Software Requirement Specification

AutoMart
dot slash

#YOCO
You Only Code Once

Schnetler Robin	Sinovcic Marko	Tanna Yashu
12220702	10693034	12076164
robin.schnetler@gmail.com	markosinovcic@gmail.com	tanna.yashu@gmail.com

August 21, 2014

1 Introduction

This document is the Software Requirement Specification (SRS) for the AutoMart program that needs to be developed. The company responsible for the project is Dot Slash. The project must be implemented by #YOCO.

2 Vision

dot slash(TM) currently has a website in place for AutoMart for sale of used and new vehicles, including motorcycles and trucks. Users place their advertisements by entering textual data along with the vehicle images on the website. The system currently has a AI textual parser that takes the data and extrapolates relevant vehicle properties to create a valid advertisement. The advertisement consists of textual data representing vehicle details as well as images that contain vehicle being advertised. This data is entered into the existing AutoMart's database.

AutoMart currently faces spamming, poor image quality and misrepresentation problems. Spamming is caused by bots that upload irrelevant images to their system, causing many invalid advertisements. Currently there is no image inspection, which allows users to upload images of inferior quality, which may not result in an invalid advertisement but rather a bad one. Finally, they have a AI text parser which parser the textual data and generates relevant vehicle details which are used to represent the advertisement in the database. However they are facing a problem where the AI classifies the text data different from the image uploaded along with that data. This resulting in advertisement representing two different vehicles.

The aim is to solve this by implementing a subsystem that will prevent their system from allowing users and bots from uploading inferior quality and irrelevant images to their system. The system also needs to ensure that the textual data and the image correspond to each other and in fact represent the same vehicle. Since they want to ensure the data matches the vehicle in the image, they wish to have certain vehicle details extrapolated from the image instead of extrapolating it from the text, which will reduce room for incorrect advertisements.

3 Architecture Requirements

3.1 Access Channel Requirements

The system dealing with evaluating the advert, images specifically, needs to be implemented in the back end of the website. Therefore there is no graphical user interface that the user will interact with. The user will only get messages back from the system stating the success or failure of the advert to be uploaded. The messages will be displayed in the browser which will be implemented by dot slash.

The training part for car recognition will have a graphical user interface that will guide the user in training the classifiers with the new data to ensure that the system stays up to date with the latest cars on the market. This part of the system will be accessible through a computer running on Windows OS.

3.2 Quality Requirements

3.2.1 Performance

The system must be used in real time when a user creates and advert for a vehicle that they wish to sell. The system must therefore process and classify the image in a maximum of 2 seconds per feature(car, make, model, colour) and a further 2 seconds to generate a rating for the image.

The detection of one feature, through experience has not taken more than 2 seconds. We have achieved this through OpenCVś object recognition framework. All features to be detected will use the exact same framework. hence we can infer this time constraint.

3.2.2 Reliability

The system will classify the feature at least 8 out of 10 times.

The openCV framework suggests usage of atleast 5000 images when classifying faces. This has worked and historical data proves that this had been effective. Our system will be using 5000 images for each characteristic.

3.2.3 Flexibility

The system needs to be trained for each classification. One classification cannot work for another feature detection use case. But the inability of classification of one feature will not result in a system failure.

This will be achieved using the rating of an image based on the features that the system was, in fact, able to detect.

3.2.4 Maintainability

The addition of new feature classifiers will be pluggable at a later stage.

We are using SOLID design principles and have implemented a strategy design pattern for each of the feature classifiers. At a later stage, if a new feature is required, it will merely have to be trained and plugged into the current strategy framework.

It is also essential that the information we obtain from the image is accurate since the basis of AutoMartś business operations will rely on how well we implement our algorithm and how accurate it is.

3.3 Tradeoffs

In order to achieve our processing speeds of 2 seconds per feature in the image as well as a further 2 seconds to generate a rating for the image, we will need to use a lot of memory for buffering so that a lot of data is immediately at hand. This requires a large amount of memory as the images we are working with will require sizeable chunks of RAM. We will also use a lot of processing power since there is a lot of processing that happens on the image. Thus to achieve these speeds we will have to sacrifice memory and processing power.

To ensure our system correctly classifies features 80% of the time will need a lot of training for the system to achieve this level of accuracy. Thus this will take time, hence to achieve accuracy we will need to sacrifice a lot of time to train our program. As mentioned, we will need approximately 5000 images for every feature that we want to train. Take into consideration how many features we aim to correctly classify, as well as all the man-power it takes to prepare images for this training (cropping and sorting followed by the actual training being the most time consuming). This training process is also very memory intensive as the training process has been adapted to use 4GB of memory. We also need to consider the processing power required for the training.

Lastly, we aimed to achieve flexibility by ensuring our system can be trained for each classification. This again requires a lot of training which, as above, is very time consuming and memory intensive above needing string processing power.

3.4 Integration Requirements

The final product needs to integrate with current AutoMart system. This is to be achieved by implementing an API through which AutoMarts website will be able to access the functions needed to determine adverts validity.

3.5 Architecture Constraints

The system needs to be implemented in C#. The system also needs to be able to communicate with the Microsoft SQL Server.

4 Functional Requirements

4.1 Introduction

4.1.1 Scope

The subsystem should be implemented with in the currents AutoMart system and should not replace it. The subsystem needs to provide features that will help AutoMart reduce the number of poor and irrelevant advertisements on their website.

The subsystem is to be implemented in the front end of the system. When the user submits the their advert on the website, the subsystem needs to take the adverts images and evaluate them in order to decide whether the advert wished to be uploaded by the user should be allowed to be entered into the systems database.

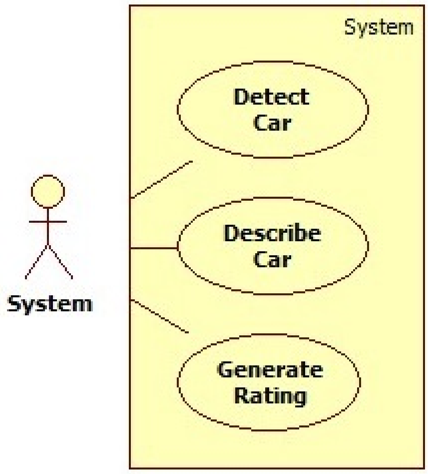
The subsystem performs four sequential tests on the adverts images. For each image uploaded a image object will be created and used to store image information. The first test checks if the image uploaded contains a car. If the system detects a car in the image, the test is seen as successful and goes on to the next test. If the system does not detect a car in the image, further image tests come to a halt. A relevant message will be stored in the image object and the system proceeds onto the next image. The second test determines the colour of the car. AutoMart currently has seventeen colour buckets that used to describe the colour of the car. The system needs to calculate which colour bucket is the closest to the cars colour. After the colour bucket has been determined, it stores the colour into the image object. Further testing can proceed even if the system was unable to determine the colour. Third test tries to determine the make of the car. The system uses textual data to help it determine the make, as determining the make blindly will be to time consuming. The detected make is stored in the image object and proceeds onto the final test. If the system was unable to detect a make of the car, the testing comes to a halt. Final test tries to determine the model of the car. It uses previous test information to help it determine the model of the car. If the model is determined the information is stored into the image object and proceeds onto the next image.

After all the images have been put through the tests, the image objects representing the images are put through a final set of tests to ensure that there is no information conflict. The system first checks if all the objects have the same colour. If there is conflict it tries to resolve it by checking how many object have the same colour and how many don't. The information is also compared against the textual data. If the colour differs between the objects or it differs from the textual data the system will conclude that the images are representing different cars. This would mean that the advertisement is invalid and further testing will halt, and the advert will not be allowed to be uploaded to the system. The second test compares all the make information. If conflict is encountered the same principles as with in the colour conflict will be applied. Finally, the last test compares all the model information. The model testing will be more lenient than the first two tests. If the models differ in most of the objects and also differ from textual data, the textual data will be used to state the model of the car.

After all the test have been passed or halted unexpectedly, the system will provide relevant error messages or informative messages that were encountered during the image testing. The user will have a chance to fix any of the errors a system encountered. If the test were successful initially or after the user has fixed the errors the advert will be allowed to be uploaded to the system.

Because the system is working with ever changing product, the system needs to accommodate for learning new cars. This way the system will never become unusable at a later stage as it will be able to stay updated with new cars.

Figure 1: High Level System Use Case Diagram.



4.1.2 Limitations

The system may be limited on model recognition, as sometimes it is not possible to distinguish the model of the car by only face value properties that are given by an image. Therefore the system may not be able to determine the model of the car correctly every time.

4.1.3 Exclusions

The system should only be implemented to work with cars and should not consider for the existence of any other motor vehicles such as motorcycles and trucks.

4.2 Required Functionality

Figure 2: Detect Car Low Level Use Case Diagram.

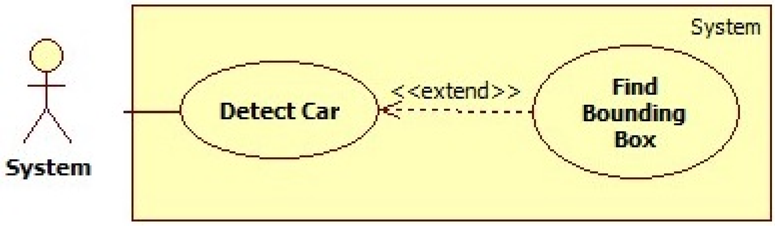


Figure 3: Describe Car Low Level Use Case Diagram.

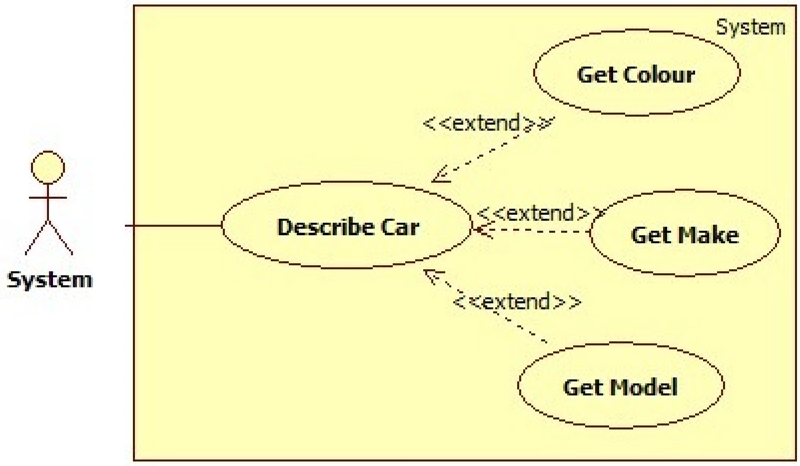
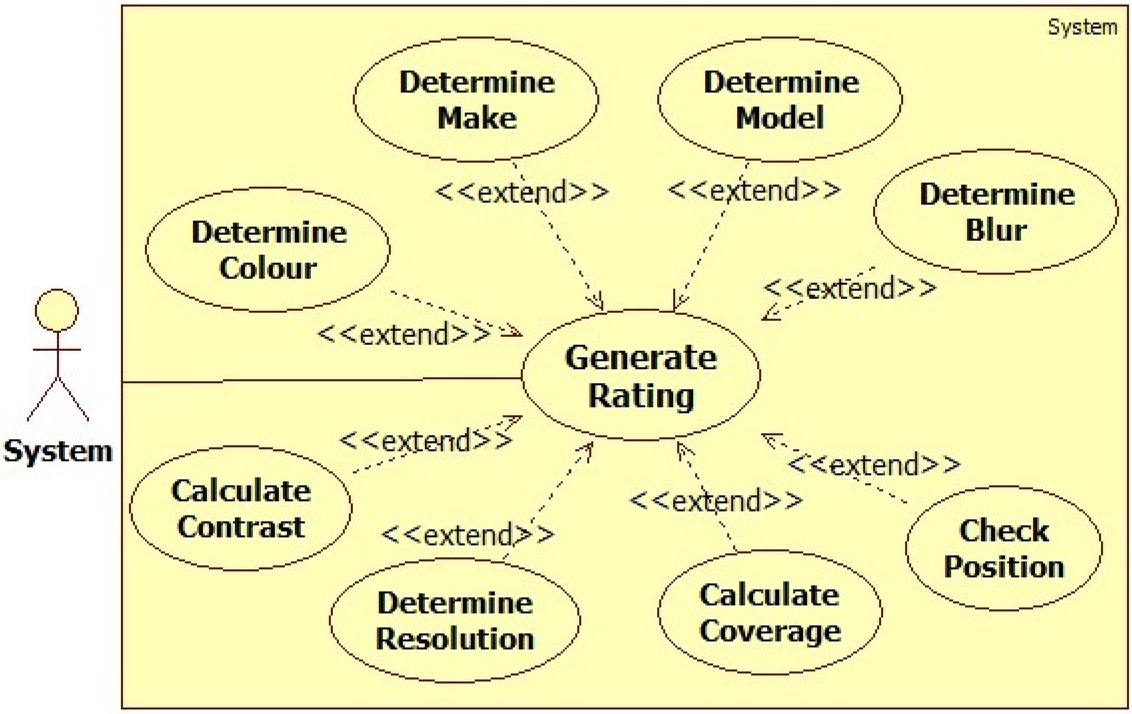


Figure 4: Generate Rating Low Level Use Case Diagram.



4.3 Use Cases

Detect Car Subsystem

Find Bounding Box	
Determines if the car exists in the photo and returns co-ordinates of the box containing the car	
Priority	Critical
Pre-Conditions	A valid image needs to be provided.
Post-Conditions	Car detection status provided.

Describe Car Subsystem

Determine Colour	
Determines the colour of the car and finds the closest colour bucket.	
Priority	Important
Pre-Conditions	A car needs to exist in the bounding box.
Post-Conditions	Colour of the car is determined.

Determine Make	
Determines the make of the car.	
Priority	Nice to have
Pre-Conditions	A car needs to exist in the bounding box. A make classifier needs to exist.
Post-Conditions	Make status determined.

Determine Model	
Determines the model of the car.	
Priority	Nice to have
Pre-Conditions	A car needs to exist in the bounding box.
Post-Conditions	Model status determined.

Generate Rating Subsystem

Get Colour Value	
Returns a weighted value based whether the colour has been determined or not.	
Priority	Important
Pre-Conditions	Use case "Determine Colour" was executed successfully.
Post-Conditions	Colour value successfully calculated and returned.

Get Make Value	
Returns a weighted value based whether the make has been determined or not.	
Priority	Important
Pre-Conditions	Use case "Determine Make" was executed successfully.
Post-Conditions	Make value successfully calculated and returned.

Get Model Value	
Returns a weighted value based whether the model has been determined or not.	
Priority	Important
Pre-Conditions	Use case "Determine Model" was executed successfully.
Post-Conditions	Model value successfully calculated and returned.

Calculate Blur	
Stretches the bounding box to 480x320 pixels and calculates and returns the blur value.	
Priority	Important
Pre-Conditions	Bounding box exists.
Post-Conditions	Blur value successfully calculated and returned.

Calculate Position	
Determines where in the photo the car placed, and returns the value based on the position.	
Priority	Important
Pre-Conditions	Bounding box exists.
Post-Conditions	Car position value successfully calculated and returned.

Calculate Coverage	
Calculates the coverage the car occupies in the photo.	
Priority	Important
Pre-Conditions	Bounding box exists.
Post-Conditions	Car coverage value successfully calculated and returned.

Calculate Resolution	
.	
Priority	Important
Pre-Conditions	Valid image provided.
Post-Conditions	Photo resolution value successfully calculated and returned.

Calculate Contrast	
Calculates contrast ratio of the photo.	
Priority	Important
Pre-Conditions	Valid image provided.
Post-Conditions	Photo contrast value successfully calculated and returned.

4.4 Process Specifications

Figure 5: Activity diagram of FindBoundingBox