenticado no sistema

Pós-condições

UC-06 Fazer Download Este caso de uso especifica a ação de fazer o download de um arquivo de vídeo do servidor. Somente usuários autenticados podem executar essa ação. Deve ser passado o código do arquivo que será baixado.

Atores Funcionário

Pré-condições O usuário precisa estar autenticado no sistema

Pós-condições O usuário possui o arquivo de vídeo

Fluxo básico

1. O ator decide baixar um arquivo do sistema.
2. O sistema recebe o código do arquivo.
3. O sistema cria um vídeo temporário com todas as imagens que encontrar no servidor.
4. O sistema força o download do arquivo.

UC-07 Visualizar Sistema Este caso de uso especifica a ação de visualizar o sistema. O objetivo é reunir toda a informação relativa ao sistema e exibi-la de forma clara e concisa.

Atores Funcionário

Pré-condições O usuário precisa estar autenticado no sistema

Pós-condições

UC-08 Atualizar Status da Zona PHP Este caso de uso especifica a ação de alterar o estado da zona que é feita pelo funcionário da segurança. Ele pode alterar o estado da zona para os seguintes estados: "Seguro, alerta ou invasão". Deve ser salvo um log da ação realizada. Deve ser passados o código da zona e o novo estado.

Atores Funcionário

Pré-condições O usuário precisa estar autenticado no sistema

Pós-condições O estado da zona é alterado. Um log da ação é criado

UC-09 Enviar Imagem Este caso de uso especifica a ação de envio de imagem para o sistema de informação. Devem ser passados: o código da zona que a imagem pertence, o código do arquivo de vídeo que está sendo construído, o número do frame que essa imagem representa e os dados da imagem. Essa ação retorna o estado da zona cujo código foi passado.

Atores ServerPHP

Pré-condições A zona deve estar cadastrada. O arquivo de vídeo deve estar cadastrado

Pós-condições A imagem fica salva no servidor. O estado da zona é retornado ao usuário

Fluxo básico

1. O sistema recebe um post do tipo enviarImagem.
2. O sistema recebe os dados do post: código da zona, código do arquivo vídeo, numero do frame, string da imagem.
3. O sistema busca o status da zona.
4. O sistema decodifica o string da imagem e a salva no servidor.
5. O sistema retorna o status da zona.

Fluxo alternativo A

1. No passo 3 do Fluxo Básico, caso haja erro na busca pelo status.
2. O sistema retorna um erro ao ator.

UC-10 Verificar dados de Login Este caso de uso especifica a ação de validação dos dados de autenticação. Devem ser passadas as informações de login: "usuário e senha".

Atores ServerPHP

Pré-condições Usuário e senha para serem validados

Pós-condições Usuário e senha validados

UC-11 Registrar Evento Este caso de uso especifica a ação de registrar os eventos que acontecem no sistema com o objetivo de criar um log de seu ciclo de vida. Devem ser passados o código da zona em que o evento ocorreu, o código da ação ocorrida, o código do funcionário que realizou a ação e , se existir, o código do arquivo de vídeo gerado.

Atores ServerPHP

Pré-condições As ações: "Marcar como Seguro, Marcar como Alerta, Marcar como Invasão, Renomear Zona, Inserir Zona" devem estar cadastradas. O funcionário, a zona e o arquivo de vídeo também devem estar cadastrados

Pós-condições Um evento é criado

UC-12 Carregar Zonas Este caso de uso especifica a ação de cadastrar uma zona no sistema de informação para passar a monitorá-la ou alterar o estado de uma zona já cadastrada. Devem ser passados o código da zona e o seu estado. Quando o estado da zona está em alerta ou invasão deve ser passado o código do arquivo de vídeo que será criado.

Atores ServerPHP

Pré-condições Zona não cadastrada ou zona cadastrada com um estado diferente do estado enviado

Pós-condições Zona cadastrada e com novo estado

Fluxo básico

1. O sistema recebe um post do tipo carregarZonas.
2. O sistema recebe os dados do post: número de zonas, código da zona, código do arquivo vídeo.
3. O sistema verifica se a zona já existe.
4. O sistema cadastra a nova zona.
5. Se o estado for 2 (alerta) ou 3 (invasão). O sistema cadastra o arquivo de vídeo.
6. O sistema registra o evento.

Fluxo alternativo A

1. No passo 3 do Fluxo Básico, caso a zona já exista.
2. O sistema atualiza o estado da zona.
3. O sistema vai para o estado 5 do Fluxo Básico.

B.2 Diagramas de sequência

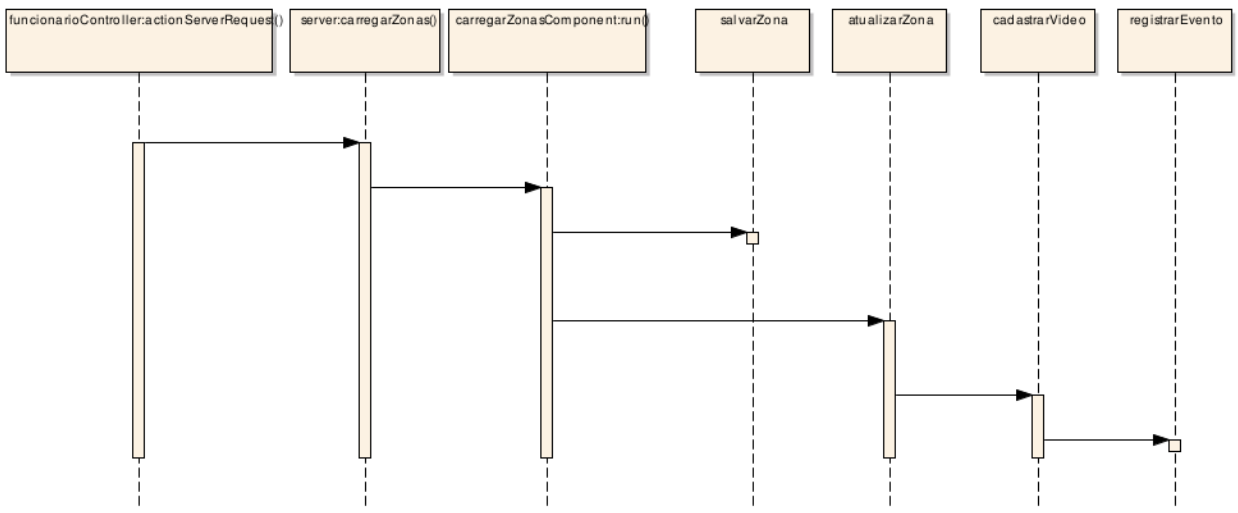


Figura 18 – Diagrama de sequência do caso carregar zonas

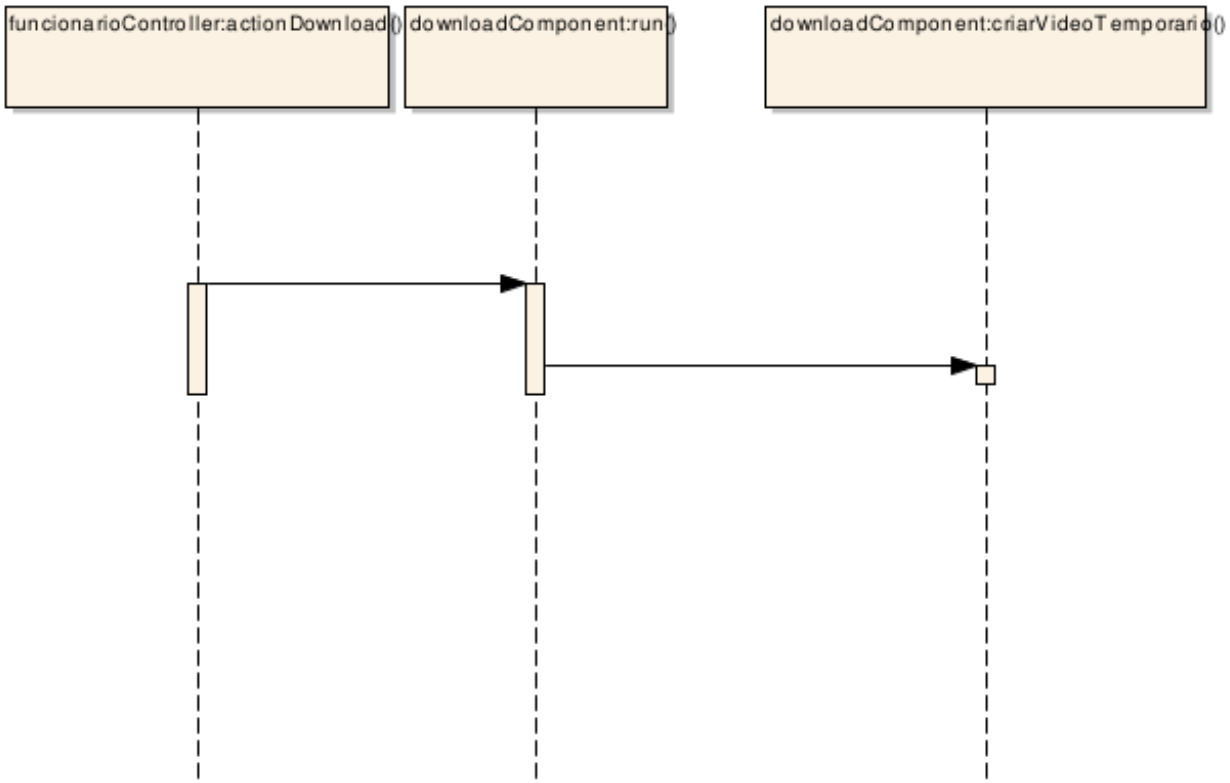


Figura 19 – Diagrama de sequência do caso fazer download

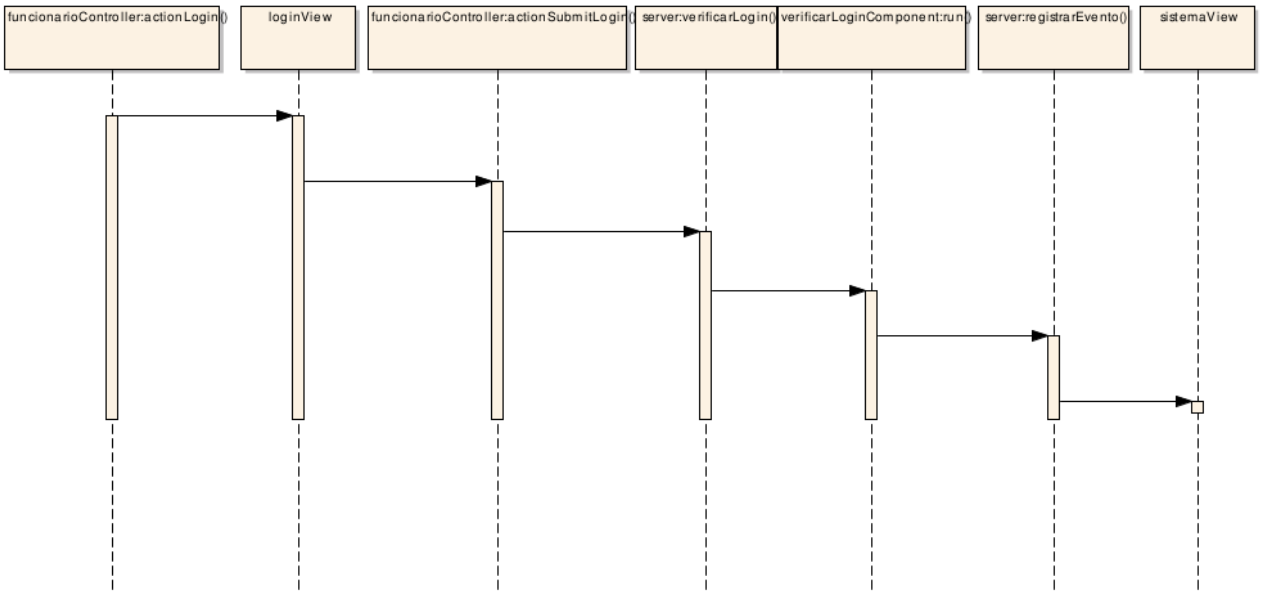


Figura 20 – Diagrama de sequência do caso fazer login

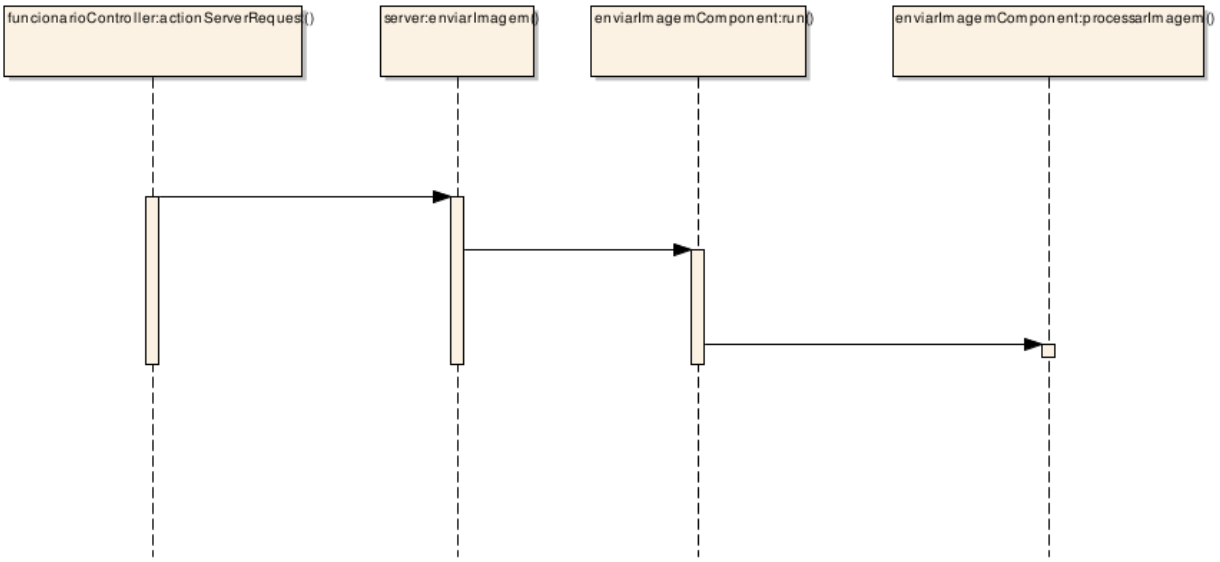


Figura 21 – Diagrama de sequência do caso enviar imagem

B.3 Diagrama de classes

O diagrama de classes demonstra a estrutura estática das classes de um sistema e suas relações que servem de modelo para os objetos. Através deste diagrama podemos mais uma vez perceber a interação dos elementos da estrutura MVC-Component adotada para esse sistema.

B.4 Diagrama entidade relação

Utilizamos a técnica de abordagem Entidade-Relacionamento que é considerada como um padrão para a modelagem conceitual. Essa técnica procura representar de forma abstrata os dados que serão armazenados no banco de dados tendo como base o conceito de que o mundo é formado por um conjunto de objetos chamados de entidades e seus relacionamentos entre si.

Este modelo de dados dará origem ao script SQL que utilizamos para construir a nossa estrutura de dados e para ilustrar esse modelo de dados vamos usar um modelo diagramático chamado Diagrama Entidade-Relacionamento.

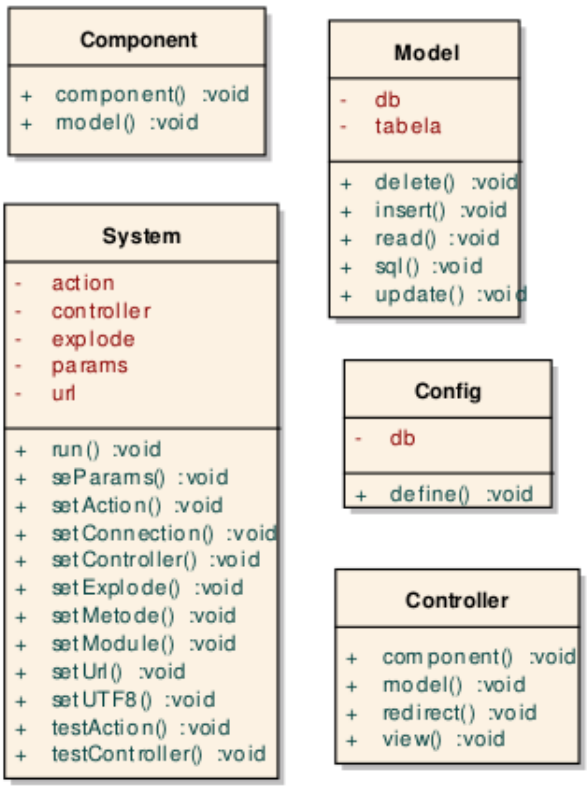


Figura 22 – Diagrama de classes da Framework

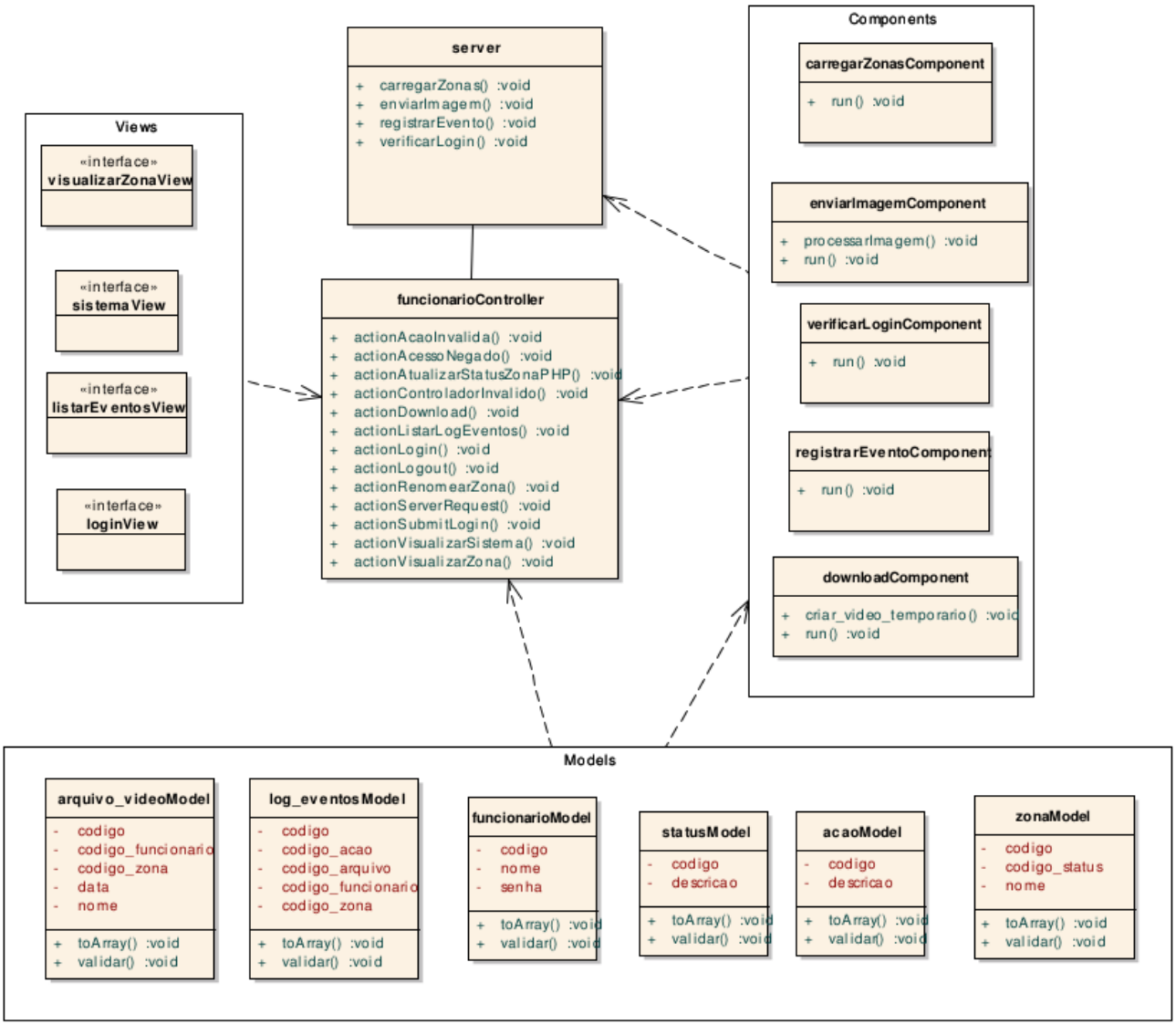


Figura 23 – Diagrama de classes da aplicação

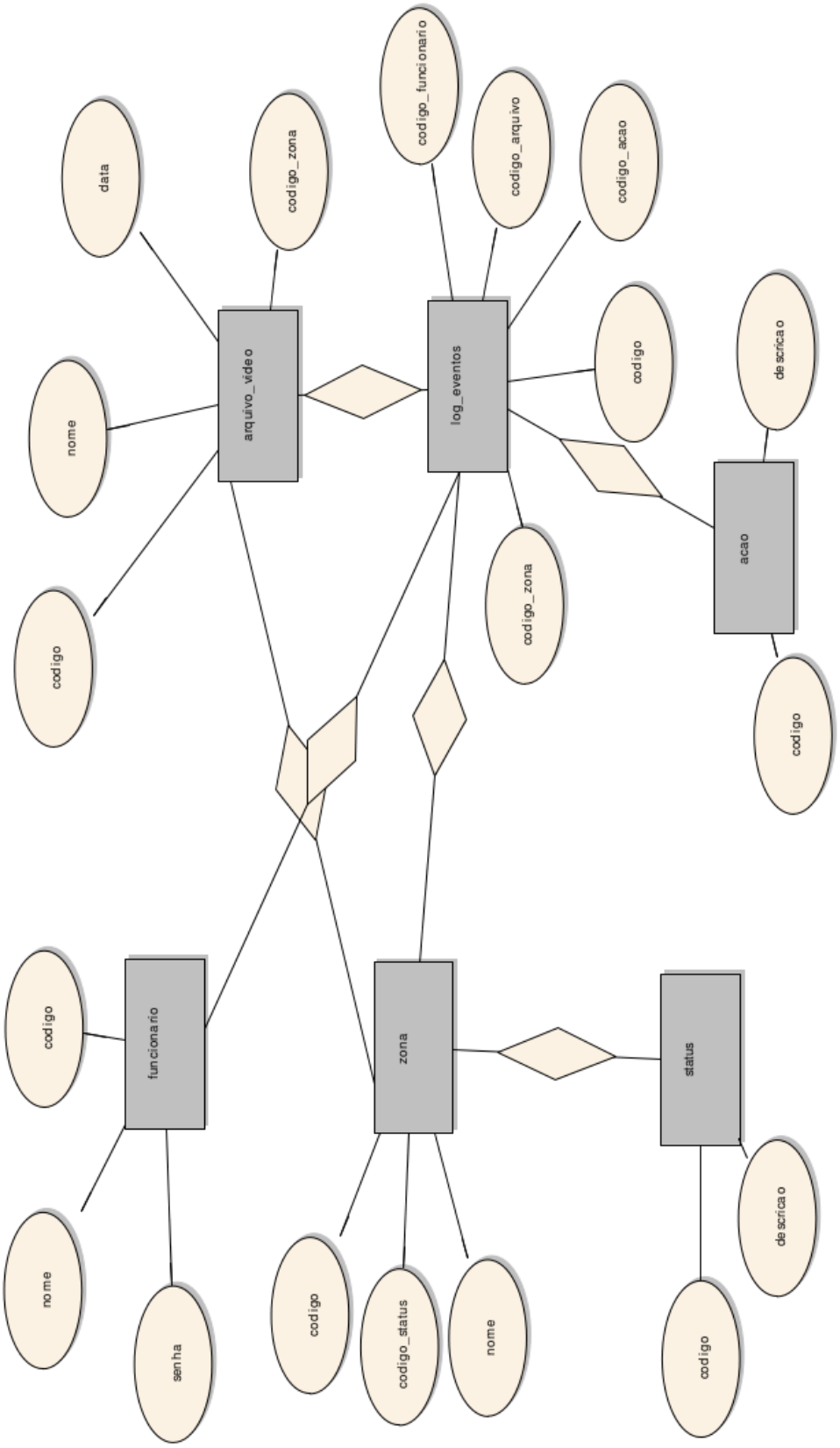


Figura 24 – Diagrama entidade relação

APÊNDICE C – Descrição do sistema PHP

C.0.1 Framework

Desenvolver software em PHP puro é uma tarefa cansativa e demorada, por isso com o objetivo de tornar o desenvolvimento dinâmico optamos por desenvolver um framework sob os moldes do padrão MVC. As características desse framework são:

- Orientado a objetos
- Utilizada o padrão MVC
- Faz tratamento de url
- Possui acesso ao banco de dados simplificado (create, read, update, delete)
- Conecta-se com vários bancos de dados simultaneamente

O framework possui os seguintes arquivos:

Config.php Esse arquivo é responsável por definir diretórios importantes como o diretório raiz da aplicação e o diretório de estilos utilizados. Ele também é responsável pela configuração de acesso (host,senha,dbname) ao banco de dados utilizado.

Component.php Esse arquivo contém a superclasse de todos os componentes da aplicação, ela contém funções que são comumente usadas em qualquer componente como as funções incluir model, component.

Controller.php Esse arquivo contém a superclasse de todos os controladores da aplicação, ela contém funções que são comumente usadas em qualquer controle como as funções incluir model, component, view e redirecionar páginas.

Model.php Esse arquivo contém a superclasse de todos os modelos da aplicação, ela contém funções que são comumente usadas em qualquer modelo como as funções de acesso ao banco de dados create, read, update e delete.

System.php Esse arquivo contém a classe que gerencia o sistema, ele é responsável tanto por identificar na url o controlador e a ação que devem ser chamados, como também organizar os parâmetros passados via url. Essa classe faz consistência dos dados testando se o controlador e a ação são válidos, seta a codificação como UTF8 e ainda instancia a conexão com os banco de dados definidos em Config.php.

C.0.2 Models

A partir dos diagramas de entidade relação são definidos os modelos que serão usados para acesso ao banco de dados, esses modelos possuem como variáveis as colunas de cada tabela e uma função validar() responsável por fazer a consistência dos dados antes de enviá-los ao banco. Todos herdam a classe Model.php e porisso as suas funções.

- arquivo_videoModel.php
- zonaModel.php
- log_eventosModel.php
- acaoModel.php
- statusModel.php
- funcionarioModel.php

C.0.3 Views

As views do sistema são a porta de comunicação com o usuário, a interface homem máquina. Através delas enviaremos a informação do estado do sistema ao funcionário da segurança e receberemos informação dele após a sua verificação. O sistema terá quatro views desenvolvidas com o auxílio do template Slate Admin (ADMIN, 2013). São elas:

- loginView.php (figura 25)

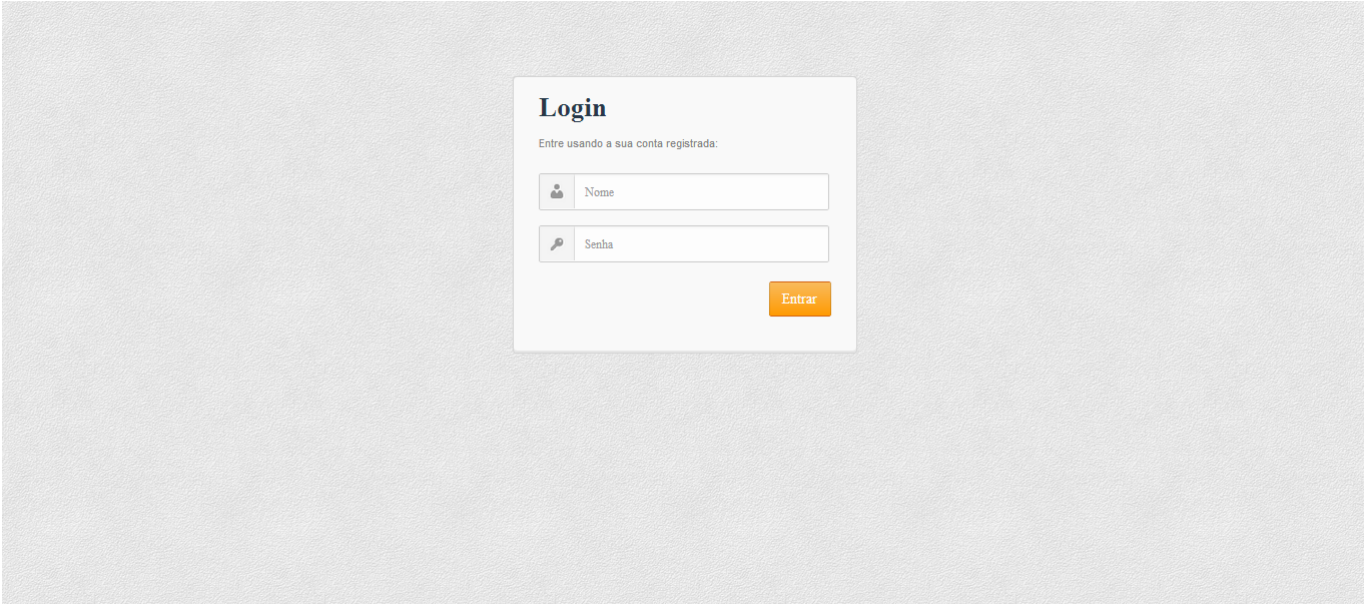


Figura 25 – View da página de Login

Figura 26 – View da página do sistema

- sistemaView.php (figura 26)
- visualizarZonaView.php
- listarEventosView.php

C.0.4 Controllers

O controlador do sistema é responsável por intermediar as operações do usuário com a lógica e consultas do banco de dados. Para esse sistema criamos o funcionarioController.php que possui as seguintes funções:

actionAtualizarStatusZonaPHP($params) Função que recebe um código de zona e um código de status e altera o status da zona no banco de dados.

actionVisualizarSistema($params) Função responsável por apresentar a view de visualização do sistema.

actionDownload($params) Função que recebe o código do arqui_video e inicia o download.

actionListarLogEventos($params) Função que busca a lista de eventos e chama a view responsável por sua exibição

actionLogin($params) Função que chama a view com o formulário de login.

actionLogout($params) Função que encerra a sessão e redireciona para a página de login.

actionRenomearZona($params) Função que recebe o código da zona e o novo nome então faz a alteração no banco de dados.

actionServerRequest($params) Função que faz as chamadas de servidor.

actionAcaoInvalida($params) Função que exibe o erro de ação inválida.

actionControladorInvalido($params) Função que exibe o erro de controlador inválido.

actionAcessoNegado($params) Função que exibe o erro de acesso negado.

actionSubmitLogin($params) Funçao que recebe as informações de login e verifica sua validade.

actionVisualizarZona($params) Função que recebe o código da zona, busca as informações da zona e chama a view de exibição.

C.0.5 Components

Os componentes são usados para isolar a lógica do controlador e deixá-lo somente com a função de direcionamento. Com esse fim foram criados:

class verificarLoginComponent Classe responsável por verificar as informações de login.

class registrarEventoComponent Classe responsável por registrar um evento.

class enviarImagemComponent Classe responsável por receber as imagens

class downloadComponent Classe responsável por executar a rotina de download.

class carregarZonasComponent Classe responsável por registrar todas as zonas que estão sendo monitoradas.

C.0.6 Server

O server é um arquivo que contém as funções que realizam os serviços do sistema. São elas:

carregarZonas($params) Função que recebe uma chamada ros e cadastra as zonas que estão sendo monitoradas no bando de dados.

enviarImagem($params) Função que recebe a imagem e salva no servidor.

registrarEvento($codigo_zona, $codigo_arquivo, $codigo_acao, $codigo_funcionario, $conexao) Função que registra o evento ocorrido.

verificarLogin($usuario, $senha) Função que recebe os dados de login e verifica a validade deles.

APÊNDICE D – Algoritmos para detecção de invasão

D.1 Comparação de frames subsequentes

Neste projeto utilizaremos um simples algoritmo que compara imagens subsequentes pixel a pixel. Se um pixel localizado na mesmo posição em imagens subsequentes é diferente, significa que há algo diferente na imagem, ou seja, algo se movimentando.

O problema deste método é que pequenas alterações e ruídos poderiam ser tomados como movimentos e consequentemente invasões. Por exemplo, a oscilação de luminosidade no ambiente altera levemente o pixel e, portanto, é considerado um pixel diferente. Para sanar este problema foi aplicado a todas as imagens recebidas pelo agente um filtro binário, que transforma todos os pixels em 1 ou 0. Desta maneira pequenos ruídos e variações de luminosidade teriam o mesmo valor e não influenciariam no resultado final. Além disso, para fazer a comparação das imagens subsequentes o processo torna-se apenas a aplicação do operador lógico XOR pixel a pixel.

A partir da imagem original da figura 27, foi obtido a partir deste algoritmo o resultado observado na figura 28.

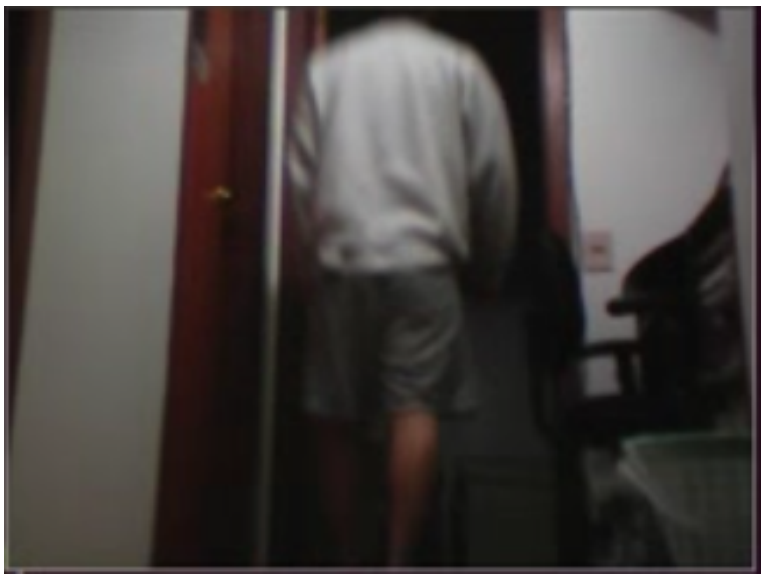


Figura 27 – Imagem original

Para analisar se houve ou não uma invasão observamos a quantidade de pixels diferentes em imagens subsequentes. Se existe mais de 1% de pixels diferentes é conside-

Figura 28 – Resultado da comparação de frames subsequentes a partir da imagem original da figura 27

rado que há um movimento estranho na imagem e um publisher deste agente envia uma mensagem de status avisando que houve a detecção de uma invasão nesta câmera.

D.2 Comparação com o primeiro frame

Outro algoritmo que poderia ser utilizado para a detecção de invasões poderia ser a comparação com o primeiro frame captado, ou seja, a imagem quando é sabido que não há movimentação na área e ela se encontra em um estado de segurança. Além disso, este algoritmo tem um ponto positivo, pois poderia ser utilizado para detectar objetos estranhos e imóveis na imagem.

Utilizando novamente a imagem original 27 é utilizado o algoritmo em questão e o resultado pode ser visualizado na figura 29.

O resultado obtido é, em alguns casos, melhor do que o obtido pela comparação de frames subsequentes. No entanto, este algoritmo é extremamente sensível, pois pequenas diferenças na imagem podem ser consideradas diferentes e detectadas como invasão apesar de não ser absolutamente nada. Um exemplo deste problema pode ser observado através da imagem original 30 e o resultado da comparação com o primeiro frame na figura 31.

Como pode ser observado, nesta figura é detectado uma diferença na imagem, que, na verdade, não existe. Esta diferença pode ser causado por pequenas diferenças de luminosidade e/ou outras condições do espaço. Desta maneira, este algoritmo não seria utilizável em um ambiente aberto, onde haveria diferença de luminosidade entre o dia e a noite, por exemplo.

Figura 29 – Resultado da comparação da imagem original da figura 27 com o primeiro frame

Figura 30 – Imagem original

D.3 Contornos de imagens

Uma solução melhor para detectar a invasão de um ambiente seriam a detecção de bordas na imagem eliminando o fundo da imagem. Com esta função seria fácil detectar movimento no ambiente. Existem funções prontas para isso no OpenCV, por exemplo, a função *findContours*. O resultado deste algoritmo pode ser visto nas figuras 32 e 33.

Os resultados obtidos nessas imagens foram obtidas no desenvolvimento do programa fora do ambiente *ROS*. No entanto, ao utilizar esta função dentro do ambiente *ROS* houve um problema de compatibilidade entre o *OpenCV* e o *ROS* e, por isso, esta

Figura 31 – Resultado da comparação da imagem original da figura 30 com o primeiro frame

Figura 32 – Primeiro resultado do algoritmo para encontrar os contornos de imagens

solução não pode ser utilizada.

Figura 33 – Segundo resultado do algoritmo para encontrar os contornos de imagens

APÊNDICE E – Documentação projeto ROS

My Project

Generated by Doxygen 1.7.6.1

Tue Nov 5 2013 00:00:35

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

Class Documentation

2.1 Communicator Class Reference

Public Member Functions

- Communicator ()
- ~Communicator ()
- void run ()

Protected Member Functions

- void fixPercentEncoding (std::string &str)
- int sendImage (std::string zone, std::string video, std::string frame, std::string image)
- std::string getImageb64 (std::string image_name)

2.1.1 Detailed Description

Agent Communicator analyzes if there is an jpg image saved in the images directory. If it does, the agent send the image to the server via an HTTP request

2.1.2 Constructor & Destructor Documentation

2.1.2.1 Communicator::Communicator ( ) [inline]

Constructor of class Communicator

2.1.2.2 Communicator::~Communicator ( ) [inline]

Destructor of class Communicator

2.1.3 Member Function Documentation

2.1.3.1 void Communicator::fixPercentEncoding (std::string & *str*) [inline, protected]

Replace special characters with percent encoding

Parameters

str	String to be fixed

2.1.3.2 std::string Communicator::getImageb64 (std::string *image_name*) [inline, protected]

Encode image to base64

Parameters

_image	Image to be encoded

Returns

Encoded string of the image

2.1.3.3 void Communicator::run ( ) [inline]

Run the agent communicator in order to analyze images in directory and send found images

2.1.3.4 int Communicator::sendImage (std::string *zone,* std::string *video,* std::string *frame,* std::string *image*) [inline, protected]

Send image encoded in base64 to the server

Parameters

zone	Number of the zone where the sensor is
video	Unique identifier from video
frame	Number of the frame this image correspond to
image	Image encoded in base64

Returns

State of the zone from the server

The documentation for this class was generated from the following file:

- communicator.cpp

2.2 Interpreter Class Reference

Public Member Functions

- Interpreter (int zone_number, int interpreter_number)
- ~Interpreter ()

Protected Member Functions

- void imageCallback (const sensor_msgs::ImageConstPtr &msg)

Protected Attributes

- ros::NodeHandle **nh_**
- image_transport::ImageTransport **it_**
- image_transport::Subscriber **image_sub_**
- ros::Publisher **status_pub_**
- cv::Mat **_mat**

2.2.1 Detailed Description

Agent Interpreter is responsible for getting the images feed from the corresponding sensor, analyzing it and sending a message feed containing a status, which is true if there is something strange in the image, f.i. an intruder, and false if everything is normal

2.2.2 Constructor & Destructor Documentation

2.2.2.1 Interpreter::Interpreter (int *zone_number,* int *interpreter_number*) [inline]

Construtor of class Interpreter

Parameters

zone_number	Number of the zone the interpreter will act on
interpreter_number	Number of the new interpreter

2.2.2.2 Interpreter::~Interpreter ( ) [inline]

Destructor of class Interpreter

2.2.3 Member Function Documentation

2.2.3.1 void Interpreter::imageCallback (const sensor_msgs::ImageConstPtr & *msg*) [inline, protected]

Callback function, that is called when the sensor sends an image

Parameters

msg	Image sent by the sensor

The documentation for this class was generated from the following file:

- my_subscriber.cpp

2.3 Monitor Class Reference

Public Member Functions

- Monitor (int zone_number, int interpreter_number, int sensor_number)
- ~Monitor ()

Protected Attributes

- ros::NodeHandle **nh_**
- ros::Subscriber **status_sub_** [10]
- image_transport::ImageTransport **it_**
- image_transport::Subscriber **image_sub_** [10]

2.3.1 Detailed Description

Class Monitor

Agent Monitor converges signals from all sensors and interpreters in a zone and creates one SensorAgent for each pair of agents sensor and interpreter

2.3.2 Constructor & Destructor Documentation

2.3.2.1 Monitor::Monitor (int *zone_number,* int *interpreter_number,* int *sensor_number*) [inline]

Construtor of class Monitor

Parameters

zone_number	Number of the zone the monitor will control
interpreter_number	Number of interpreters running in this zone
sensorNumber	Number of sensors running in this zone

2.3.2.2 Monitor::~Monitor ( ) [inline]

Destructor of class Monitor

The documentation for this class was generated from the following file:

- monitor.cpp

2.4 SensorMonitor Class Reference

Public Member Functions

- SensorMonitor (int _zone_number, int _sensor_number)
- ~SensorMonitor ()
- void imageCallback (const sensor_msgs::ImageConstPtr &msg)
- void statusCallback (const std_msgs::Bool status)

Protected Member Functions

- void updateState (int zone, int sensor, int state, std::string video)
- int getZoneState (int zone)
- std::string getVideoCode ()
- std::string fixDigits (int input, int number_digits)

Protected Attributes

- int **zone_number**
- int **sensor_number**
- int **zone_state**
- int **sensor_state**
- int **frame_count**
- std::string **video_code**

2.4.1 Detailed Description

Agent SensorMonitor groups signals from one sensor and its corresponding interpreter and responds to an invasion in a zone by saving the corresponding image to jpg file

2.4.2 Constructor & Destructor Documentation

2.4.2.1 SensorMonitor::SensorMonitor (int *_zone_number,* int *_sensor_number*) [inline]

Construtor of class SensorMonitor

Parameters

zone_number	Number of the zone the monitor is controlling
sensorNumber	Number of the sensor in this zone

2.4.2.2 SensorMonitor::~SensorMonitor ( ) [inline]

Destructor of class SensorMonitor

2.4.3 Member Function Documentation

2.4.3.1 std::string SensorMonitor::fixDigits (int *input,* int *number_digits*) [inline, protected]

Get integer with the wanted numbers of digits

Parameters

input	Integer to transform
number_digits	Desired number of digits

Returns

String containing integer with the number of digits desired

2.4.3.2 std::string SensorMonitor::getVideoCode ( ) [inline, protected]

Create an unique identifier for a video of a camera in a zone

Returns

Unique identifier

2.4.3.3 int SensorMonitor::getZoneState (int *zone*) [inline, protected]

Get the zone state from the server

Parameters

zone	Number of the zone where the sensor is

Returns

State of the zone from the server

2.4.3.4 void SensorMonitor::imageCallback (const sensor_msgs::ImageConstPtr & *msg*) [inline]

Callback function, that is called when the sensor sends an image

Parameters

msg	Image sent by the sensor

2.4.3.5 void SensorMonitor::statusCallback (const std_msgs::Bool *status*) [inline]

Callback function, that is called when the interpreter sends an status message

Parameters

status	Whether the interpreter identified someone in the image or not

2.4.3.6 void SensorMonitor::updateState (int *zone,* int *sensor,* int *state,* std::string *video*) [inline, protected]

Update state of the sensor in the server through an HTTP request

Parameters

zone	Number of the zone where the sensor is
sensor	Number of the sensor in this zone
state	State of the sensor
video	Unique identifier from video

The documentation for this class was generated from the following file:

- monitor.cpp

APÊNDICE F – Documentação projeto PHP

My Project

Generated by Doxygen 1.8.5

Thu Nov 7 2013 12:28:21

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Data Structure Index

2.1 Data Structures

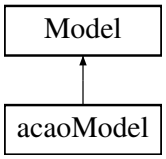
Here are the data structures with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 acaoModel Class Reference

Inheritance diagram for acaoModel:



Public Member Functions

- validar ()
- toArray ()

Data Fields

- **$_tabela** = "acao"
- **$_codigo**
- **$_descricao**

Additional Inherited Members

3.1.1 Detailed Description

Classe responsável pelo acesso a tabela "acao" do banco de dados

Definition at line 5 of file acaoModel.php.

3.1.2 Member Function Documentation

3.1.2.1 toArray ( )

Função que transforma as variáveis de um objeto tipo Model num array.

Definition at line 18 of file acaoModel.php.

3.1.2.2 validar ( )

Função que verifica se os dados estão prontos para ser inseridos no banco de dados

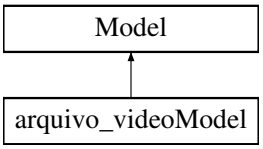
Definition at line 13 of file acaoModel.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/app/models/funcionario/acaoModel.php

3.2 arquivo_videoModel Class Reference

Inheritance diagram for arquivo_videoModel:



Public Member Functions

- validar ()
- toArray ()

Data Fields

- **$_tabela** = "arquivo_video"
- **$_codigo**
- **$_codigo_funcionario**
- **$_codigo_zona**
- **$_data**
- **$_nome**

Additional Inherited Members

3.2.1 Detailed Description

Classe responsável pelo acesso a tabela "aruivo_video" do banco de dados

Definition at line 5 of file arquivo_videoModel.php.

3.2.2 Member Function Documentation

3.2.2.1 toArray ( )

Função que transforma as variáveis de um objeto tipo Model num array.

Definition at line 21 of file arquivo_videoModel.php.

3.2.2.2 validar ( )

Função que verifica se os dados estão prontos para ser inseridos no banco de dados

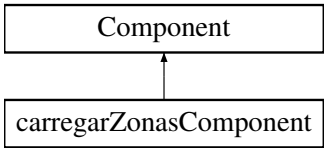
Definition at line 15 of file arquivo_videoModel.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/app/models/funcionario/arquivo_videoModel.php

3.3 carregarZonasComponent Class Reference

Inheritance diagram for carregarZonasComponent:



Public Member Functions

- __construct ()
- run ()

Additional Inherited Members

3.3.1 Detailed Description

Classe responsável por registrar todas as zonas que estão sendo monitoradas.

Definition at line 4 of file carregarZonasComponent.php.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 __construct ( )

Construtor da classe carregarZonasComponent

Definition at line 10 of file carregarZonasComponent.php.

3.3.3 Member Function Documentation

3.3.3.1 run ( )

Função que recebe o post com as informações da zona e as registra no banco de dados.

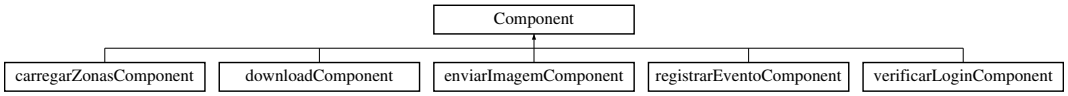
Definition at line 16 of file carregarZonasComponent.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/app/components/funcionario/carregarZonasComponent.php

3.4 Component Class Reference

Inheritance diagram for Component:



Protected Member Functions

- model ($nome)
- component ($nome)

3.4.1 Detailed Description

Classe base dos componentes

Definition at line 4 of file Component.php.

3.4.2 Member Function Documentation

3.4.2.1 component (*$nome*) [protected]

Função que inclui um componente para ser usado no componente

Parameters

$nome	nome do componente

Definition at line 14 of file Component.php.

3.4.2.2 model (*$nome*) [protected]

Função que inclui um model para ser usado no componente

Parameters

$nome	nome do model

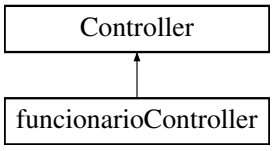
Definition at line 8 of file Component.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/system/Component.php

3.5 Controller Class Reference

Inheritance diagram for Controller:



Protected Member Functions

- view ($nome, $params=array(), $dados=array())
- model ($nome)
- component ($nome)
- redirect ($controller, $action, $params)

3.5.1 Detailed Description

Classe base dos controladores

Definition at line 4 of file Controller.php.

3.5.2 Member Function Documentation

3.5.2.1 component (*$nome*) [protected]

Função que inclui um componente para ser usado no controlador

Parameters

$nome	nome do componente

Definition at line 22 of file Controller.php.

3.5.2.2 model (*$nome*) [protected]

Função que inclui um model para ser usado no controlador

Parameters

$nome	nome do model

Definition at line 16 of file Controller.php.

3.5.2.3 redirect (*$controller, $action, $params*) [protected]

Função que redireciona para uma nova página.

Parameters

$controller	nome do controlador
$action	nome da ação
$params	parametros que serao passados via url

Definition at line 30 of file Controller.php.

3.5.2.4 view (*$nome, $params* = array(), *$dados* = array()) [protected]

Função que inclui o html de uma view

Parameters

$nome	nome da view

$params	conteúdo da url
$dados	informações a serem passadas para a view.

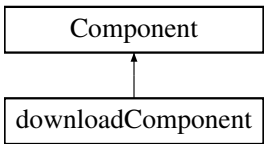
Definition at line 10 of file Controller.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/system/Controller.php

3.6 downloadComponent Class Reference

Inheritance diagram for downloadComponent:



Public Member Functions

- __construct ($codigo_arquivo)
- run ()

Additional Inherited Members

3.6.1 Detailed Description

Classe responsável por executar a rotina de download.

Definition at line 4 of file downloadComponent.php.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 __construct (*$codigo_arquivo*)

Construtor da classe downloadComponent

Parameters

$codigo_arquivo	código do arquivo que será baixado

Definition at line 12 of file downloadComponent.php.

3.6.3 Member Function Documentation

3.6.3.1 run ( )

Função que cria o video temporário e força o download do video.

See Also

criar_video_temporario($temp, $codigo_arquivo).

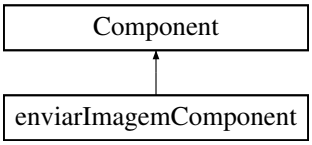
Definition at line 30 of file downloadComponent.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/app/components/funcionario/downloadComponent.php

3.7 enviarImagemComponent Class Reference

Inheritance diagram for enviarImagemComponent:



Public Member Functions

- __construct ()
- run ()

Additional Inherited Members

3.7.1 Detailed Description

Classe responsável por receber as imagens

Definition at line 4 of file enviarImagemComponent.php.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 __construct ( )

Construtor da classe enviarImagemComponent

Definition at line 10 of file enviarImagemComponent.php.

3.7.3 Member Function Documentation

3.7.3.1 run ( )

Função que recebe o post da imagem e a salva no servvidor

See Also

processarImagem($data, $arquivo, $frame)

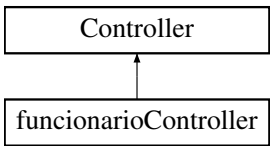
Definition at line 40 of file enviarImagemComponent.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/app/components/funcionario/enviarImagemComponent.php

3.8 funcionarioController Class Reference

Inheritance diagram for funcionarioController:



Public Member Functions

- __construct ()
- actionAtualizarStatusZonaPHP ($params)
- actionVisualizarSistema ($params)
- actionDownload ($params)
- actionListarLogEventos ($params)
- actionLogin ($params)
- actionLogout ($params)
- actionRenomearZona ($params)
- actionServerRequest ($params)
- actionAcaoInvalida ($params)
- actionControladorInvalido ($params)
- actionAcessoNegado ($params)
- actionSubmitLogin ($params)
- actionVisualizarZona ($params)

Additional Inherited Members

3.8.1 Detailed Description

Classe responsável por gerenciar as ações do funcionario e fazer chamadas de servidor

Definition at line 5 of file funcionarioController.php.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 __construct ( )

Construtor da classe funcionarioController

See Also

validaAcesso()

Definition at line 12 of file funcionarioController.php.

3.8.3 Member Function Documentation

3.8.3.1 actionAcaoInvalida (*$params*)

Função que exibe o erro de ação inválida.

Definition at line 259 of file funcionarioController.php.

3.8.3.2 actionAcessoNegado (*$params*)

Função que exibe o erro de acesso negado.

Definition at line 269 of file funcionarioController.php.

3.8.3.3 actionAtualizarStatusZonaPHP (*$params*)

Função que recebe um código de zona e um codigo de status e altera o status da zona no banco de dados.

Definition at line 104 of file funcionarioController.php.

3.8.3.4 actionControladorInvalido (*$params*)

Função que exibe o erro de controlador inválido.

Definition at line 264 of file funcionarioController.php.

3.8.3.5 actionDownload (*$params*)

Função que recebe o codigo do arquivo_video e inicia o download.

Definition at line 151 of file funcionarioController.php.

3.8.3.6 actionListarLogEventos (*$params*)

Função que busca a lista de eventos e chama a view responsável por sua exibição

Definition at line 163 of file funcionarioController.php.

3.8.3.7 actionLogin (*$params*)

Função que chama a view com o formulário de login.

Definition at line 196 of file funcionarioController.php.

3.8.3.8 actionLogout (*$params*)

Função que encerra a sessão e redireciona para a página de login.

Definition at line 201 of file funcionarioController.php.

3.8.3.9 actionRenomearZona (*$params*)

Função que recebe o codigo da zona e o novo nome então faz a alteração no banco de dados.

Definition at line 207 of file funcionarioController.php.

3.8.3.10 actionServerRequest (*$params*)

Função que faz as chamadas de servidor.

Definition at line 223 of file funcionarioController.php.

3.8.3.11 actionSubmitLogin (*$params*)

Funçao que recebe as informações de login e verifica sua validade.

Definition at line 274 of file funcionarioController.php.

3.8.3.12 actionVisualizarSistema (*$params*)

Função responsável por apresentar a view de visualização do sistema.

Definition at line 134 of file funcionarioController.php.

3.8.3.13 actionVisualizarZona (*$params*)

Função que recebe o codigo da zona, busca as informações da zona e chama a view de exibição.

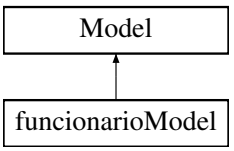
Definition at line 287 of file funcionarioController.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/app/controllers/funcionarioController.php

3.9 funcionarioModel Class Reference

Inheritance diagram for funcionarioModel:



Public Member Functions

- validar ()
- toArray ()

Data Fields

- **$_tabela** = "funcionario"
- **$_codigo**
- **$_nome**
- **$_senha**

Additional Inherited Members

3.9.1 Detailed Description

Classe responsável pelo acesso a tabela "funcionario" do banco de dados

Definition at line 6 of file funcionarioModel.php.

3.9.2 Member Function Documentation

3.9.2.1 toArray ( )

Função que transforma as variáveis de um objeto tipo Model num array.

Definition at line 20 of file funcionarioModel.php.

3.9.2.2 validar ( )

Função que verifica se os dados estão prontos para ser inseridos no banco de dados

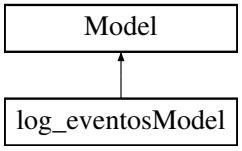
Definition at line 14 of file funcionarioModel.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/app/models/funcionario/funcionarioModel.php

3.10 log_eventosModel Class Reference

Inheritance diagram for log_eventosModel:



Public Member Functions

- validar ()
- toArray ()

Data Fields

- **$_tabela** = "log_eventos"
- **$_codigo**
- **$_codigo_acao**
- **$_codigo_arquivo**
- **$_codigo_funcionario**
- **$_codigo_zona**

Additional Inherited Members

3.10.1 Detailed Description

Classe responsável pelo acesso a tabela "log_eventos" do banco de dados

Definition at line 5 of file log_eventosModel.php.

3.10.2 Member Function Documentation

3.10.2.1 toArray ( )

Função que transforma as variáveis de um objeto tipo Model num array.

Definition at line 21 of file log_eventosModel.php.

3.10.2.2 validar ( )

Função que verifica se os dados estão prontos para ser inseridos no banco de dados

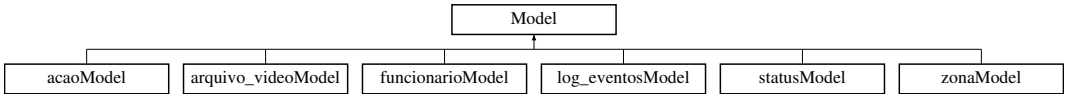
Definition at line 15 of file log_eventosModel.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/app/models/funcionario/log_eventosModel.php

3.11 Model Class Reference

Inheritance diagram for Model:



Public Member Functions

- __construct ($connection)
- insert (array $dados)
- read ($where=null, $last=null)
- update (array $dados, $where)
- delete ($where=null)
- sql ($sql)

Data Fields

- **$_tabela**

Protected Attributes

- **$_db**

3.11.1 Detailed Description

Classe base dos models responsável pela comunicação com o banco de dados.

Definition at line 4 of file Model.php.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 __construct (*$connection*)

Construtor da classe Model $param $connection conexão com o banco de dados

Definition at line 10 of file Model.php.

3.11.3 Member Function Documentation

3.11.3.1 delete (*$where* = `null`)

Função que realiza a query DELETE

Parameters

$where	condição para a exclusão

Definition at line 51 of file Model.php.

3.11.3.2 insert (array *$dados*)

Função que realiza a query INSERT

Parameters

$dados	dados que serão inseridos no banco de dados.

Definition at line 16 of file Model.php.

3.11.3.3 read (*$where* = `null`, *$last* = `null`)

Função que realiza a query SELECT

Parameters

$where	condição de busca.

Definition at line 25 of file Model.php.

3.11.3.4 sql (*$sql*)

Função que executa uma query qualquer.

Parameters

$sql	código sqp da query que será executada.

Definition at line 59 of file Model.php.

3.11.3.5 update (array *$dados*, *$where*)

Função que realiza a query UPDATE

Parameters

$dados	dados que serão atualizados
$where	condição para a atualização

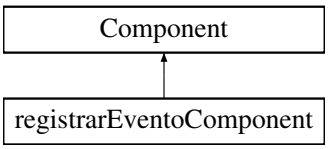
Definition at line 40 of file Model.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/system/Model.php

3.12 registrarEventoComponent Class Reference

Inheritance diagram for registrarEventoComponent:



Public Member Functions

- __construct ($codigo_zona, $codigo_arquivo, $codigo_acao, $codigo_funcionario, $conexao)
- run ()

Additional Inherited Members

3.12.1 Detailed Description

Classe responsável por registrar um evento.

Definition at line 4 of file registrarEventoComponent.php.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 __construct (*$codigo_zona, $codigo_arquivo, $codigo_acao, $codigo_funcionario, $conexao*)

Construtor da classe registrarEventoComponent

Parameters

$codigo_zona	codigo da zona que sofreu o evento
$codigo_arquivo	codigo do arquivo que foi criado, pode ser null
$codigo_acao	codigo da ação realizada
$codigo_-funcionario	codigo do funcionario que disparou o evento
$conexao	conexão com o banco de dados

Definition at line 19 of file registrarEventoComponent.php.

3.12.3 Member Function Documentation

3.12.3.1 run ( )

Função responsável por cadastra o evento

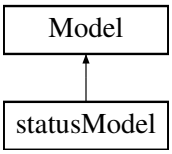
Definition at line 28 of file registrarEventoComponent.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/app/components/funcionario/registrarEventoComponent.php

3.13 statusModel Class Reference

Inheritance diagram for statusModel:



Public Member Functions

- validar ()
- toArray ()

Data Fields

- **$_tabela** = "status"
- **$_codigo**
- **$_descricao**

Additional Inherited Members

3.13.1 Detailed Description

Classe responsável pelo acesso a tabela "status" do banco de dados

Definition at line 5 of file statusModel.php.

3.13.2 Member Function Documentation

3.13.2.1 toArray ( )

Função que transforma as variáveis de um objeto tipo Model num array.

Definition at line 18 of file statusModel.php.

3.13.2.2 validar ( )

Função que verifica se os dados estão prontos para ser inseridos no banco de dados

Definition at line 12 of file statusModel.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/app/models/funcionario/statusModel.php

3.14 System Class Reference

Public Member Functions

- __construct ()
- run ()

3.14.1 Detailed Description

Classe responsável pela execução do sistema.

Definition at line 4 of file System.php.

3.14.2 Constructor & Destructor Documentation

3.14.2.1 __construct ( )

Construtor da classe System

Definition at line 13 of file System.php.

3.14.3 Member Function Documentation

3.14.3.1 run ( )

Função que executa a ação do controlador

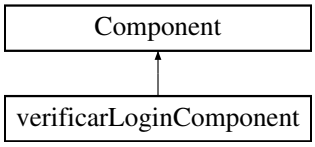
Definition at line 115 of file System.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/system/System.php

3.15 verificarLoginComponent Class Reference

Inheritance diagram for verificarLoginComponent:



Public Member Functions

- __construct ($usuario, $senha)
- run ()

Additional Inherited Members

3.15.1 Detailed Description

Classe responsável por verificar as informações de login.

Definition at line 4 of file verificarLoginComponent.php.

3.15.2 Constructor & Destructor Documentation

3.15.2.1 __construct (*$usuario, $senha*)

Construtor da classe verificarLoginComponent

Parameters

$usuario	usuario cadastrado
$senha	senha cadastrada

Definition at line 13 of file verificarLoginComponent.php.

3.15.3 Member Function Documentation

3.15.3.1 run ( )

Função que verifica a validade dos dados de login

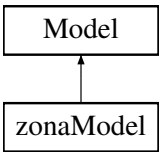
Definition at line 19 of file verificarLoginComponent.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/app/components/funcionario/verificarLoginComponent.php

3.16 zonaModel Class Reference

Inheritance diagram for zonaModel:



Public Member Functions

- validar ()
- toArray ()

Data Fields

- **$_tabela** = "zona"
- **$_codigo**
- **$_codigo_status**
- **$_nome**

Additional Inherited Members

3.16.1 Detailed Description

Classe responsável pelo acesso a tabela "zona" do banco de dados

Definition at line 5 of file zonaModel.php.

3.16.2 Member Function Documentation

3.16.2.1 toArray ( )

Função que transforma as variáveis de um objeto tipo Model num array.

Definition at line 19 of file zonaModel.php.

3.16.2.2 validar ( )

Função que verifica se os dados estão prontos para ser inseridos no banco de dados

Definition at line 13 of file zonaModel.php.

The documentation for this class was generated from the following file:

- C:/xampp/htdocs/sistema_de_seguranca/app/models/funcionario/zonaModel.php

Index

APÊNDICE G – Código fonte de geração do Banco de Dados

```
-- phpMyAdmin SQL Dump
-- version 3.5.2.2
-- http://www.phpmyadmin.net
--
-- Servidor: 127.0.0.1
-- Tempo de Geracao:
-- Versao do Servidor: 5.5.27
-- Versao do PHP: 5.4.7

SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

--
-- Banco de Dados: `sistema_seguranca`
--

-- --------------------------------------------------------

--
-- Estrutura da tabela `acao`
--

CREATE TABLE IF NOT EXISTS `acao` (
  `codigo` int(11) NOT NULL AUTO_INCREMENT,
  `descricao` varchar(45) NOT NULL,
  PRIMARY KEY (`codigo`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=6 ;
```

```
--
-- Extraindo dados da tabela `acao`
--

INSERT INTO `acao` (`codigo`, `descricao`) VALUES
(1, 'Marcar como seguro'),
(2, 'Marcar como alerta'),
(3, 'Marcar como invasao'),
(4, 'Renomear Zona'),
(5, 'Inserir Zona');

-- --------------------------------------------------------

--
-- Estrutura da tabela `arquivo_video`
--

CREATE TABLE IF NOT EXISTS `arquivo_video` (
  `codigo` varchar(24) NOT NULL,
  `nome` varchar(45) NOT NULL,
  `data` datetime NOT NULL,
  `codigo_zona` int(11) NOT NULL,
  PRIMARY KEY (`codigo`),
  KEY `indice_codigo_zona` (`codigo_zona`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Estrutura da tabela `funcionario`
--

CREATE TABLE IF NOT EXISTS `funcionario` (
  `codigo` int(11) NOT NULL AUTO_INCREMENT,
  `senha` varchar(45) NOT NULL,
  `nome` varchar(45) NOT NULL,
  PRIMARY KEY (`codigo`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=2 ;
```

```
--
-- Extraindo dados da tabela `funcionario`
--

INSERT INTO `funcionario` (`codigo`, `senha`, `nome`) VALUES
(111000111, 'naopodelogarcomoserver', 'Server'),
(1, '21232f297a57a5a743894a0e4a801fc3', 'admin');

-- --------------------------------------------------------

--
-- Estrutura da tabela `log_eventos`
--

CREATE TABLE IF NOT EXISTS `log_eventos` (
  `codigo` int(11) NOT NULL AUTO_INCREMENT,
  `codigo_zona` int(11) NOT NULL,
  `codigo_funcionario` int(11) NOT NULL,
  `codigo_arquivo` varchar(24) DEFAULT NULL,
  `codigo_acao` int(11) NOT NULL,
  PRIMARY KEY (`codigo`),
  KEY `indice_codigo_acao` (`codigo_acao`),
  KEY `indice_codigo_zona` (`codigo_zona`),
  KEY `indice_codigo_funcionario` (`codigo_funcionario`),
  KEY `indice_codigo_arquivo` (`codigo_arquivo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

-- --------------------------------------------------------

--
-- Estrutura da tabela `status`
--

CREATE TABLE IF NOT EXISTS `status` (
  `codigo` int(11) NOT NULL AUTO_INCREMENT,
  `descricao` varchar(45) NOT NULL,
  PRIMARY KEY (`codigo`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=4 ;
```

```
--
-- Extraindo dados da tabela `status`
--

INSERT INTO `status` (`codigo`, `descricao`) VALUES
(1, 'seguro'),
(2, 'alerta'),
(3, 'invasao');

-- --------------------------------------------------------


--
-- Estrutura da tabela `zona`
--

CREATE TABLE IF NOT EXISTS `zona` (
  `codigo` int(11) NOT NULL AUTO_INCREMENT,
  `codigo_status` int(11) NOT NULL,
  `nome` varchar(45) NOT NULL,
  PRIMARY KEY (`codigo`),
  KEY `indice_codigo_status` (`codigo_status`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

--
-- Restricoes para as tabelas dumpadas
--


--
-- Restricoes para a tabela `arquivo_video`
--
ALTER TABLE `arquivo_video`
  ADD CONSTRAINT `arquivo_video_ibfk_1` FOREIGN KEY (`codigo_zona`) REFERENCE

--
-- Restricoes para a tabela `log_eventos`
--
ALTER TABLE `log_eventos`
  ADD CONSTRAINT `log_eventos_ibfk_1` FOREIGN KEY (`codigo_zona`) REFERENCES
```

```
  ADD CONSTRAINT `log_eventos_ibfk_4` FOREIGN KEY (`codigo_acao`) REFEREN
  ADD CONSTRAINT `log_eventos_ibfk_5` FOREIGN KEY (`codigo_funcionario`)
  ADD CONSTRAINT `log_eventos_ibfk_6` FOREIGN KEY (`codigo_arquivo`) REFE

--
-- Restricoes para a tabela `zona`
--
ALTER TABLE `zona`
  ADD CONSTRAINT `zona_ibfk_1` FOREIGN KEY (`codigo_status`) REFERENCES `

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```