Willburg Outdoor PTY(ltd.)

Smart Image Identifier

Version 1.0

Software Requirements Specifications and Technology Neutral Process Design

Project Team:
Stephen Swanepoel u11032091
Dian Veldsman u12081095

Contents

Software Requirements Specification

1 Introduction

This document aims to set out the functional and non-functional requirements of a system as specified by the Willburg Outdoor PTY(ltd.). The system is required to assess an image by sorting the image into baskets and groupings and then identify if a human is present in the picture. The document will also serve the purpose of communicating the requirements and specifications as needed by the client.

1.0.1 Vision

The client for this project, Willburg Outdoor PTY(ltd.), has called for the design of an application that will assess an images, identify a human(s) in the image and send the result back as a response to the server based on the outcome of the image which was processed.

1.0.2 Background

Willburg Outdoor is company that is passionate about South Africa. With this passion comes the need to protect its farmers and it animals from those who intend to harm them. The client intends to use the system to assist in fighting the following problems: animal poaching and farm attacks. Willburg provides its' clients with a camera system which is interfaced with a dashboard as well as a mobile application. The cameras snap pictures when movement is detected and sends them to the Willburg server. The images are then pushed to the respective users. Willburg intend on providing its' users with real-time push notifications when a human has been detected in an image captured by a camera on their property. Smart Image Identifier will play an influential role in the prevention of both farm attacks as well as poaching.

2 Software architecture overview

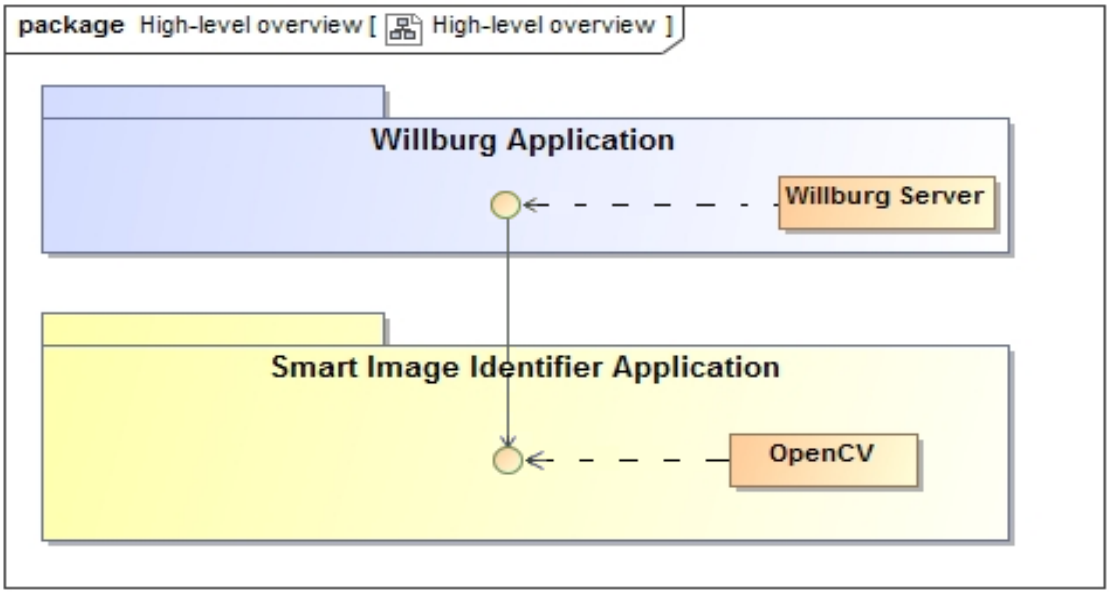

Figure 1: A high-level overview of the software architecture of the Smart Image Identifier

3 Overall Software architecture

3.1 Architecture Requirements

3.1.1 Access and Integration Requirements

Any and all interactions by the user will be done through the Willburg Application - this is an application previously designed by the project owner for mobile platforms. All user interactions fall outside of our project scope.

3.1.2 Quality Requirements

3.1.2.1 Critical

3.1.2.1.1 Performance requirements and tactics

Requirements

- Images which are processed are resized to improve detection as well as the speed at which they are processed by the server.
- The system has been created with the most minimal and efficient coding practices possible, given that the result must still be reliable and robust.
- The system needs to perform the processing of image as well as the identification of humans in under 2 seconds.

Tactics

1. Thread pooling occurs when a request is received. If a thread is available, it is given to the request for handling. A queue structure is used to handle the thread pooling such that it will add a thread to the pool once the request is complete and pull a thread when there is an incoming request.
2. Threads are used to run multiple requests to inorder to spread the load accross time.

3.1.2.1.2 Scalability requirements and tactics

Requirements

- Modular programming has be used in order to ensure that there are no restrictions in terms of the systems ability to be extended and improved upon at later stages. It also ensures easy testing of core functionality.
- The server should be able to process multiple images at once to ensure the real-time restriction is met for all clients of Willburg.
- The system should be able to perform correctly even while under strain due to bulk processing without any loss of data and ensuring the real-time restriction is met.

Tactics

1. As stated in the previous section threads and thread pooling will be used to handle multiple request within the shortest amount possible. This will ensure that system is able to handle an increase in load without impact on the performance of the system.

2. Queueing is used to handle requests when the system is in high demand to ensure that no request is lost.

3.1.2.1.3 Integratibility requirements and tactics

Requirements

- The system will intergrate with the exisiting Willburg architecture and

Tactics

1. The system is designed to support pluggable adapters to allow remote services to use to use the system.

3.1.2.1.4 Auditability

Requirements

1. All images processed will be logged and traceable(user, date, time etc.) once pushed into the database.

Tactics

1. Auditability will be implemented within the system via a log which will contain all transactions processed by the system. The application server will support specifying logging logic (advices) as crosscutting concerns within aspects or at least provide the ability to apply logging via interception.
2. Interception is used for auditability to ensure the system contains a full log of all events and details about which client launched the event. This will be useful in ensuring a response may linked to the respective client.

3.1.2.2 Nice-to-have

3.1.2.2.1 Flexibility requirements

Requirements

- The system should be able to perform correctly even while under strain due to bulk processing without any loss of data.
- The system should be modularized to ensure both easy testing of components as well as addition of components without affecting the system.

3.1.2.2.2 Maintainability requirements

Requirements

- The system should be designed in technologies which offer user support and continuous updates for extra functionality.
- The system must be well structured and documented to ensure future developers are able to understand and maintain the system.
- The modular design of the system must be such that if changes must be made to a part of the system, only that part itself should be changed.

3.1.2.2.3 Reliability requirements

Requirements

- The system must be thoroughly tested on the server side, to ensure it will not cause faults or problems.
- The notification warning users will receive must be sent out immediately after the image has been processed in a real-time manner.

3.1.3 Architectural constraints

3.1.3.1 Architeture Constraint

1. Higher level layers can interact with layers below but lower levels should never interact with levels above.

3.1.3.2 Design Constraint

1. All application software will be modularized into classes using object-oriented design principles.

3.1.3.3 Implementation Constraint

1. All software components of the application shall be programmed in Java programming language.

3.2 Architecture design

This section specifies the very high-level software architecture design, i.e. the software architecture design for the first level of granularity. It includes the allocation of architectural responsibilities to architectural components, any tactics which should be used at the current level of granularity to address quality requirements

3.2.1 Infrastructure

The layered architectural pattern provides an infrustructure which limits access of components within one layer to components which are either in the same or the next lower level layer. One of the strengths is that a layer can be easily replaced. In particular, one can add further access channels without changing any lower level layers within the software system.

4 Image Processing System Architecture

This section specifies the requirements and design of the software architecture second level of granularity. The image processing service is responsible for the manipulation of an image and the detection of a human.

4.1 Image processing

This section specifies the the software architecture requirements and design for the image processing application.

4.1.1 Software architecture requirements

The architectural requirements include the refined quality requirements and the architectural responsibilities. The architectural constraints for this lower level component are the same as for the system as a whole.

4.1.1.1 Access and Integration Requirements

4.1.1.1.1 Access Channel Requirements The image processing application will address the first interaction between the existing server and the application. The image processing application wil alsol address the manipulation of images and identification of humans.

4.1.1.1.2 Integration Channel Requirements The image processing application must integrate with the image procsessing system in order for images to be processed.

4.1.1.2 Architectural responsiblities

The architectural responsibilities of the application server

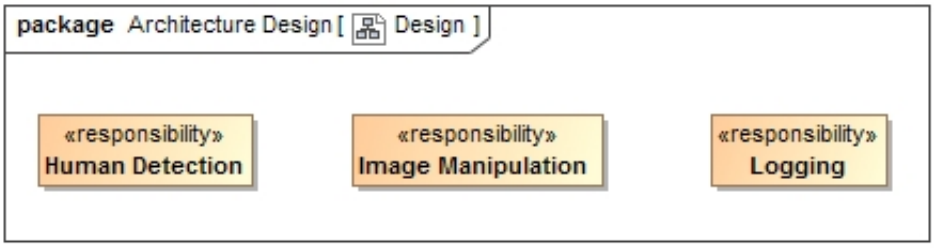

Figure 2: The architectural responsibilities of the image processing application

4.1.2 Architecture design

4.1.2.1 Architectural components

The architectural responsibility allocation the application server

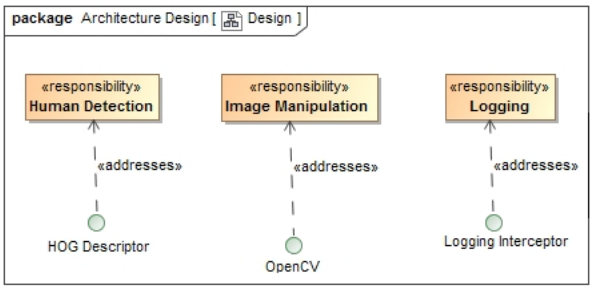

Figure 3: The abstract components to which the image processing applicatio

4.1.2.2 Application component concepts and constraints

The central concept which will be used to specify application logic (the business processes) are the concepts of:

- a *service contract* which encodes the requirements for a service
- a *service* which encodes the concrete implementation of a service.

These services which encapsulate the processes will be constrained to be stateless, i.e. no state is maintained across service requests. Long living state is maintained in domain objects which are typically persisted and which should not have any business logic.

4.1.2.3 Frameworks and technologies

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. The library is cross-platform and free for use under the open-source BSD license.

4.1.2.3.1 Concrete realization of architectural components

4.1.2.3.1.1 Concrete realization of human detection

OpenCV histogram of oriented gradients (HOG) which is a feature descriptor used to detect objects in computer vision and image processing. The HOG descriptor technique counts occurrences of gradient orientation in localized portions of an image - detection window, or region of interest (ROI). OpenCV offers a default implementation in conjunction HOG descriptor to detect humans called `people_detection`.

4.1.2.3.1.2 Concrete realization of image manipulation

OpenCV has implemented functions that allow image to be manipulated. Such as enlarging the image or grayscaling it.