Docker-based Automation Framework for Automating Software Application's Deployment and Performance Evaluation: Applied on Image Processing Algorithms

By

Acil Ramadan Ibrahim Abdel Naby

A Thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of

M.Sc. in Computer Engineering

Memorial University of Newfoundland

October 2018

Under the supervision of

Theodore S. Norvell, Ph.D., P.Eng.

Associate Professor

Electrical and Computer Engineering

Faculty of Engineering & Applied Science

Memorial University of Newfoundland

St. John's, NL A1B 3X5, Canada

Mohamed Shehata, Ph.D., P.Eng.

Assistant Professor

Electrical and Computer Engineering

Faculty of Engineering & Applied Science

Memorial University of Newfoundland

St. John's, NL A1B 3X5, Canada

Abstract

Automating the execution of software algorithms is a complex task that involves many dependencies, e.g., algorithm's procedures, inputs, outputs, the used programming language, operating system, and hardware dependencies. Then, each type of software algorithms requires a specific performance evaluation criterion to evaluate the algorithm while and after execution. Finally, a comparison between multiple algorithms is performed to select the most suitable algorithm for a specific task. In this work, a framework is designed based on Docker is used for those two purposes: automating the execution of the provided algorithm(s), automating the evaluation of these algorithms and producing a comparison evaluation after ADESA completes the software(s)'s execution.

The framework is applied to the Automatic Evaluation of Image Processing Algorithms (AEIPA). AEIPA automates the processes of: searching for a matching data set or uploading a new data set, uploading a new IPA that has been developed using any programming language, searching for matching IPA(s), executing all those IPAs with one or more data set, and comparing the results of the executed IPA(s) using each data set. AEIPA is a prototype system for implementing ADESA system. As a case study, two face detection algorithms that are implementing the Haar Cascade classifier using C++ with OpenCV and Python with OpenCV, have been used to evaluate AEIPA. Results from this work confirm that AEIPA is open to supporting the execution of any programming language by using Docker images.

Acknowledgments

First and foremost, I would like to thank my supervisors sincerely, Dr. Theodore S. Norvell and Dr. Mohamed A. Shehata, who guided me through this research experience, taught me how research is done, provided insightful advice, and developed my attention to details. I would also like to extend my gratitude to the department of computer engineering, Memorial University in Newfoundland. Without their generous support, this thesis would have been impossible. NSERC (Natural Sciences and Engineering Research Council of Canada) Strategic Projects Grant and scholarships from the University's School of Graduate Studies has financially supported this research.

I want to take this opportunity also to express my heartfelt gratitude to my family for their unwavering encouragement. They are my backbone to hold me up and my always supporters. Also, I would like to dedicate all my work to my beloved father Ramadan I. Abdel Naby who passed away during my master's journey.

Table of Contents

Publications

- A. Naby, M. S. Shehata, T. S. Norvell, “AEIPA: Docker-based system for Automated Evaluation of Image Processing Algorithms” IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE) 2017

- A. Naby, M. S. Shehata, T. S. Norvell, “A Survey of Distributed Architectures for Computer Vision Systems” Newfoundland Electrical and Computer Engineering Conference (NECEC) 2015

Abbreviations

DEPSAA	Deployment and Evaluation of Performance of Software Algorithms Automation framework
ADESA	Automation framework for Deployment and Evaluation of Software Applications
AEIPA	Automated Evaluation of Image Processing Algorithms
IPAs	Image Processing Algorithms
MVC	Model-View-Controller framework
DTOs	Data Transfer Objects
DAOs	Data Access Objects

Table of Figures

Chapter 1

1. Introduction

1.1 Motivation

Software engineering is apprehensive about developing composite software systems. These software systems go through the software development life cycle (SDLC): requirements definition, system design and analysis, implementation, testing and quality control, verification and validation, maintenance, deployment, reengineering, and reuse as detailed in [12]. The increasing complexity of the software systems led to the need to use automated approaches in order to develop and evolve in an economical and timely manner. Applying automation to software engineering activities increases significantly the productivity and quality of the software being developed. Automating software systems are constructed by adapting, modeling, and representing knowledge in software processes and artifacts.

In this work, research focuses on automating two main software engineering processes. The first one is automating the deployment and execution processes of any software. This research is narrowed to the study of software algorithms specifically because of their atomicity characteristic. The second problem is automating the evaluation of the executed software, i.e., automating the software quality control process.

The process of developing software algorithms involves many different factors such as the used programing language, operating system, hardware specifications, multiple inputs to

the algorithm and multiple outputs. The combination of all these factors led significantly to the complexity of automatically deploying the software.

Software algorithm's evaluation is the most expensive task in software development life cycle (SDLC) [1]. Evaluating an algorithm takes over 50% of the cost of the SDLC. Besides, the algorithm's evaluation is the chief of detecting malfunctioning software and errors that would lead to additional cost [1]. Software customers reported that they had spent twenty-one billion US dollars on software maintenance as a result of inadequate software evaluation and errors reported. Investing in software evaluation infrastructure saves these previously mentioned expenses [2]. Time limitations and human resources costs are noteworthy restraints of the evaluation process [4, 5]. That is why software evaluation helps to improve the quality and effectiveness of the software algorithm and reduces the software costs in the long run.

In this thesis, the algorithm's evaluation has been automated in order to improve the effectiveness of the evaluation process. A report is generated after evaluating the algorithm while being executed. The report contains some predefined evaluation matrices, in addition to a comparison between all the algorithms being executed. Software engineering developers can focus more on developing the software algorithm and performing complex test cases, leaving the repetitive evaluation tasks to be executed automatically. By this, human resources can be used more efficiently, which consequently may result in saving in the testing, evaluation processes and overall development budget [3]. Investing in the development of automating the evaluation of algorithm gets larger evaluation coverage and saves all the headache of delivering a malfunctioning software.

Software industry recognized the importance of automating the evaluation of software that 2.6 billion dollars are invested in automating the evaluation tools of software in 2004 [6]. Evaluation automation is more suitable for running repetitive tasks using different inputs [7,8]. Also, the evaluation automation process depends predominantly on the testability of the software algorithm [9]. In this work, the software algorithm to be evaluated needs to be developed in an independent form. Meaning that the software algorithm needs to be an encapsulated set of procedures that require a predefined type of input(s) and produce an expected type of output(s) to become eligible for evaluation automation. The output(s) of the software algorithm is used as an input to the evaluation algorithm in order to evaluate the quality and efficiency of the software algorithm. Typical evaluation automation process includes the development of an evaluation algorithm(s), executing that evaluation algorithm(s) using the output of the software algorithm versus the expected output, and verifying the final results.

The reduction of software costs in addition to the improvement of the quality of software products are the key objectives in all software development processes [10, 11]. Thus, a successful introduction of automating the software evaluation infrastructure combines these two objectives.

The outcome of this study is an implemented framework that serves twofold. First, automating the software algorithm(s) deployment and execution. Secondly, automating the evaluation process of the execution results and generating an evaluation report. Further, a discussion is provided about the usability, applicability, and maintainability of the obstacles faced during the conducted research. Results show that the software engineering life cycle is speeded up by automating the deployment and execution of the software and automating the evaluation

of the software's execution. Also, Docker has provided ultimate portability by separating the development environment and the deployment environment. Software dependencies are no more a concern to the software developers while designing their code and that is because of using fully independent Docker images that require only the source code in order to be executed.

1.2 Research Questions

The primary aim of this research is to study and develop a software automation framework based on Docker for automating the deployment and evaluation processes of software algorithms. This research leads to some fundamental questions as follows:

Is it possible to automate the creation and deployment of the software working environment?

Using Docker, a whole working environment can be created using a sequence of Unix commands to be executed using Docker Daemon. In this work, a separate Docker image is built for each unique working environment. For example, a Docker image can contain Ubuntu V.16, Python V.3.7.1. Another Docker image will contain Ubuntu V.16, Python V.2.7.15. Each Docker image is unique and can be instantiated into Docker containers by just selecting the needed working environment type.

Is automating the evaluation of a software algorithm useful?

The more time the execution of a software algorithm takes the more time evaluating that software takes. Algorithms require evaluating the final results of the execution. Also, some algorithms' outcomes, like the output of Image Processing Algorithms (IPAs), would be only in a visual representation. Each output has to be analyzed and compared to benchmark outputs in order to evaluate the performance and accuracy of the algorithm. Quantitative results are required for an accurate comparison of the IPAs. This process is extremally hectic and time-consuming. The Evaluation module allows the visual results of the algorithms to be examined according to a set of standard evaluation matrices. Typically, the Evaluation submodule automatically generates evaluation matrices [13]: true positive, false positive, true negative, false negative, true positive rate, false positive rate, precision, and recall, for the visual results of IPAs.

Does the software automation framework presented in this work enhance the mobility of executing multiple software applications through a distributed system?

Yes, the implemented framework generates independent Docker containers that contain a separate working environment. Each algorithm executes in a separate Docker container, and multiple containers can be executed at the same time in parallel. Each of the docker containers is created in a separate thread, and each Thread is processing only one data input at a time. Containerization enhances tremendously the performance of executing multiple algorithms, especially with large data sets.

1.3 Problem Overview

The primary objectives of this thesis are to develop a framework for automating software engineering activities: execution and evaluation. The new framework addresses the problem of automating the execution of software algorithms using Docker containers and automatically evaluating and comparing the results. The complexity of the unlimited dependencies combination for each software algorithm makes it hard to generalize a single approach to automate the execution and evaluation of the software algorithm. Dependencies can be the used programming language such as such as C/C++ (GCC), Java, PHP, Python, Hy (Hylang), Go (Golang), Node, Perl, Rails, Clojure, Ruby, OpenCV and Matlab, Operating system, hardware specifications, and even the slight differences between each version of the programming language.

To the other side, software developers spend much time searching for the appropriate dataset for executing the software algorithm. Also, there is a hectic effort in searching for, installing, executing and evaluating all the algorithms that match the same type of the algorithm being developed. As explained in the case study of the Automated Execution of Image Processing Algorithms (AEIPA) in chapter 4, each IPA requires a specific type of images dataset. Preparing the required images' dataset, executing them against each IPA, and evaluating the final results have always been time-consuming problems to the developers.

1.4 Organization of the Thesis

This thesis is structured as follows. First, a background on the research problem and the research questions are explained in the first chapter, the Introduction. In Chapter 2, an overview of the related work in software engineering automation. This chapter defines software algorithms, their structure, and place in software systems, software automation, software evaluation automation, and Docker containerization are provided. A general-purpose framework system design is explained thoroughly in Chapter 3 with its architecture and implementation. In the same chapter, a the four modules will be explained in details: ADESA Input Supplier, ADESA Software Factory, Docker, and ADESA Data Repository.

A high-level explanation of the framework system design and generalizability of this work is in chapter 3. Then, a more detailed digging into the twofold: software deployment automation system design and performance evaluation automation are explained in detail in fourth and fifth chapters respectively. An implementation of ADESA (Automatic Evaluation of Image Processing Algorithms) is explained intensively as a case study in Chapter 4. The thesis concludes with a summary of the contributions, limitations, and outline of the future work in Chapter 5. All the related references are listed in the last chapter, Chapter 6.

Chapter 2

2. Related Work

This chapter will review the works related to software engineering automation for both deployment and evaluation activities. For a better understanding of this domain and the different aspects of the problem, this review will begin with an overview of software algorithms since they are the basic components that need to be automated. Also, algorithm's inputs and outputs, software automation, software evaluation automation, and Docker containerization are explained in detail. After this, software engineering automation will be reviewed in how each activity in the SDLC is automated. A discussion is provided at the end of this chapter explains how these existing works motivated the work have presented in this thesis.

2.1. Software Algorithms

A software algorithm is the specification of a well-defined set of computational instructions that accomplish a certain computer task. Any computer program is designed based on an algorithm(s). An algorithm is the set of procedures that are implemented in order to solve an algorithmic problem. An algorithmic problem can be defined by specifying some value(s) as input instances and produces some other value(s) as output(s) after executing the algorithm [14, 15]. Thus, the software algorithm is the sequence of procedures, with a precise description, for transforming inputs into outputs as in figure 1.

Figure 1 Software Algorithm Components

As an example, consider the problem search problem using a key. This problem searches a list of values to find a certain searching key. The output should be the index number of the matching value or a zero in case not finding the key in the list. This algorithmic search problem can be defined as follows:

Input: A list of n numbers (x_1, x_2,…, x_n) and a key (k).

Output: i where $0 \leq i \leq n$.

As an example, given the instance input list (W, Y, N, I, D) and a searching key (N), a searching algorithm returns as output (3). Note how general this algorithm is. It can work correctly in the characters list, and numbers list using the equality operator.

A software algorithm is considered working correctly only if every input instance results in the correct output. Contrary, the correctness of some algorithms is not as important as controlling the error rate. Examples of these algorithms can be found in image processing algorithms and number theoretic algorithms. Evaluating a whole system performance relies on the efficiency of the used algorithms as much as the efficiency of the used hardware, and operating system.

2.2. Analyzing Software Algorithms

Analyzing any software algorithm measures, the predicted the resources that the algorithm requires. Resources can be computer hardware, memory, computational time, and communication bandwidth. A software algorithm is said to be good by identifying four desirable properties: correctness, efficiency, and the ability to be reused [12, 14].

As mentioned in the previous section, a software algorithm is marked as correct as long as it every input instance yields correct answer every time the algorithm executes. A set of precise reasoning needs to be determined to prove that an algorithm is correct. Implementing such a software algorithm is time-consuming and is liable to errors.

Algorithm's efficiency must not be disregarded, especially for real-time programming. Identifying the efficiency of an algorithm can be done by analyzing and comparing it to many other software algorithms in the same specific problem category. The comparison process between many software algorithms is difficult because the behavior of each algorithm is different for each possible input instance. The taken time by the search algorithm, as an example, is determined by the input instance, i.e., searching for a value in a large list of inputs takes much more time than searching for a value in two values list. Moreover, the running time for finding value at the beginning of a large input list, that is called best scenario case, is less than the running time in case of finding value at the end of the same input list, called worst scenario case.

On the other hand, the software algorithm's reusability is essential to adapt the algorithm to various software applications [16, 17]. Algorithms should be designed in a generic way that leads to easy and flexible implementation.

2.3. Automating Software Engineering Processes

2.4. Docker Engine

Docker (http://www.docker.com) is an open source technology that performs virtualization in the operating system level. Docker is a software platform is designed for both Linux and Windows. Docker simulates the OS kernel's resource isolation features as in cgroups in Linux. That is how Docker can create multiple Docker Containers independently using Docker Images.

Docker consists of Docker Engine and Docker Hub. Docker Engine is the core technology that builds and runs Docker containers which are instances of Docker images. Docker Hub is a Cloud-based service that shares Docker images. It allows building, testing, and storing Docker images. Docker Engine creates Docker Images using a set of well-defined Unix instructions for an executable version of a software application. A Docker Image consists of tools, system libraries, and dependencies for the executable application code.

A Docker Image is a file that includes accumulative layers used to be executed in one or multiple instances of Docker Containers. Docker containerization has gained sufficiently great importance recently because it encapsulates a whole operating system with an executable application code above an abstraction layer from physical hardware [18-20]. Docker Containers adopted DevOps and microservices. Docker Container supports the agility of deployment and evaluation of software applications as in [21]. Nevertheless, the growing need for software portability, high performance, and reproducibility from the development environment to production environment made it critical to evaluate the software application while being executed using the Docker Container. Docker containers are lightweight as they can be launched when needed without a guest OS as depicted in the below Figure 2.4.1. Docker is the leading container solution to isolate the runtime environment from the underlying host and networking resources. Docker containers provide pliable wrapping solution for deploying and shipping software applications [22].

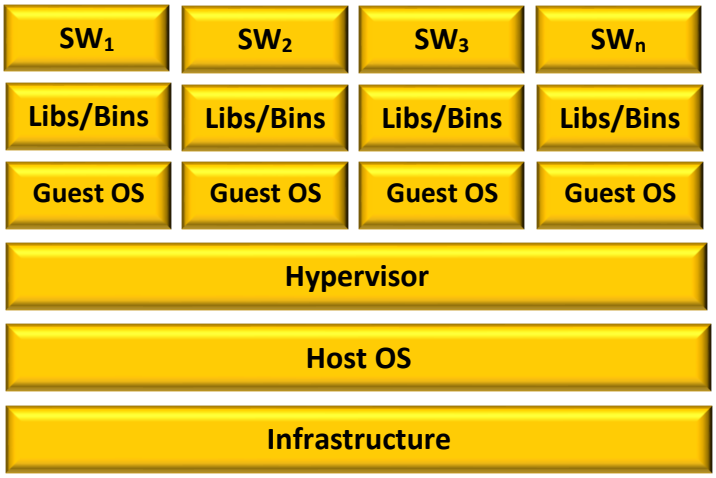


Figure 2.4.1: Virtual Machines

Containers and virtual machines (VMs) have common functionalities of resource allocation and encapsulation, but they use different architectural approaches. Containers are more efficient, scalable, portable, and easy to deploy compared to virtual machines [31, 32]. Docker engine manages resource utilization more efficiently as it uses centralized resource management within the created containers. On the other side, VM hypervisor allocates the maximum limit of resources to each virtual machine created. Docker containers dynamically allocate the resources, e.g., memory, processors, and page cache, at runtime [33].

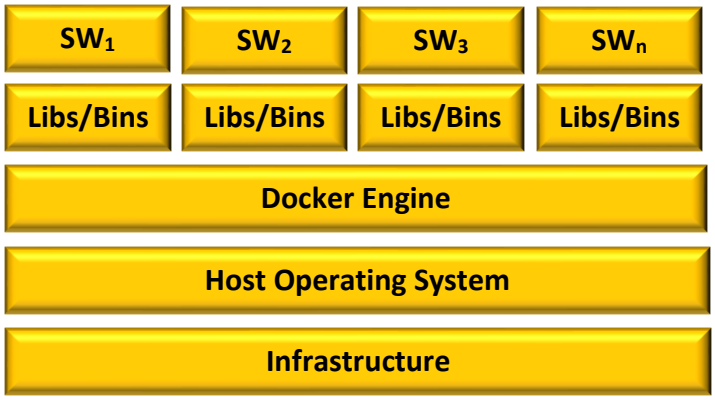


Figure 2.4.2: Docker Architecture

Docker makes installation a lot easier and unlimitedly replicable as proved in AEIPA system in [23]. Docker packages deployments into images without starting one up from scratch. The contents of a Docker image are defined online in Docker Hub. Hence, Docker images can be pushed, pulled to, and from Docker Hub without cluttering the hosting machine with lots of files.

Chapter 3

3. Automation of Deployment and Evaluation of Software Algorithms

3.1. Proposed System

This research is focused on developing a framework for automating the deployment and evaluation processes of the software algorithms, ADESA. The goal of this research can be divided into two parts: first, to automate the deployment and execution of software algorithms using input data instances and second, to automate the evaluation of the performance of the executed software algorithm and compare it with other software algorithms in the same algorithm's category. Thus, five elements are the key components of this system: input data instance, software algorithm's execution, the output of the algorithm's execution, evaluating the performance of the algorithm, comparing the algorithm with other algorithms and representing the results visually. The remainder of this chapter will present each of these elements, within the two research goals, with the design decisions that fit them.

3.2. ADESA Framework Architecture

In this section, an overview of the ADESA's framework design. UML diagrams are provided focusing on the interactions between the modules. This framework is used to automate the deployment and evaluation processes in software engineering. There are two major goals for this framework: automating the deployment and automating the evaluation of software's performance. The framework is divided into four modules: Input Supplier, Software Factory, Data Repository, and Docker Engine as in figure 3.1.

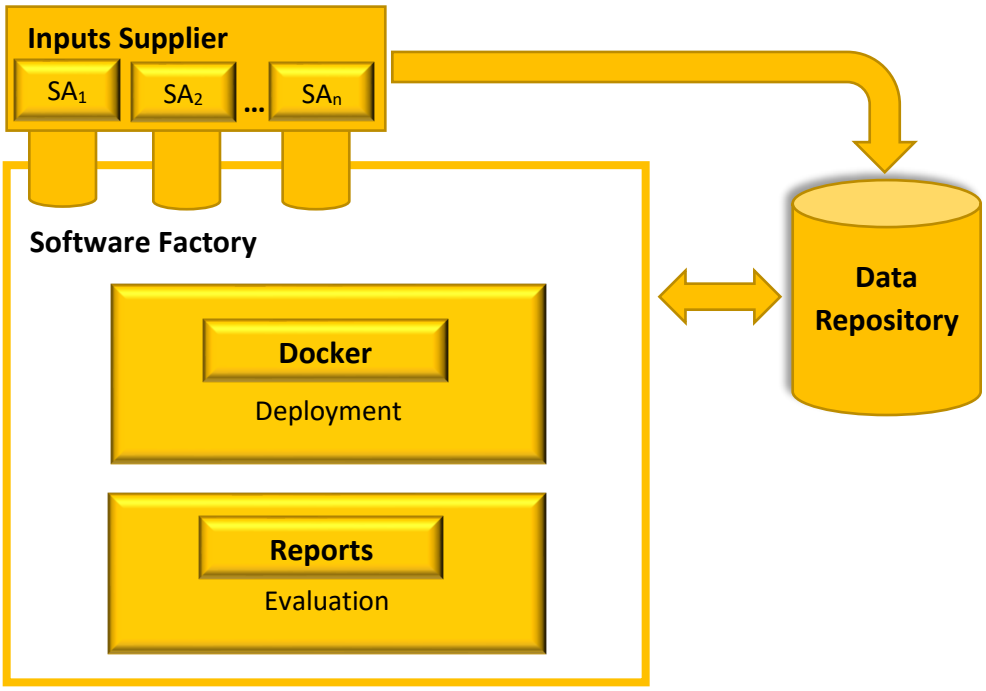


Figure 3.1: ADESA Framework Architecture

Input Supplier is responsible for providing the input instances to ADESA in a readable format. Software's execution along with its evaluation occurs in the Software Factory module. Docker Engine is used in order to execute each of the software instances in a separate environment according to the programming language that the software is written with, the operating system, and working environment dependencies. A report is generated with the evaluation metrics plus a comparison with other software while and after the software is being executed. Data Repository is where all the input instances, software source code, execution information, and execution results are stored. In the upcoming sections, a detailed description of each module is presented.

3.3. Model-View-Controller Implementation

Software applications have to interact with one, or more, thing in order to be useful. Software applications can interact with users, machines, and/or other software applications. That is why it is important to invest more efforts on the software's interfaces that interact with the software's outer world. It is likely to change the software's interfaces while keeping the underlying application constant. It is also reasonable to keep the essence of the software application separate from the interfaces. To cope with that, the Model-View-Controller architecture has been used in ADESA framework [45 - 47]. This design pattern is very useful for architecting interactive software applications as it is based on partitioning the application into independent layers [48, 49].

Model Layer

The Model layer is the unchanging essence of the software application. It is the set of classes, in object-oriented terms, while represent the core problem. The Model layer should not interact with the outer world [50]. All the interactions should go through the Controller Layer and/or Database storage [51].

Model layer in ADESA is divided into three layers: Data Transfer Objects (DTOs), Data Access Objects (DAOs), and Services. DTOs are the objects representing the software's data. DTOs are responsible for travelling the data between separate layers [52, 53]. DAOs maps application's calls from and to the database without exposing details of the database. DAOs are interfaces to databases [54]. When the business logic is complicated and expected to grow, it is preferred to create a Service layer. The Service layer is the middle layer between the DTOs layer and DAOs layer. It abstracts data access and business logic [55 – 57]. Service layer is driven by the software's use cases.

Two different systems that apply the mentioned layers above are shown in [58] and [59] for more details. Hadoop MapReduce data distribution architecture for processing input splits is presented in detail in [58]. Also, in [59], A large scale cybersearch engine (aka PolarHub) is implemented to discover the distributed geospatial services and data. PolarHub is developed using Service Oriented Architecture (SOA) and Data Access Objects (DAOs). The two systems show sustainable, high performance, scalable, and interactive platform.

View Layer

The View layer is the visual representation of the problem to be solved. It can be in the form of application program interface (API), graphical user interface (GUI), and/or command line interface (CLI). The View layer can be as simple as an input form, report sheet, or even a graphical view.

Controller Layer

The Controller layer manages the interactions between the View and Model layers. A controller is an instance that enables processing the View's input. Controller layer has the information of the operating system and used platforms.

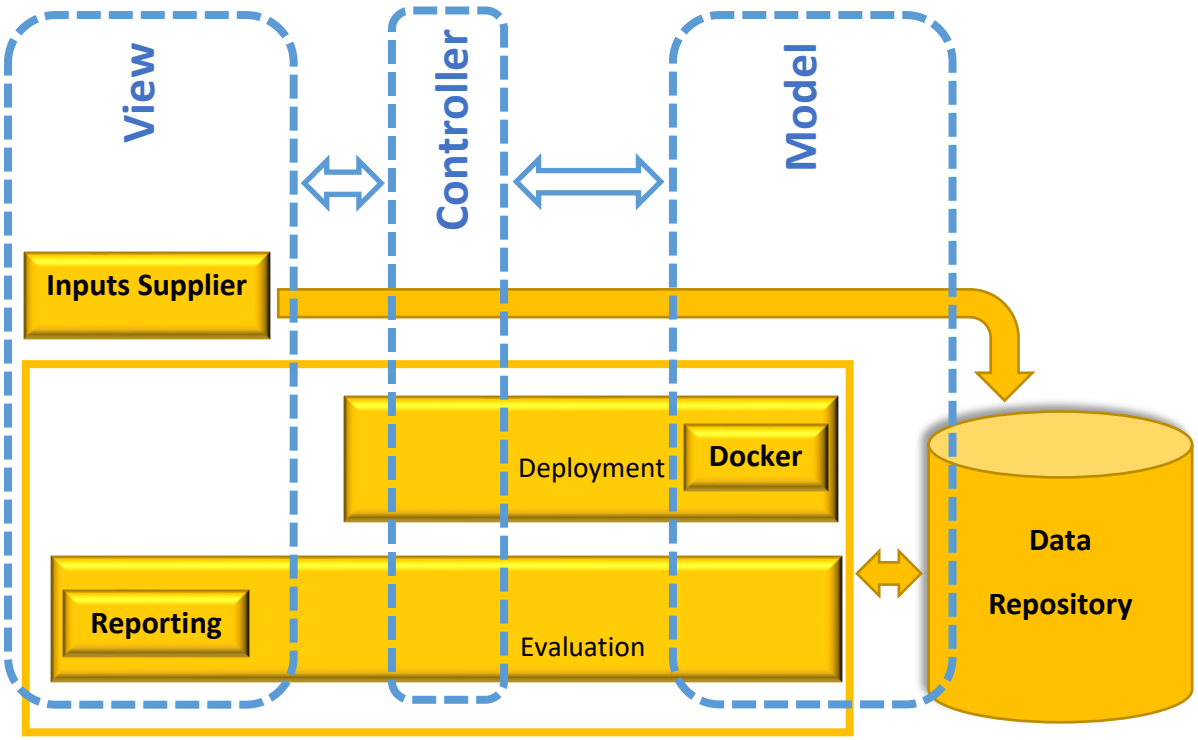


Figure 3.3.1: ADESA Framework Implementation Design

As shown in Figure 3.3.1, MVC design pattern is used in ADESA. The four modules that will be explained in the upcoming sections, are divided by the MVC layers. The Input Supplier and the Reporting modules interact with the users and are implemented totally in the View layer. Though, all the servlets controlling the interactions between the View layer and the business logic have been implemented in the Controller layer. The Controller layer maps the processing from user inputs, to the deployment module, then to the evaluation module, and finally back to the user's output results (Reporting). The Model layer contains all the business logic of ADESA. This layer

should be the least changeable component within ADESA. It has been divided into the three layers explained above: DTOs, DAOs, and Service layers.

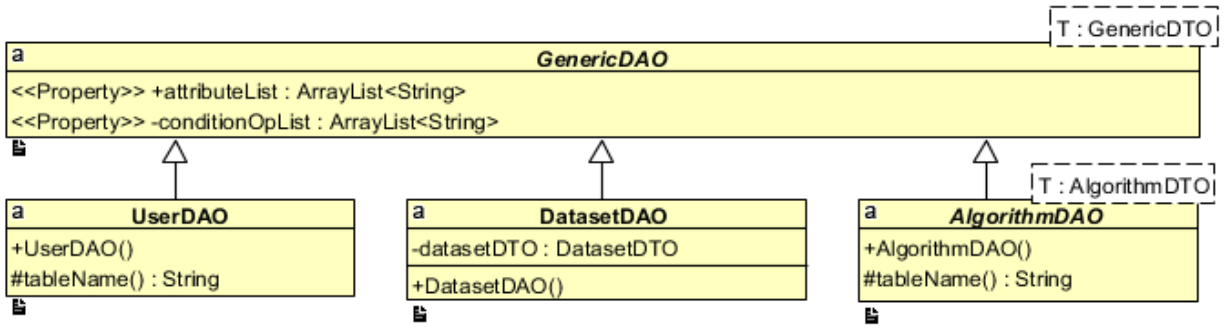


Figure 3.3.2: ADESA General DAOs

3.3.1. Input Supplier

ADESA framework plugs input instances through the Input Supplier. Input can be the input data set, software application's source code, evaluation criteria, and any other software applications to be executed and compared to. All the inputs are uploaded and stored to the Data Repository. Input data set can be anything the software application would need in order to start its processing. Any computation software program takes a set of input instances and produces a set of output values.

As an example, consider a problem of counting the number of occurrences of a word in a document. Pseudocode to solve this problem would be something like the one below:

```
word_count(String content, String key):
// content: the contents of the document
// key: word to count in the document
int count = 0;
for each word in content
    if word is equal to key
            count += 1;
    endif
```

The word_count function counts the occurrences of the second input parameter key in the first input parameter content. The inputs to such a problem would be a string of the word to count, the document, and the source code to the program that solves this problem, the programming language's name and version, other programs that would solve this problem, and the evaluation program.

The inputs to a software can be easily determined in discussions with the users of the software at an early stage of the development cycle [25]. The needed information to be used by a software application can be estimated from the user's external view, mostly in the requirements gathering activity in the SDLC [12].

ADESA applies the open-closed principle [28] that it can be extended to support any input format. ADESA users supply the needed inputs and ADESA delivers the provided input with its

interrelationships and representation to the Software Factory Module, i.e., the deployment automation. ADESA users povide the full path to a file that contains the input and ADESA's Input Supplier stores all the provided data into ADESA's Repository.

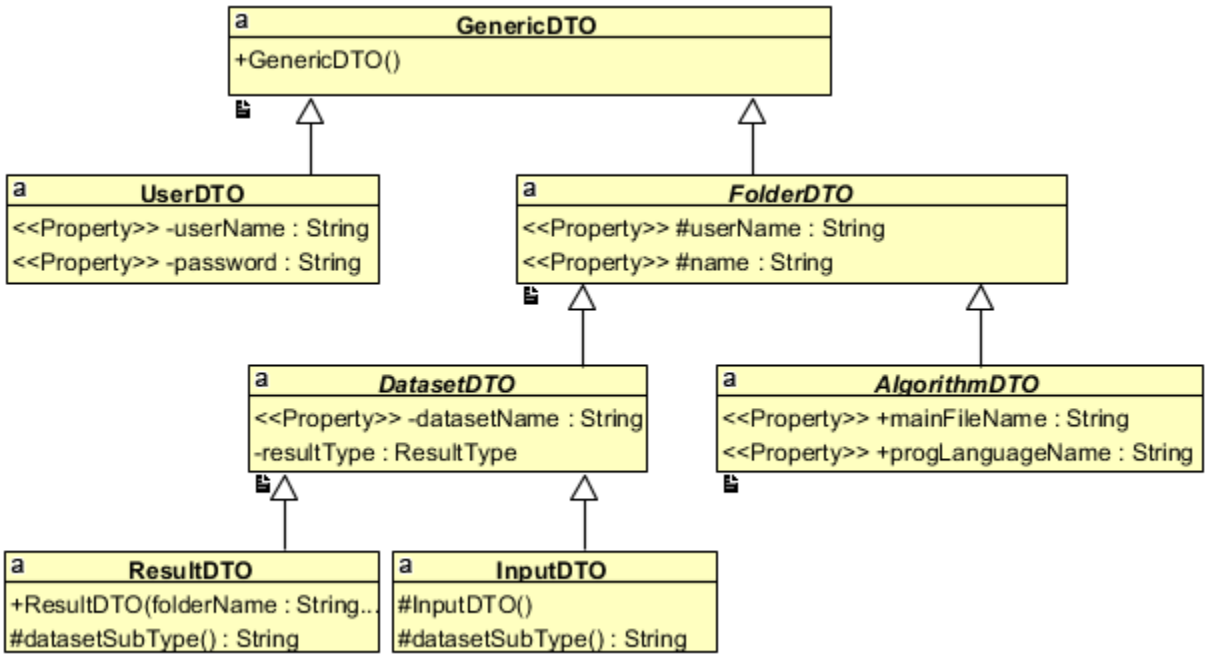


Figure 3.3.1.1: ADESA DTOs Layer

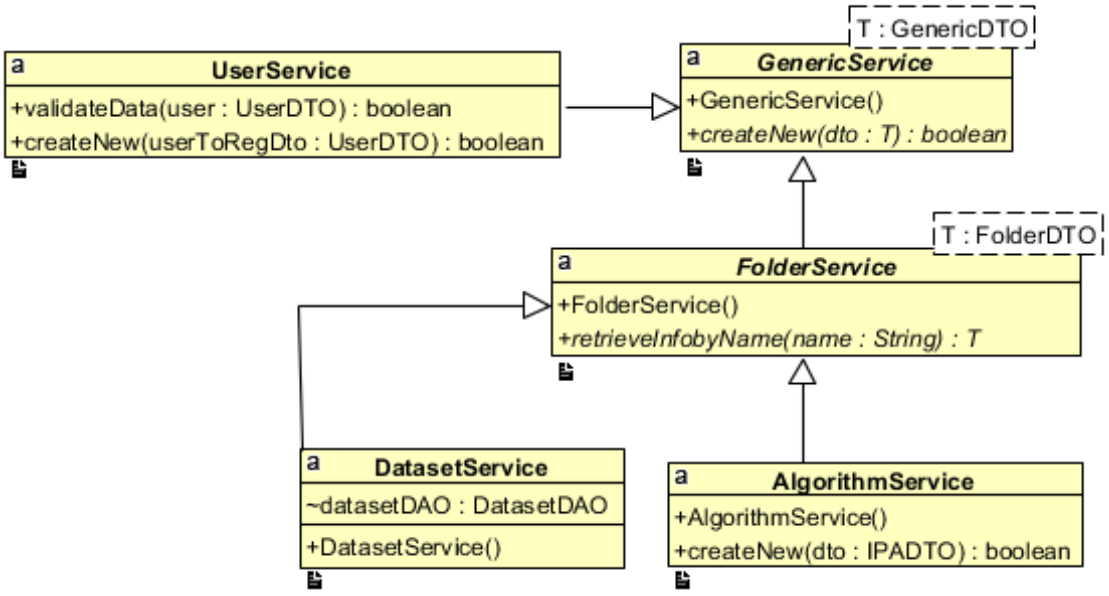


Figure 4: ADESA Service Layer

ADESA inputs and results can be all native and extended data types: Boolean, byte, unsigned byte, short, unsigned short, integer, long, float, double, RGB, complex, and file. The provided software application can be written using any preferred language supported by either Docker Hub or directly by AEIPA, such as C/C++ (GCC), Java, PHP, Python, Hy (Hylang), Go (Golang), Node, Perl, Rails, Clojure, and Ruby.

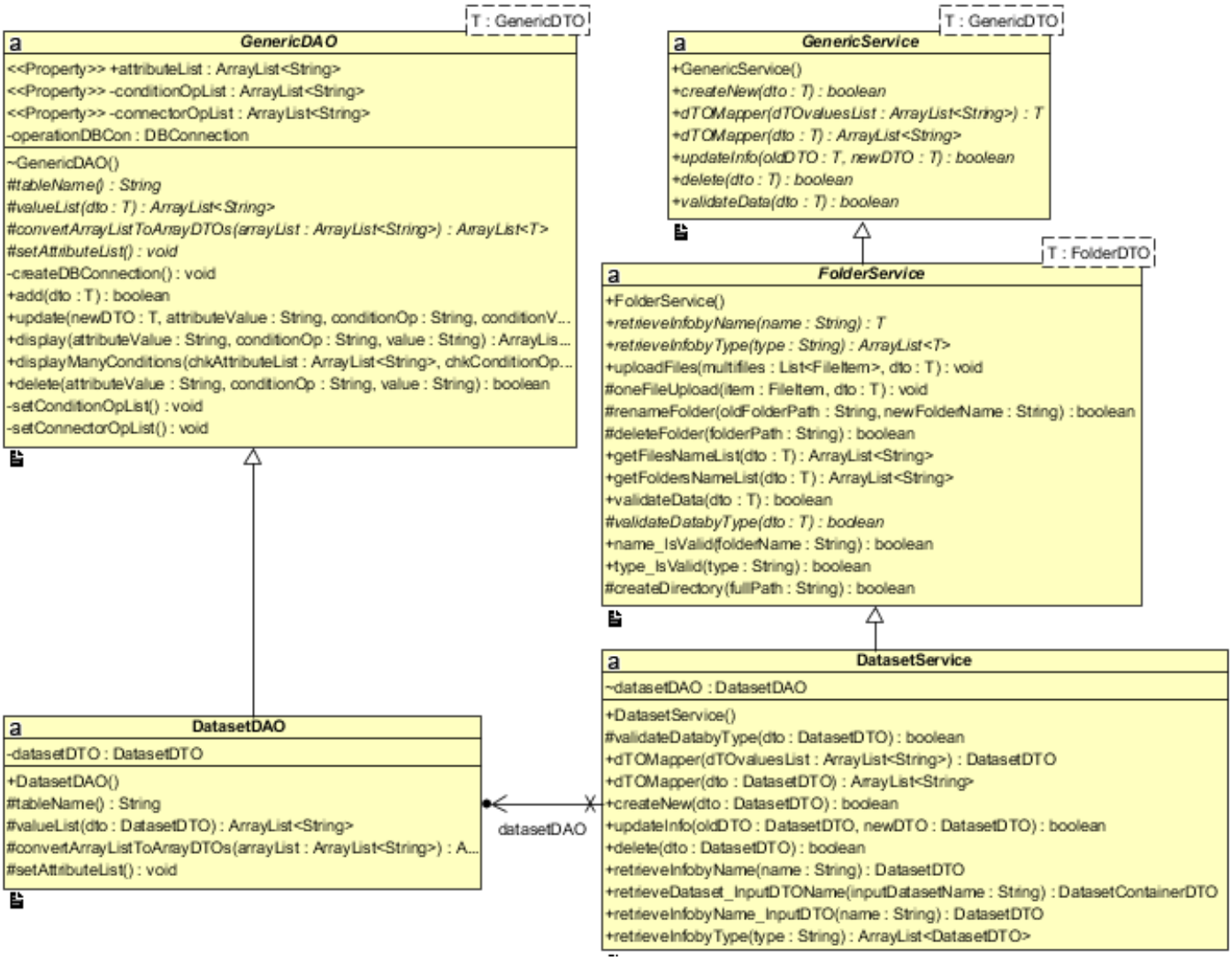


Figure 3.3.2: ADESA DAO-Service Layers relationship

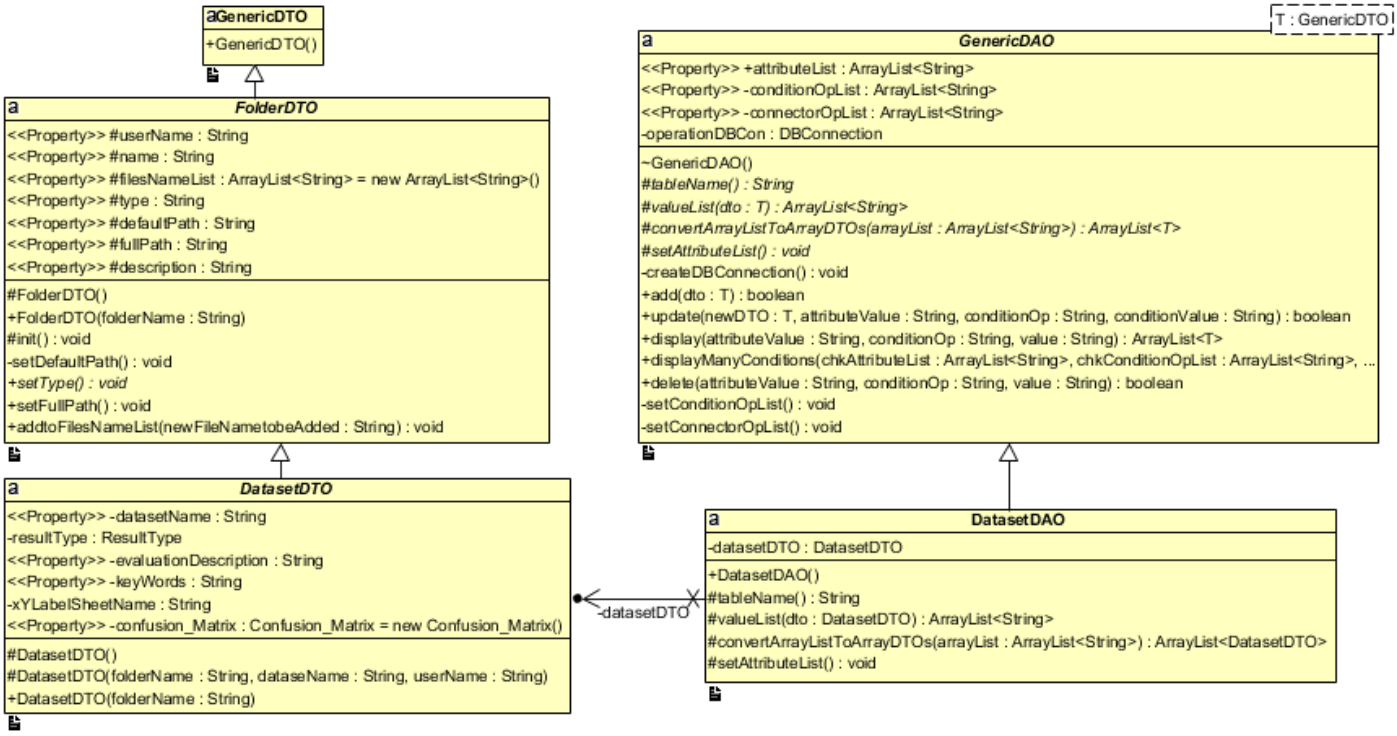


Figure 5 : ADESA Dataset DTOs-DAOs relationship

3.3.2. Software Applications Factory

In this section, an overview of the core module of ADESA that is the Software Applications Factory.

3.3.2.1. Deployment Automation Module

A sequence of instructions can be thought of as a string of "tokens." [25]

program size is determined by the data that must be processed by the program. [26]

Supported Programming Language(s)

Several

common programming languages are selected to study the

capability of the tools in reverse engineer source code. The

selected programming languages are PHP5, C++, Java, C#,

Delphi, Python and Visual Basic (V.B.). [27]

Open source software programs [29]

3.3.2.2. Performance Evaluation Automation Module

3.3.3. Data Repository

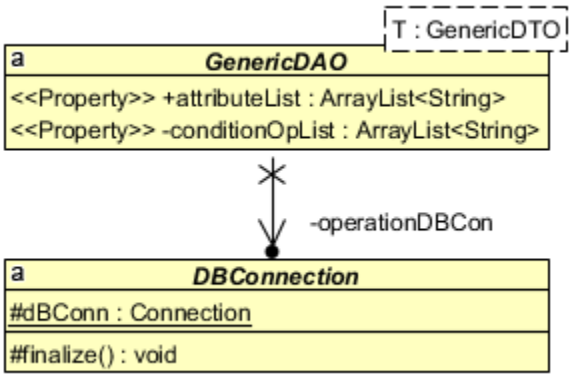


Figure 3.3.3.1: ADESA Database Repository

3.3.4. Docker Engine

Software application life cycle ought to frequent changes and updates thus raising many deployment and maintainability issues [34 - 36]. Research applications, in specific, rely on several incremental software pieces because of their experimental nature. These software applications are implemented with many environment implicit dependencies on libraries, programs, operating system, hardware specifications, and other components. As a result, these applications are difficult to install, configure, and deploy. Moving a software application from one environment to another environment used to have a little chance of running correctly without a significant effort. VM technologies were used as a solution to this problem but they had some disadvantages due to their need to create a complete cope of the OS files as explained in the previous chapter. As in [37], an overall copy of the VM requires to be reassembled and redeployed in order to add or change just a single file. In addition, it is impossible to reuse piece(s) of data or software inside a VM because VM's contents are not accessible with standard protocol [38].

Docker has recently gained an increasing level of attention because it allows software applications to execute in an encapsulated and portable package that can be distributed across computing platforms. Docker is used in ADESA framework to execute each software application in an isolated container that is created from an immutable Docker image. Docker images in

ADESA are predefined for constructing a wide range of ready-to-work environments. Each Docker image in ADESA is unique set of instructions to build a specific working environment. For example, there is a Docker image for building an environment with Ubuntu 18.04, SSH server to easily access the container, runit that replaces Ubuntu's Upstart, install_clean for installing apt packages, and Java 8.181. Another Docker image would be Ubuntu 16.04, SSH, runit, install_clean, and Java 7.67.

Input Supplier in ADESA framework provides the software application in a source code format with the environment specifications. The Docker image that matches the provided environment specifications executes and creates a Docker container. The Docker container establishes a ready environment as specified by ADESA Input Supplier.

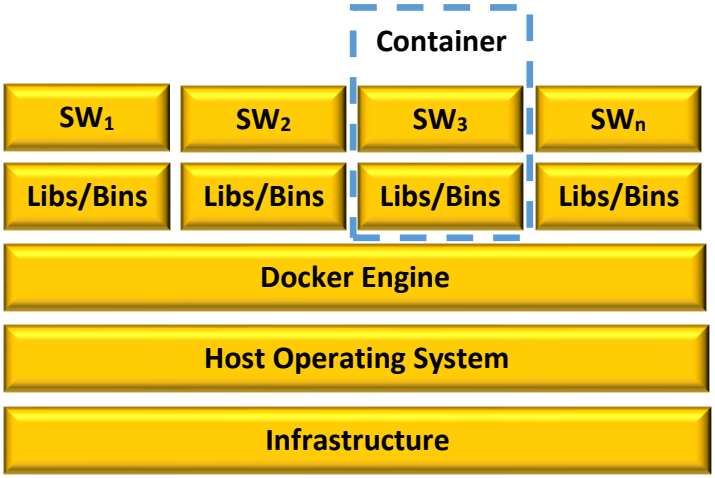


Figure 6.2.4: Docker containers and Software Applications

Docker containers created by ADESA's Docker images are prepared to execute a source code of a software program. There are many reasons to use Docker containers for executing the software applications. First, using Docker containers guarantees that each software application runs in a predictable system configuration that don't change on runtime owing to a misconfigured environment. Second, isolated Docker container prevents conflicts with other installed programs in the hosting environment. Third, many instances of Docker containers can execute in the same hosting environment, sharing the kernel with other programs.

Chapter 4

4. Image Processing Applications Case Study

Image processing is used by a broad range of applications that need digital cameras, computers, videos, and/or images. The human eye can easily distinguish significant characteristics even in very poor-quality images. The aim of image processing applications is to interpret images in a similar, or even faster, way as compared to a human being. Image processing applications powerfully manipulate images, quantify intensity, apply mathematics, detect edges, and enhance contrast operations to images. Image processing is a dynamic research area that is being continuously investigated. It is involved in many real-life applications, such as traffic monitoring [39], quality inspection [40] automobile parking [41], human identification [42], just to name a few. These applications add a lot of challenges which require a rapid evolution of image processing algorithms (IPAs).

However, too much effort is spent in evaluating the IPAs instead of focusing on IPAs enhancement. Typically, the Researches (who are the main users of AEIPA) have to go through all the following steps:

- Implementing all the related benchmark IPAs
- Setting up benchmark datasets to be ready for use
- Creating new datasets to test different challenges
- Evaluating each of the related benchmark IPAs individually

- Reporting the results which usually needs writing scripts/code manually

Although there are online repositories [43] that contain the implementation of the benchmark IPAs and benchmark datasets, researches get overwhelmed with configuring these benchmark IPAs and datasets to make them ready for use, and facing all the troubles of customizing them according to the working environment.

4.1. AEIPA Prototype

In this thesis, AEIPA (Automated Evaluation of Image Processing Algorithms) design is presented to solve the overhead that was mentioned above. AEIPA is an implementation of the ADESA framework. It allows automatic execution of IPAs, comparing them to already existing benchmark IPAs, and evaluation of the results. AEIPA is structured into three architectural modules: AEIPA Supplier, AEIPA Factory, and Docker Engine. AEIPA is isolated into a Docker container to allow agility and efficiency of the continuous integration and deployment activities. To the best of our knowledge, AEIPA (Automated Evaluation of Image Processing Algorithms) is the first system to solve these problems and facilitates the process of developing a new IPA.

This chapter is arranged as follows: Section II depicts the background of the paper. Section III illustrates the proposed system. The conclusion and future work are in section IV.

4.2. Motivation

Evaluating a new IPA is a time-consuming task. User must develop an IPA, then search for matching dataset(s) and ground truth, then execute and evaluate the IPA, then search for all related IPA(s), implement them, execute and evaluate them, finally, compare all the results together. And since image processing has been involved in most of daily life applications, the need to automate all these steps has emerged.

The major problems are the extreme variety of image processing categories, the used programming languages, IPAs' different programming languages, operating systems, and different hardware. All these steps complicated the automation process. There was need of a unified environment that would accept all these varieties and make it easy for the user to execute IPA, find all related IPA(s), execute them all against some specified dataset(s), and compare the results all together. Also, evaluation algorithms can be implemented using any programing language to be executed to evaluate the IPAs. There is no need to implement any evaluation algorithm from scratch.

4.3. Datasets

Dataset is a collection of data corresponds to a particular event or experiment. Datasets are huge in size due to the number of images\frames per each dataset. For example, in medical imaging [44], virologists generate 3D reconstructions of viruses from micrographs, biologists generate 3D confocal microscopy images, regional metabolic brain activity can be detected from PET and functional MRI scans by neuroscientists, radiologists identify and quantify tumors from MRI and CT scans. The processing and analysis of such datasets require complicated automation tools. In the past decades, image processing tasks are expensive and need specialized UNIX workstations for visualizing and analysis the images. Today most of the image processing software can be performed on inexpensive desktop working environments equipped with the appropriate graphics software and hardware.

. This paper introduces an extensible platformindependent, general-purpose image processing and visualization program specifically designed to meet the needs of a Internet-linked medical research community. The application named MIPAV (Medical Image Processing Analysis and Visualization) enables clinical and quantitative analysis of medical images over the Internet. Using MIPAV's standard userinterface and analysis tools, researchers and clinicians at remote sites can easily share research data and analyses, thereby enhancing their ability to study, diagnose, monitor, and treat medical disorders.

Dataset

- Dataset types (Camera, video, image)
- Categories (moving objects)

Groundtruth

Evaluating IPA with a Dataset

- Evaluation types
- Ground truth dataset
    - **Ground truth data:** Information collected from industry sources to determine how other firms (especially the best in class ones) achieve their high levels of performance.
    - * Every Input Dataset in AEIPA has a ground truth dataset

Evaluation reporting types

4.4. System Requirements

AEIPA system is used for automating the execution and evaluation of image processing algorithms (IPAs). Using the proposed system users are able to find all IPAs, relevant to their field, execute, and evaluate, and compare the execution results against other IPAs. Users don't have to worry anymore about the algorithm's dependencies (execution environment, or the

programing language that the IPA has been developed with). They only need to worry about developing their own IPA and upload it to AEIPA in order to be executed.

The users expected users should be: Researchers, software, and R&D developers. System can be easily used by users that there is no need for any prior knowledge about IPAs' used technologies, working environment, programing languages, or implementation details. Users don't need to worry about evaluation implementation, algorithms, or how they work.

System Scenarios:

1. *New users should be able to register to AEIPA with these data:*
    - Username
        - A mandatory field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted.
        - This field should be unique across all the users in the system.
    - Password
        - A mandatory field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted.
        - Typed letters should be hidden to the user. Letters can be replaced by asterisks or dots for security proposes.
    - First Name

- A mandatory field. A maximum of 45 letters are only accepted. Numbers and special characters should not be accepted.

- Last Name
  - A mandatory field. A maximum of 45 letters are only accepted. Numbers and special characters should not be accepted.
- Job
  - An optional field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted.
- Organization
  - An optional field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted.
- Date of Birth
  - An optional field. A valid date can be only accepted.

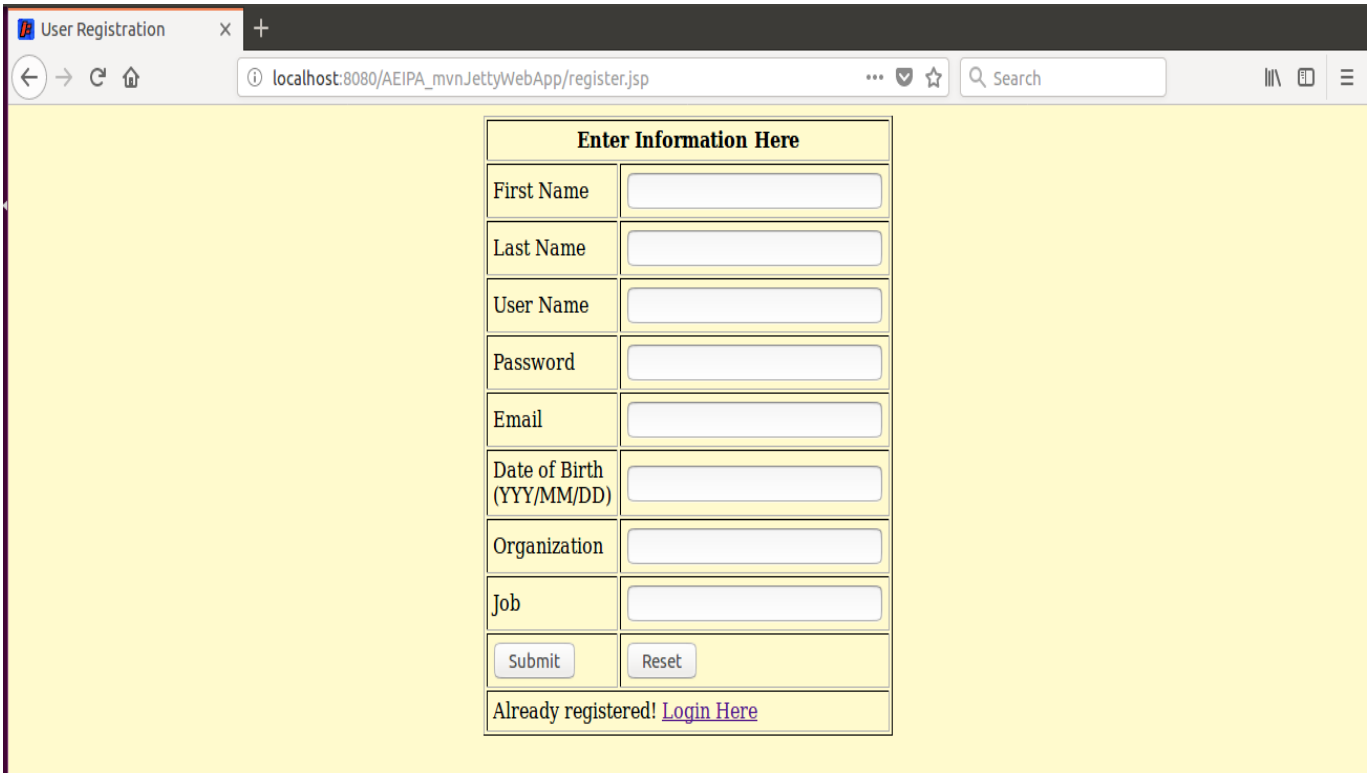


Figure 4.4.1: AEIPA User Registration

2. *User should be able to login immediately after registration using (Username and Password) as in Figure 4.4.2 with no need to any additional activation\approval steps.*
    - A valid Username and\or Password should only be accepted to log in.
    - Username and\or Password can't be empty.
    - SQL injections should be handled.

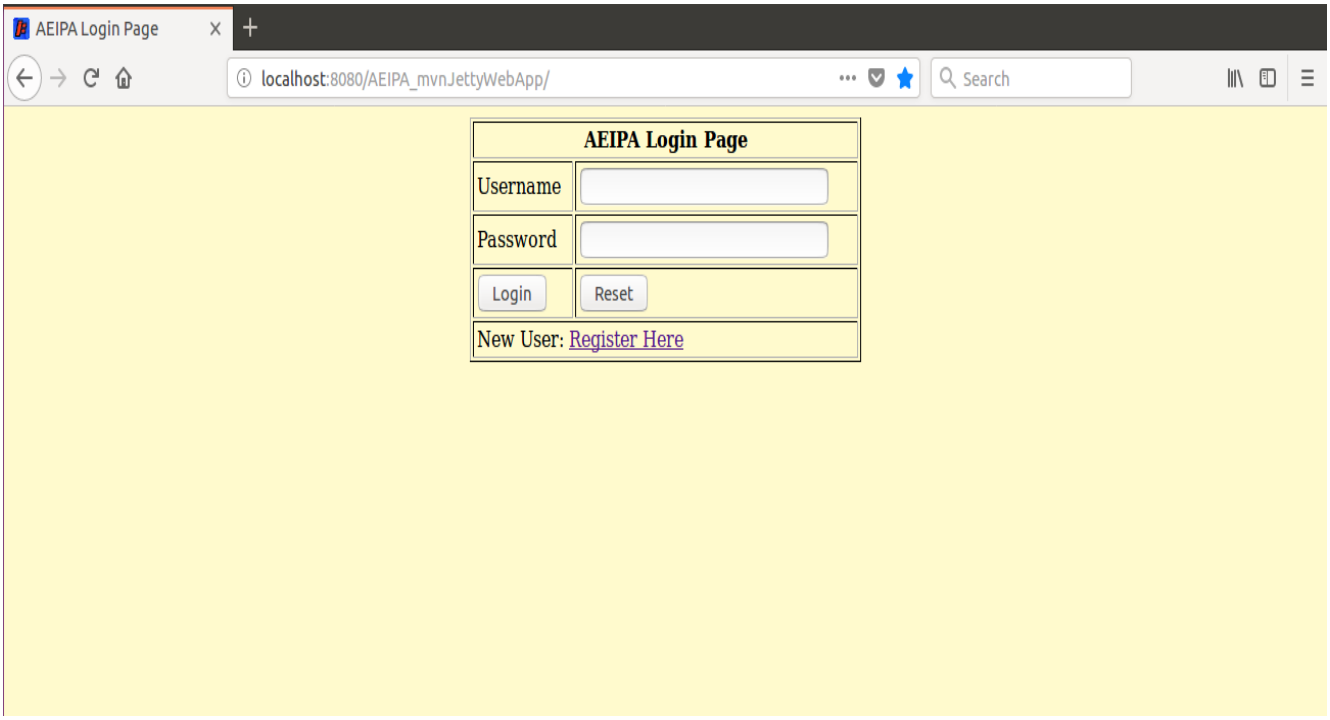


Figure 4.4.2: AEIPA Login Page

3. *Logged in users should be able to do any of the following actions in Figure 4.4.3:*

3.1 *Create a New IPA*

- *User should able to create a new IPA using these fields:*
  - Name
    - IPA name.
    - A mandatory field. A maximum of 45 characters of any combination of

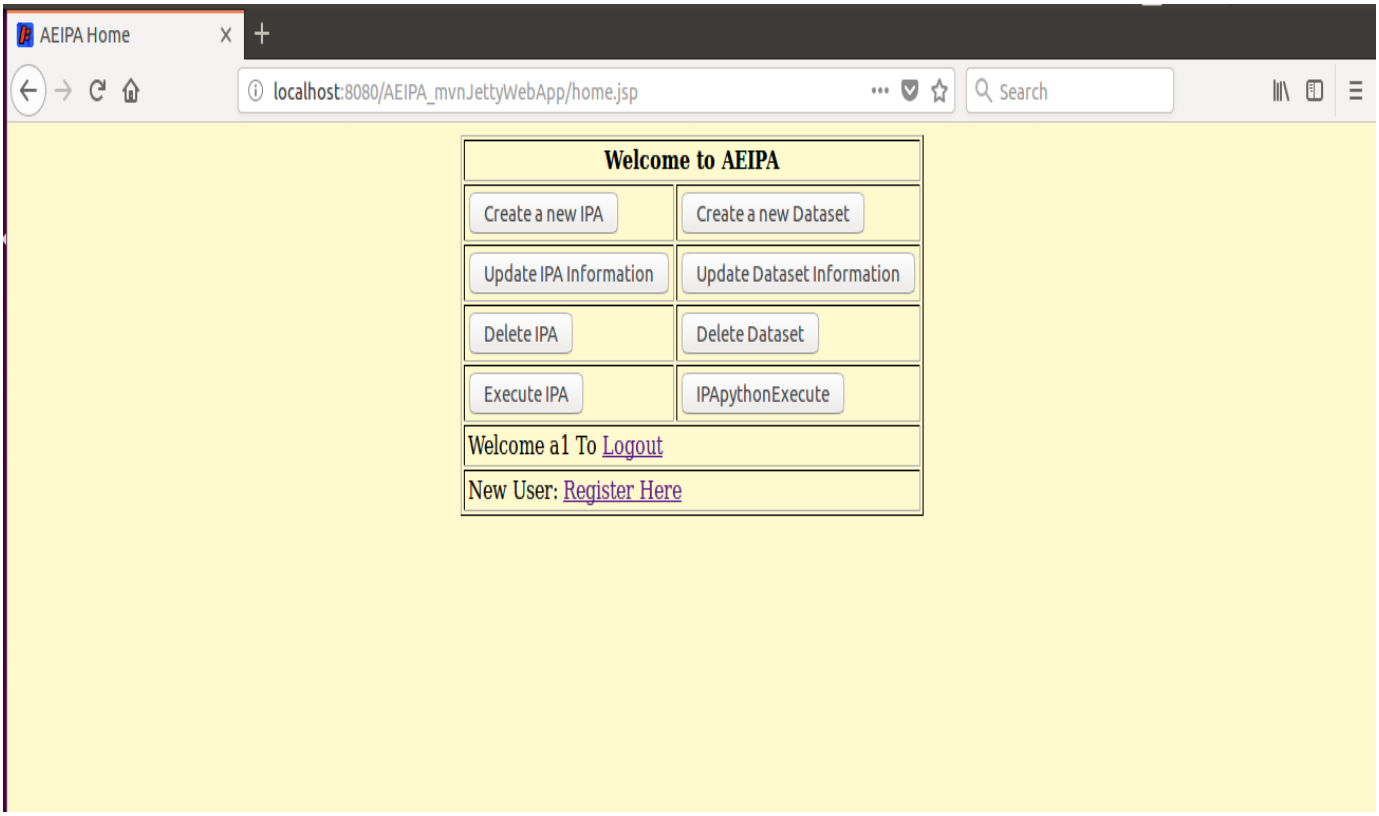


Figure 4.4.3: AEIPA Actions

- letters, numbers from 0 to 9, and\or special characters are accepted.
- This field should be unique across all the IPAs in the system.

◦ Description

- A brief description about the IPA, what it is about, the used algorithm, a comment regarding this specific uploaded IPA (for example what is new in this IPA version).
- It is an optional field. A maximum of 200 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted.

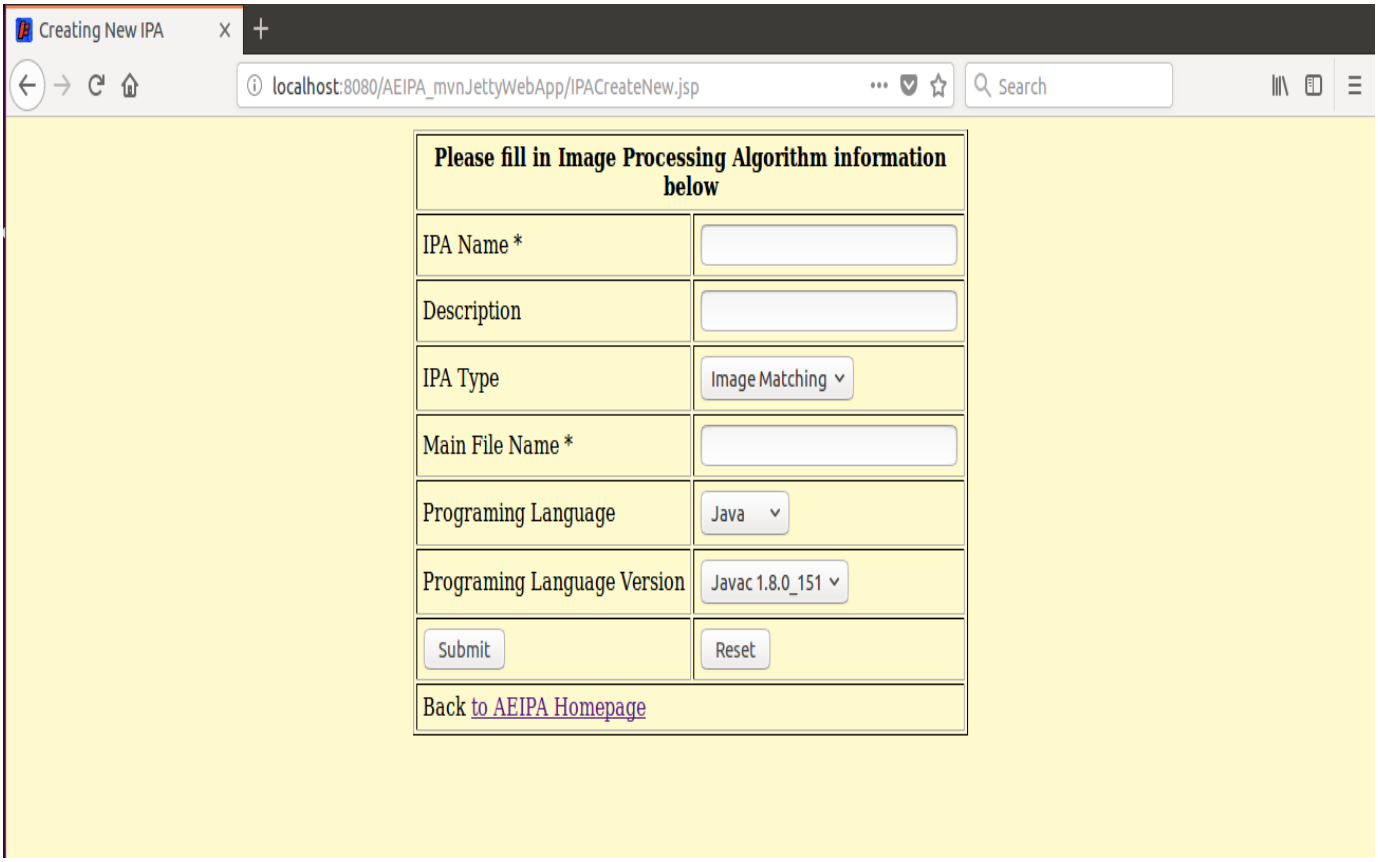


Figure 7: AEIPA Create a new IPA

- Type
  - A mandatory drop-down list. User can select the IPA type out of some specific options in a drop-down list. For now, the list contains: Image Matching or Image Segmentation.
  - This list should enable adding\removing IPAs types in future work.
- Main File Name

- The name of the main file that the execution should start with. There is no need to enter the extension of the file since the user will provide the used programing language name and version number in other fields.
- A mandatory field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted.
- User is responsible of filling the correct name of the main file. If it is not the correct name, all the executions of that IPA would fail.

◦ Programing Language Name

- The programing language that the user has developed the new IPA with.
- A mandatory drop-down list. A list of the supported programing languages should be displayed to the user. For now, the list contains: Java, Matlab, C/C++.
- AEIPA should support IPAs written in Java programing languages and can support other programing languages in future work.
- The drop-down list items should accept a maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters.
- This list should enable adding\removing programing languages in future work.

◦ Programing Language Version Number

- The version of the programing Language that the user has developed the new IPA with.

        - A mandatory drop-down list. A list of the supported versions of the selected programing language should be displayed to the user. For example, if the user has selected “Java” in the programing language, then the programing language version list should be filled with the supported versions of Java in AEIPA, the list contains: Java SE 9, Java SE 8, and Java SE7.
        - The drop-down list items should accept a maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters.
        - This list should enable adding\removing programing languages versions in future work.
    - After filling the IPA’s information, user is directed to upload the IPA’s files. Multiple files should be allowed to be uploaded with no limitation on the number of the files.
    - User has to upload all the files using one selection.
    - The IPA’s main file should be uploaded within the uploaded file with the correct name that the user has entered in the Main File Name field.

3.2 Create a New Dataset

Dataset is a collection of frames that represents a specific scene. Dataset can vary from small objects to large ones. Logged in user can create a new dataset using the below fields then uploads the file(s) of the dataset.

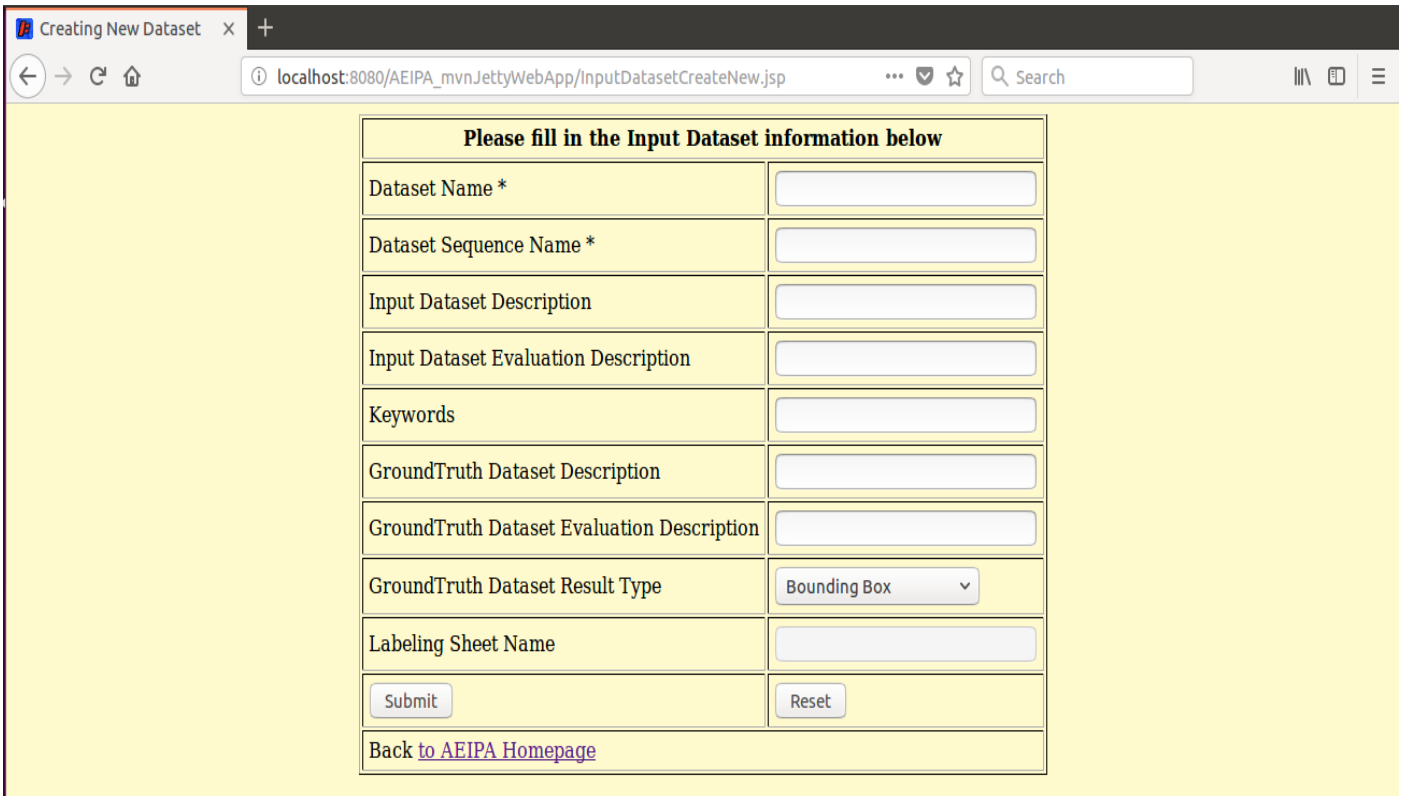


Figure 4.4.8: AEIPA Create a new Dataset Sequence

- Dataset Name
  - Dataset name.
  - A mandatory field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted.
  - This field should be unique across all the datasets in the system.
- Dataset Sequence Name
  - Dataset sequence name.
  - Every dataset can contain one or more dataset sequences

- A mandatory field. A maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted.
- This field should be unique across all the sequences within the same dataset.

- Dataset Description
  - A brief description about the dataset to be uploaded, what it is about, what does the frames contains mainly, what are the main uses of this dataset, ... etc.
  - It is an optional field. A maximum of 200 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted.
- Dataset Evaluation Description
  - A brief description about how this dataset sequence is being evaluated.
  - It is an optional field. A maximum of 200 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted.
- Keywords
  - The most important words that would describe the dataset.
  - This field is used mainly for search purposes in the existing datasets.
  - It is an optional field. A maximum of 200 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted.
- Result Type
  - Dataset evaluation result type.

    - A mandatory drop-down list. A list of the supported types of the dataset should be displayed to the user. Supported evaluation types are: Bounding Box and Labelled Bounding Box.
    - If the Labelled Bounding Box is selected, another field called “XY Label Sheet name” should be enabled so that the user must fill in the sheet file name.
    - The drop-down list items should accept a maximum of 45 characters of any combination of letters, numbers from 0 to 9, and\or special characters.
    - This list should enable adding\removing result types in future work.
- XY Label Sheet Name
    - An optional field. A maximum of 200 characters of any combination of letters, numbers from 0 to 9, and\or special characters are accepted.
    - User is responsible to fill in the correct values, so the execution and evaluation of this dataset doesn’t fail.
    - For each dataset sequence there is an XY Label sheet.
- After filling all this information, user gets directed to upload all the files in the dataset sequence. Multiple files upload is allowed. In case the dataset type is labelled bounding box then the user has to fill in the correct value and upload the file with the dataset frames in each dataset sequence.
- Finally, user is directed to upload the ground truth dataset. Ground truth dataset is the ideal result of that dataset. Every dataset sequence has its own ground truth dataset. User is responsible for matching the files names in the ground truth dataset with the dataset sequence.

3.3 Search among all IPAs

User can search for any IPA(s) in AEIPA, select and execute it (without modifying anything). User can search by IPA name, type, or even using a keyword in the description. When user enters search values, all the IPA(s) with these values should be displayed to the user, even the IPA(s) that the user didn't create. Any user should be able to search and display the IPA(s) created by other users.

Search results is a table that contains:

- Name
- Description
- Type
- Programing Language Name
- Programing Language Version Number
- User Name

User can select among the resulted IPA(s) and proceed in executing them.

3.4 Search among all Datasets

User can search by certain keywords that would describe a dataset, and/or dataset result types. Like the IPA Search use case, user can display any dataset even the one(s) that was created by other users.

3.5 Modify IPA

- User should be able to modify only the IPAs that they had created by themselves.
- User should be able to modify only these IPA's information:
    - Description
    - Type
    - Main File Name
    - Programing Language Name
    - Programing Language Version Number
- User should not be able to modify IPA's name nor the uploaded files' content.

3.6 Modify Dataset

- User should be able to modify only the dataset(s) that they had created by themselves.
- User should be able to modify only these dataset's information:
    - Dataset Description
    - Dataset Evaluation Description
    - Result Type
    - Keywords
- User should not be able to modify dataset's name, dataset sequences names, nor the uploaded files' content.

3.7 Execute IPA

User is able to execute any IPA(s) with any dataset(s) that exist in AEIPA. First, user search among all the IPAs and all the datasets. Then, the user selects the needed IPA(s) and dataset(s) to get executed. Each selected IPA is executed against each selected dataset. While executing an IPA a progress information is displayed to the user.

During the execution, evaluation algorithm(s) should be executed to evaluate the IPA being executed and compare it against the associated ground truth dataset. An evaluation algorithm can be as simple as calculating the confusion matrix (calculating the TNR: True Negative Rate, FPR: False Positive Rate, FNR: False Negative Rate, TPR: True Positive Rate).

Finally, an execution report is displayed at the end of the execution. The execution report contains a table with the confusion matrix for each of the datasets executed using an IPA as depicted below:

Table 1 : Execution Results

Dataset Name	Dataset Sequences	IPA 1				IPA 2			
Dataset 1	Sequence 1	TNR	FPR	FNR	TPR	TNR	FPR	FNR	TPR
	Sequence 2								
	Sequence 3								
Dataset 2	Sequence 1								
Dataset 3	Sequence 1								
	Sequence 2								

4.5. Spotify Docker Client

4.6. Discussion

AEIPA is a novel system that automates the process of evaluating any IPA. Users can: create a new IPA, search for a specific IPA(s), create a new dataset, search for a specific dataset(s), and execute specific IPAs using selected dataset(s) and display a comparing table result. All of this can be done regardless of environment dependencies, IPAs' different programming languages, different programing languages versions, different operating systems, and different hardware.

C++ was selected as the project programming language with the caveat that the library support multiple language bindings, permitting compilation and linking with a range of contemporary computer languages. Java provided better cross-hardware-platform portability, but it was deemed too young a language. Since part of the mission of Insight is to provide an archival mechanism for algorithms, C++ is adequate for programming a library. It has been observed that many of the software libraries still in use today are written in C or even in Fortran; however they serve the purposes of usable APIs and algorithm archives so long as they support multiple language bindings

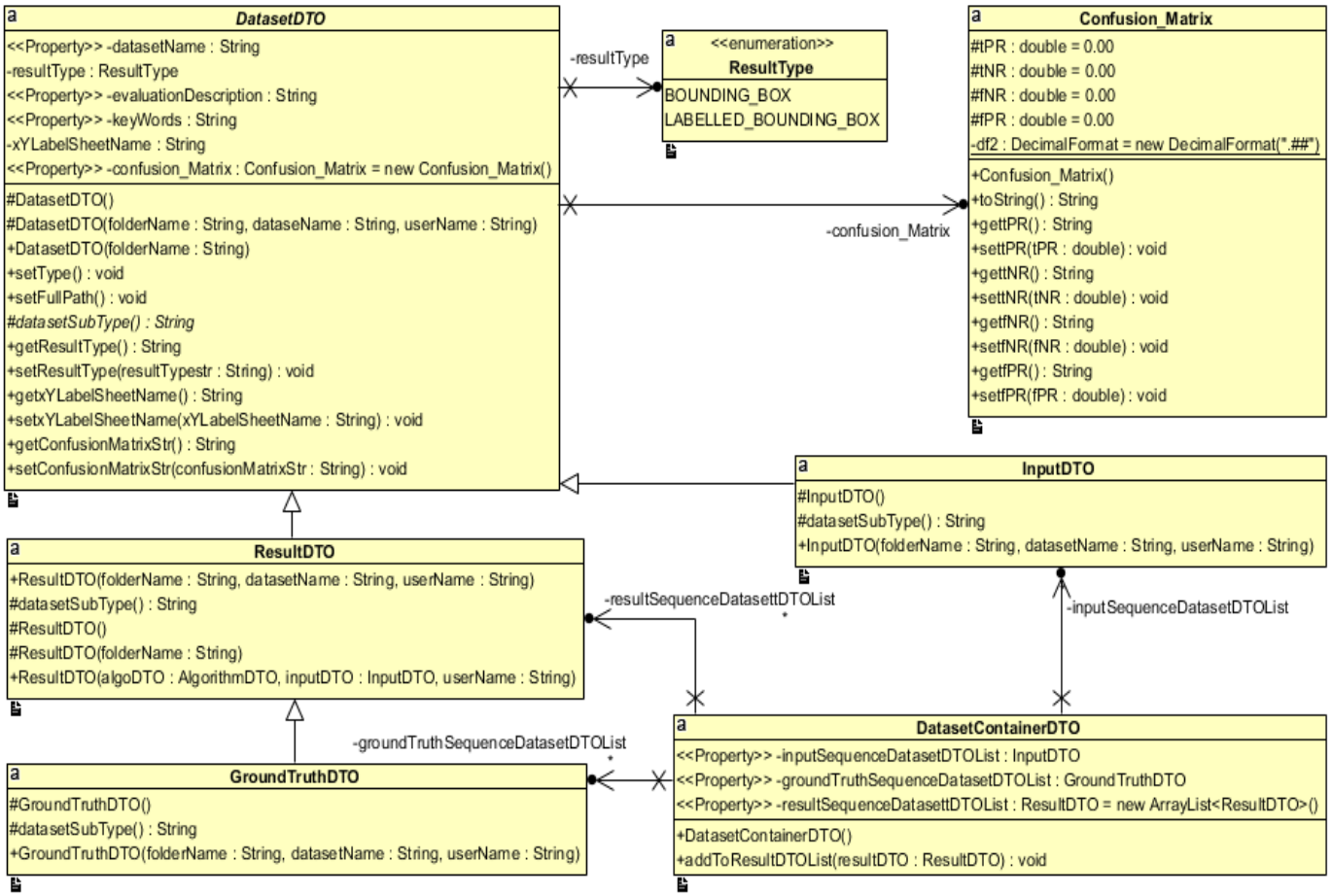


Figure 8: AEIPA Input Supplier Dataset

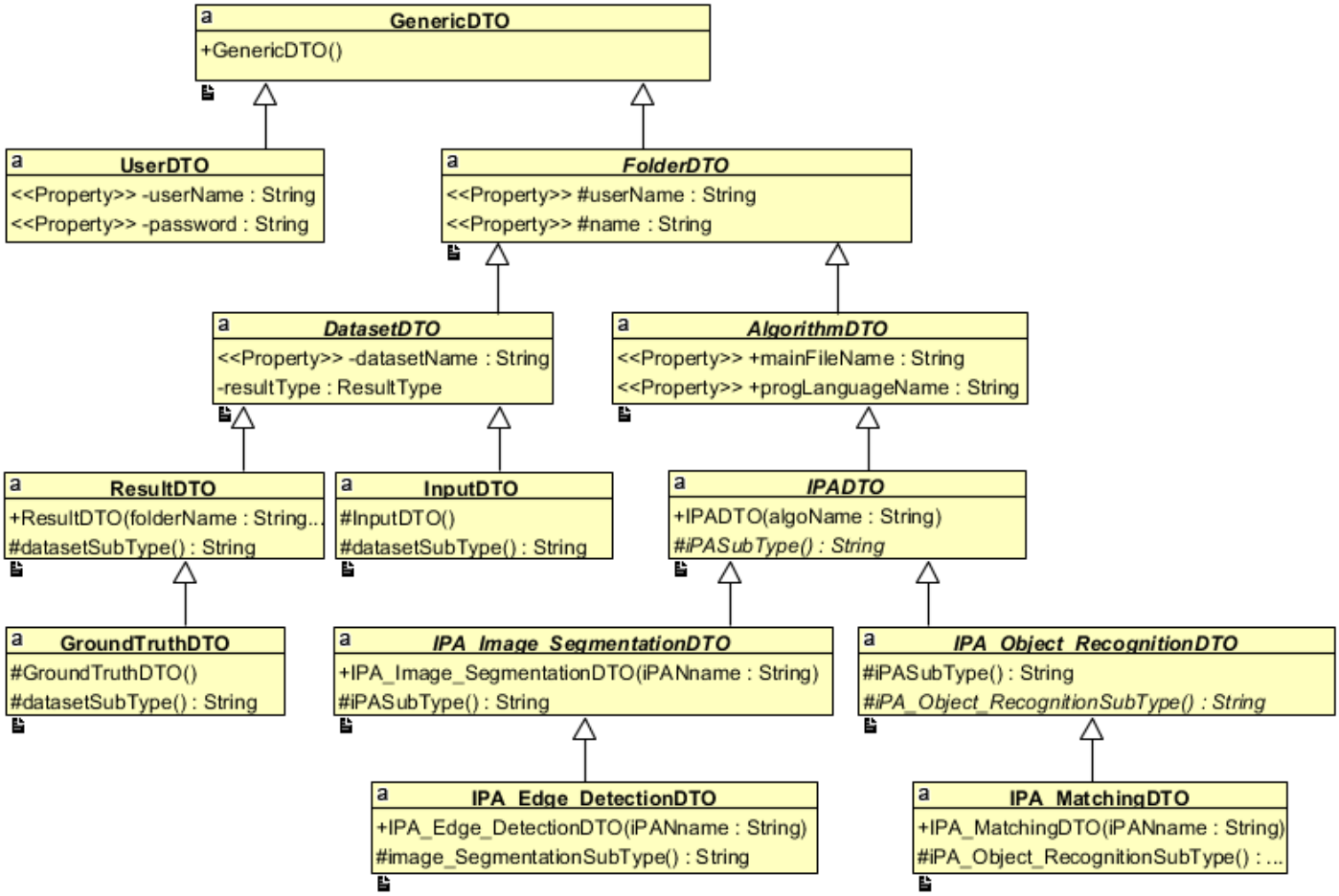


Figure 9 : AEIPA DTOs

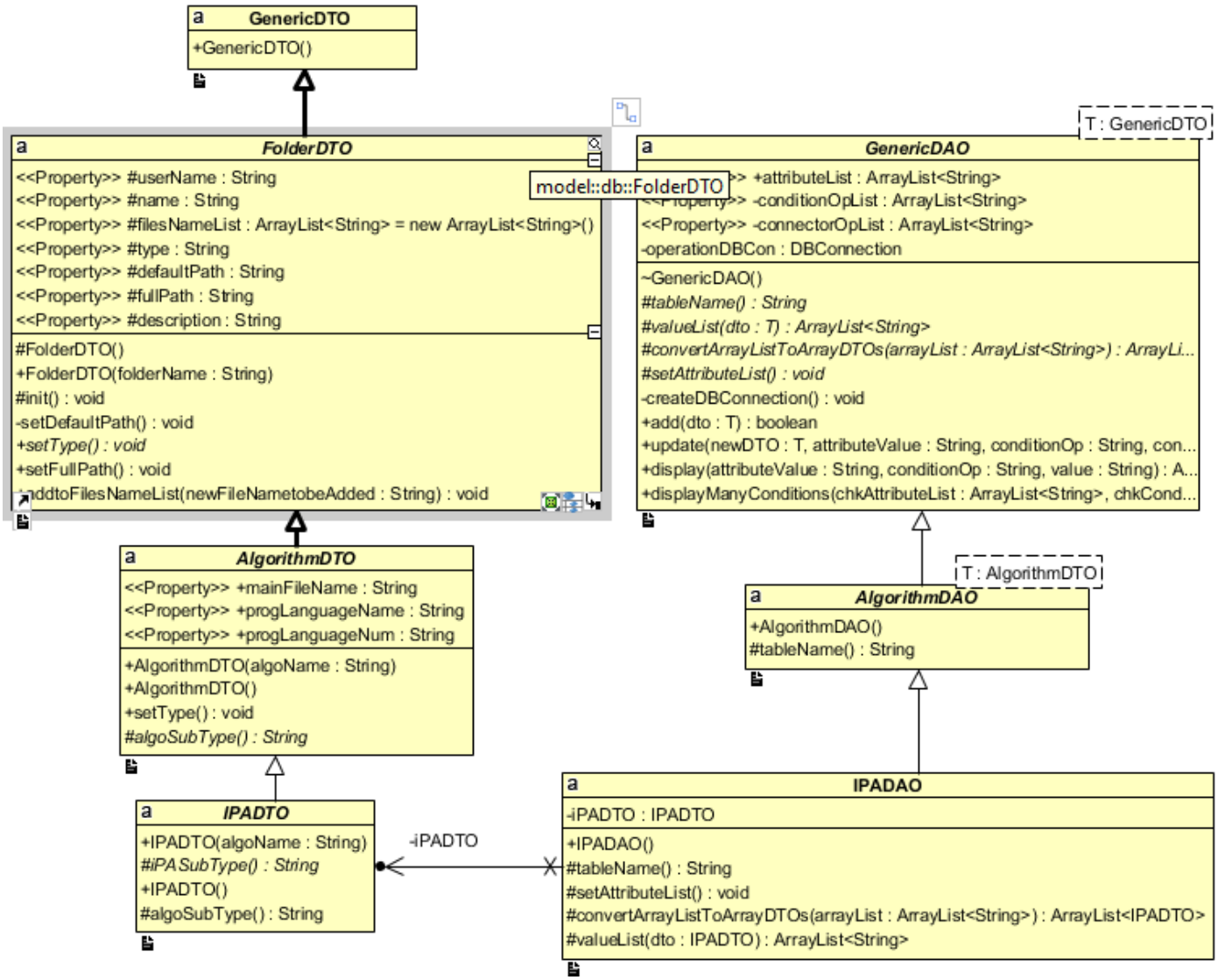
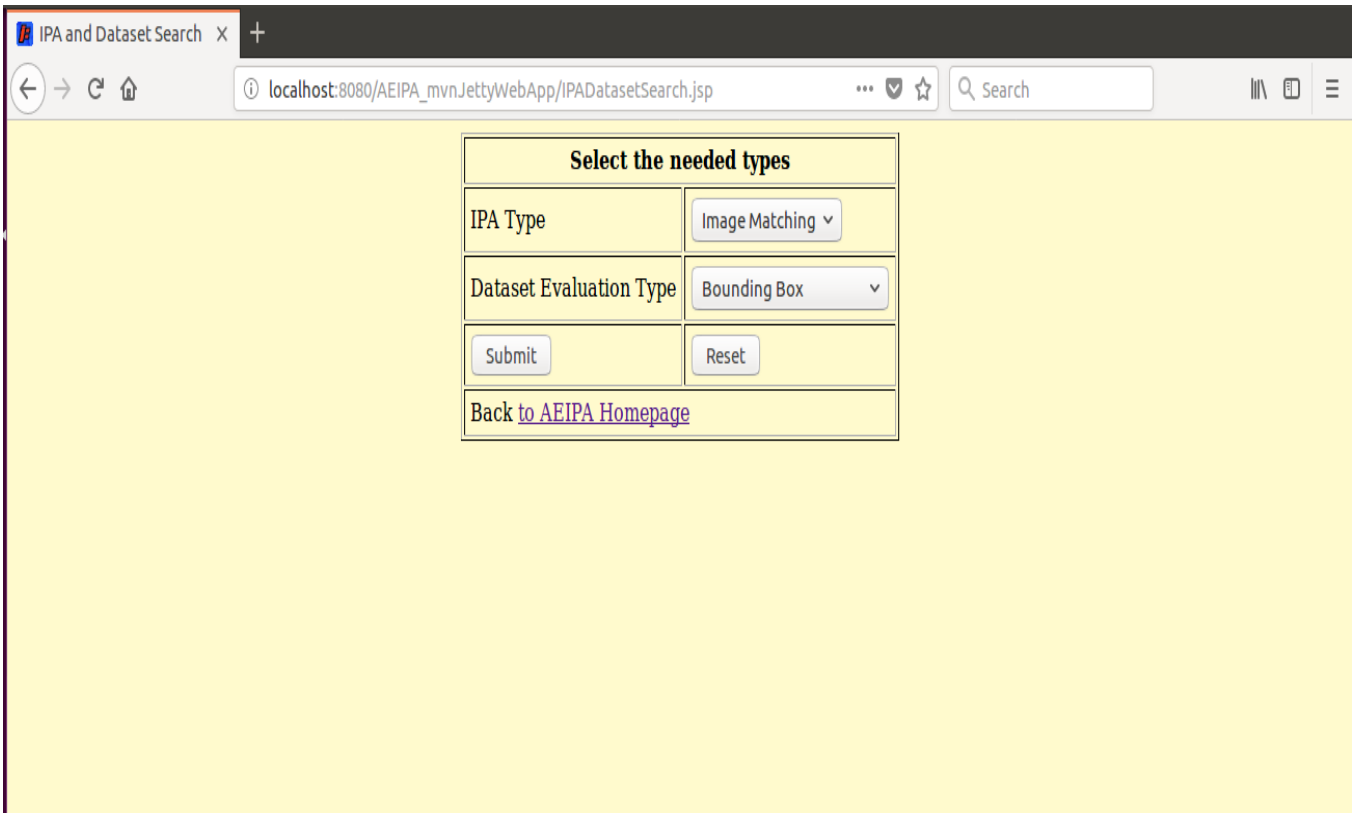
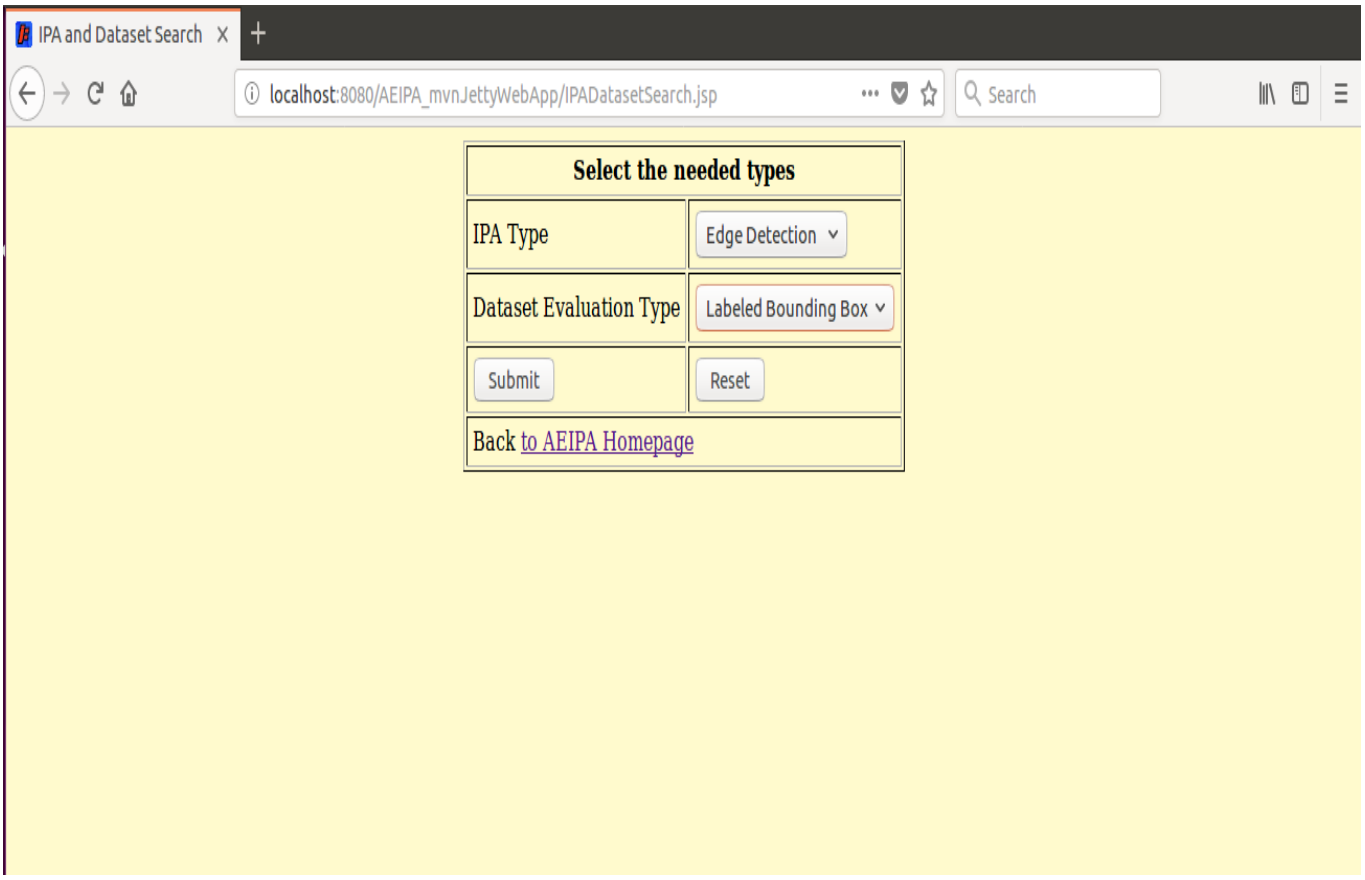
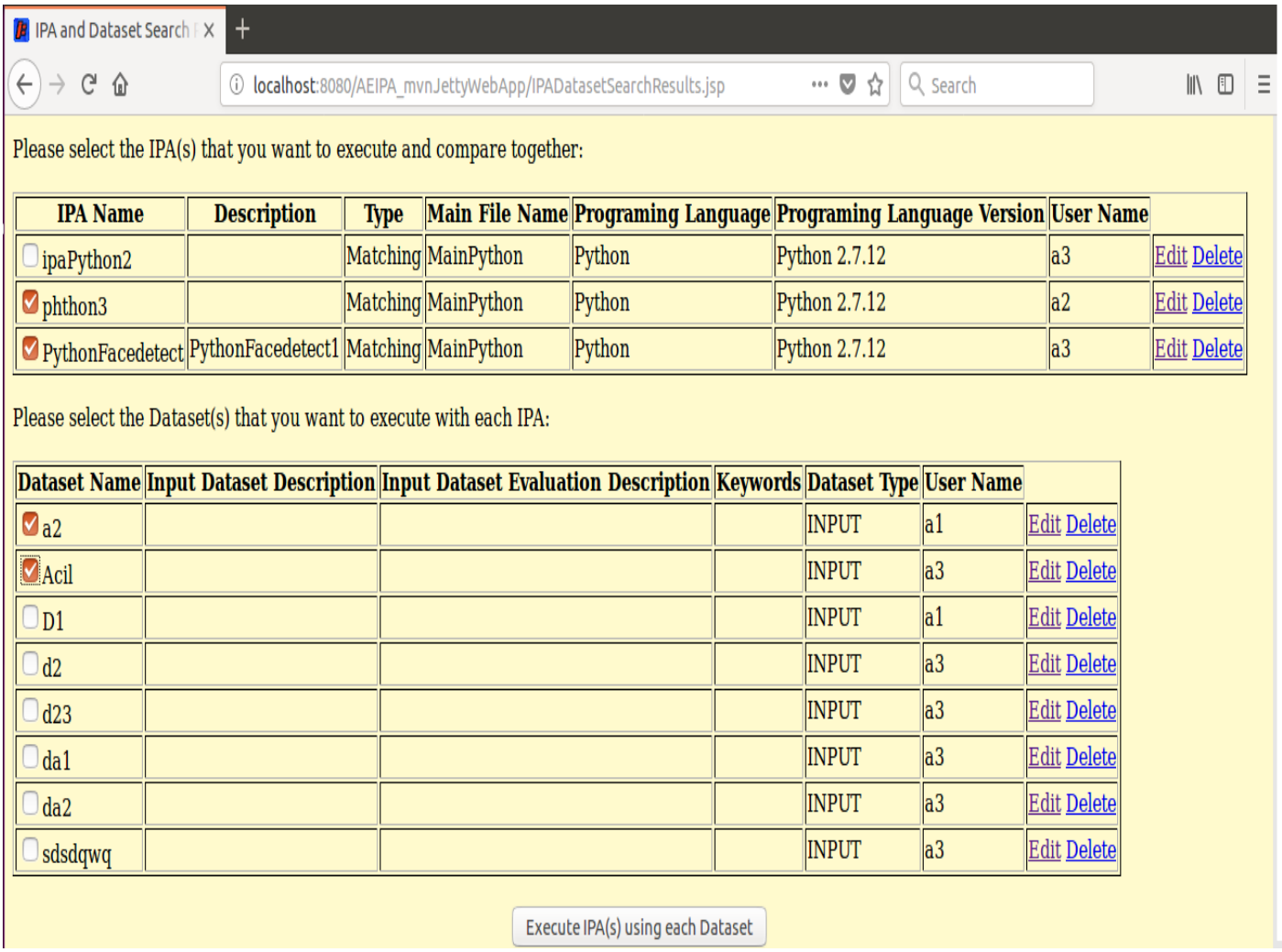
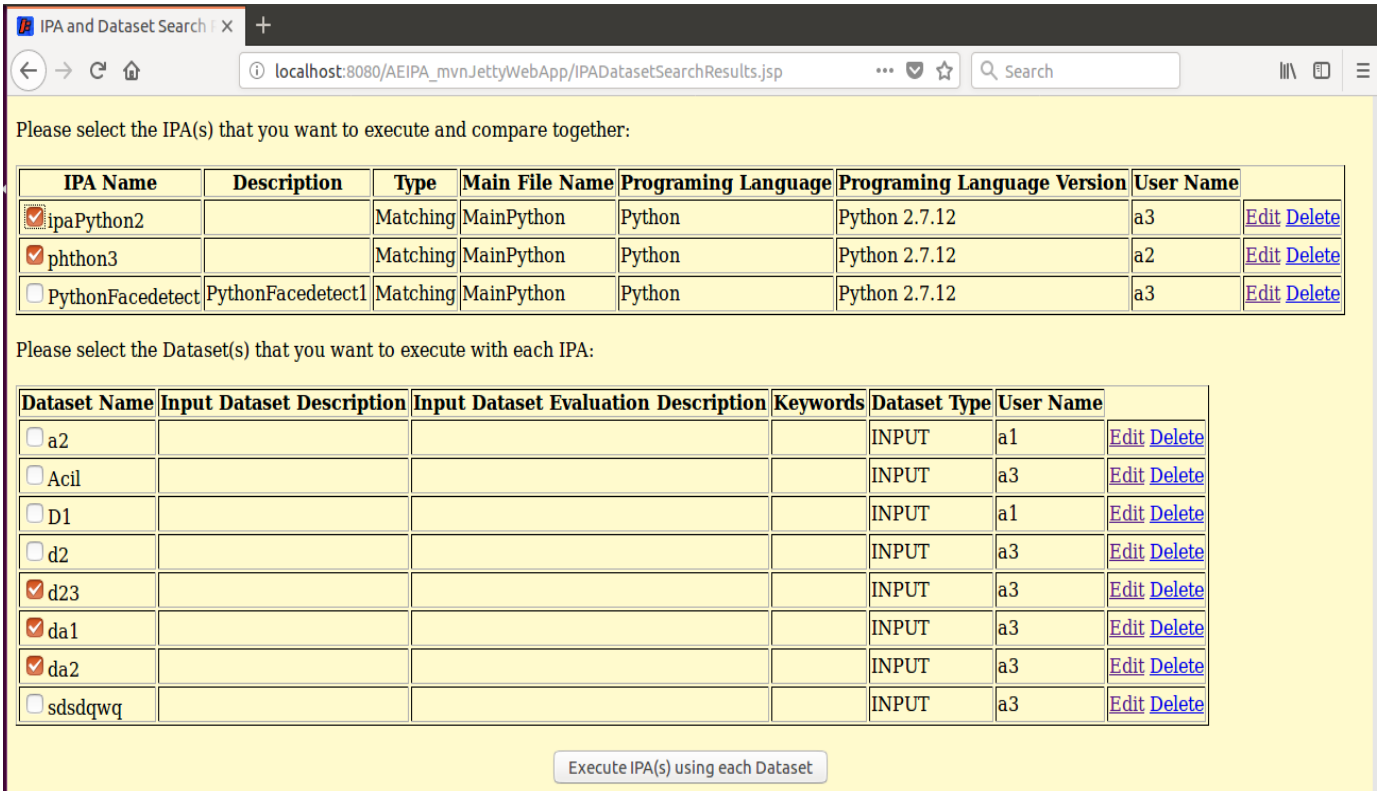


Figure 10 AEIPA DTOs-DAOs relationship


IPA and Dataset Search
localhost:8080/AEIPA_mvnJettyWebApp/IPADatasetSearch.jsp
Search
Select the needed types
IPA Type
Image Matching
Dataset Evaluation Type
Bounding Box
Submit
Reset
Back to AEIPA Homepage


IPA and Dataset Search
localhost:8080/AEIPA_mvnJettyWebApp/IPADatasetSearch.jsp
Search
Select the needed types
IPA Type
Edge Detection
Dataset Evaluation Type
Labeled Bounding Box
Submit
Reset
Back to AEIPA Homepage


IPA and Dataset Search
localhost:8080/AEIPA_mvnJettyWebApp/IPADatasetSearchResults.jsp
Search
Please select the IPA(s) that you want to execute and compare together:
IPA Name | Description | Type | Main File Name | Programing Language | Programing Language Version | User Name
ipaPython2 | Matching | MainPython | Python | Python 2.7.12 | a3 | Edit Delete
phthon3 | Matching | MainPython | Python | Python 2.7.12 | a2 | Edit Delete
PythonFacedetect | PythonFacedetect1 | Matching | MainPython | Python | Python 2.7.12 | a3 | Edit Delete
Please select the Dataset(s) that you want to execute with each IPA:
Dataset Name | Input Dataset Description | Input Dataset Evaluation Description | Keywords | Dataset Type | User Name
a2 | INPUT | a1 | Edit Delete
Acil | INPUT | a3 | Edit Delete
D1 | INPUT | a1 | Edit Delete
d2 | INPUT | a3 | Edit Delete
d23 | INPUT | a3 | Edit Delete
da1 | INPUT | a3 | Edit Delete
da2 | INPUT | a3 | Edit Delete
sdsdqwq | INPUT | a3 | Edit Delete
Execute IPA(s) using each Dataset



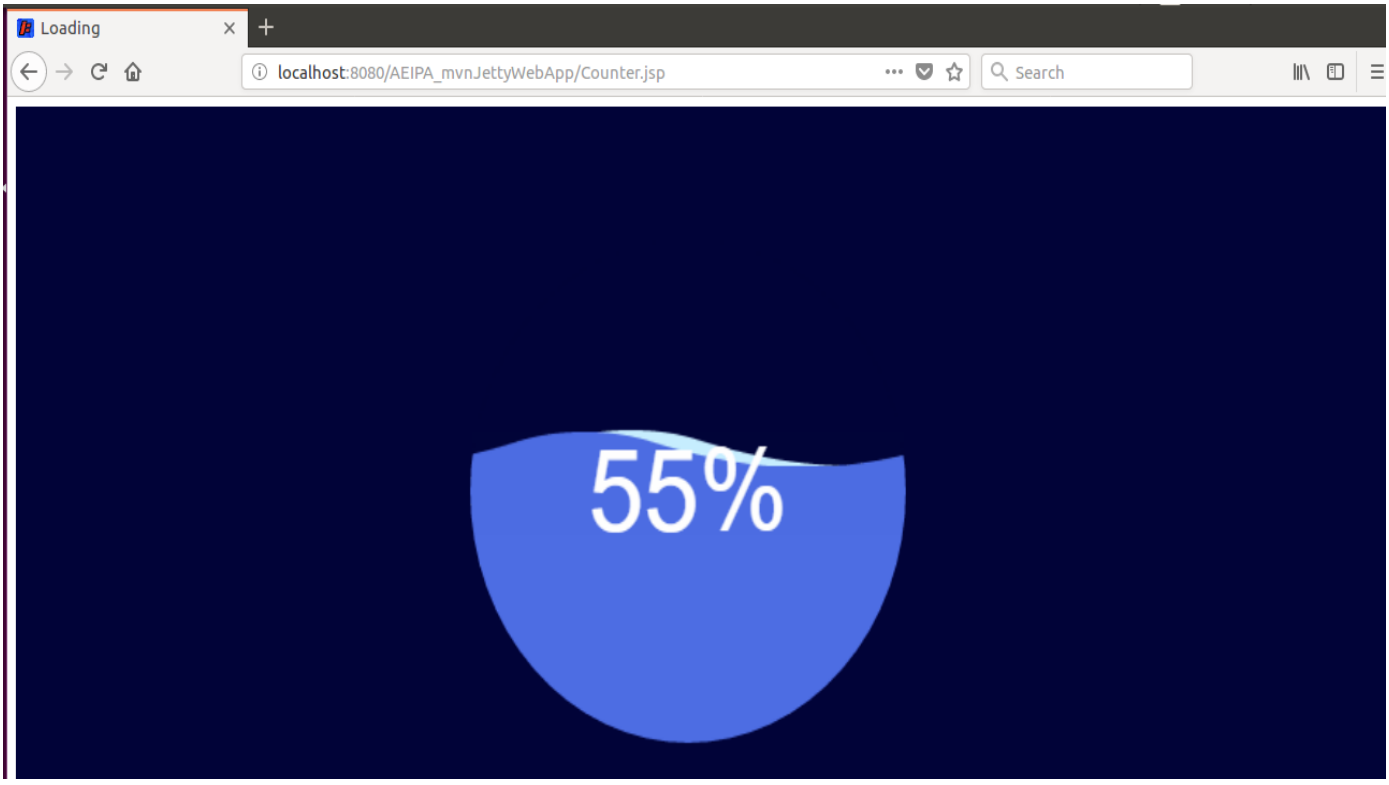
Please select the IPA(s) that you want to execute and compare together:

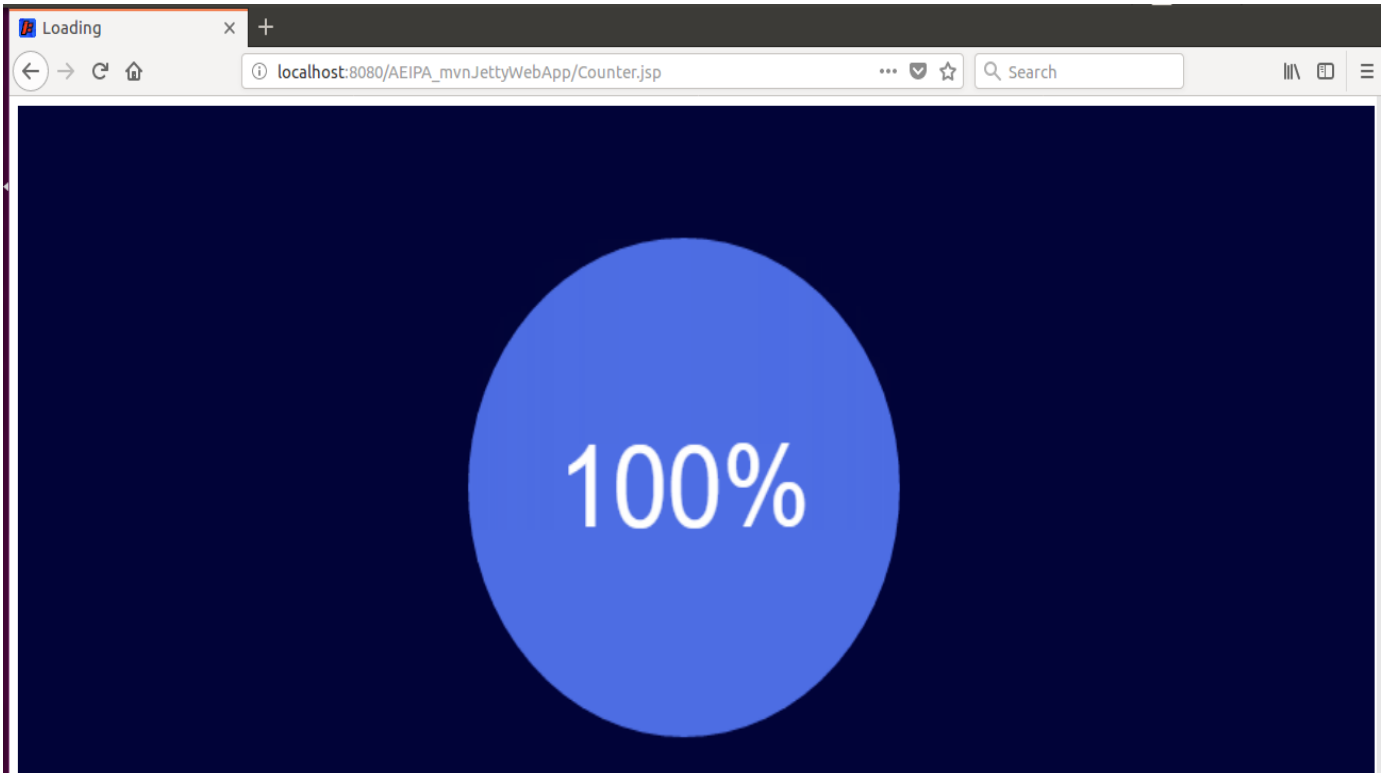
IPA Name	Description	Type	Main File Name	Programing Language	Programing Language Version	User Name	
☑ ipaPython2		Matching	MainPython	Python	Python 2.7.12	a3	Edit Delete
☑ phthon3		Matching	MainPython	Python	Python 2.7.12	a2	Edit Delete
☐ PythonFacedetect	PythonFacedetect1	Matching	MainPython	Python	Python 2.7.12	a3	Edit Delete

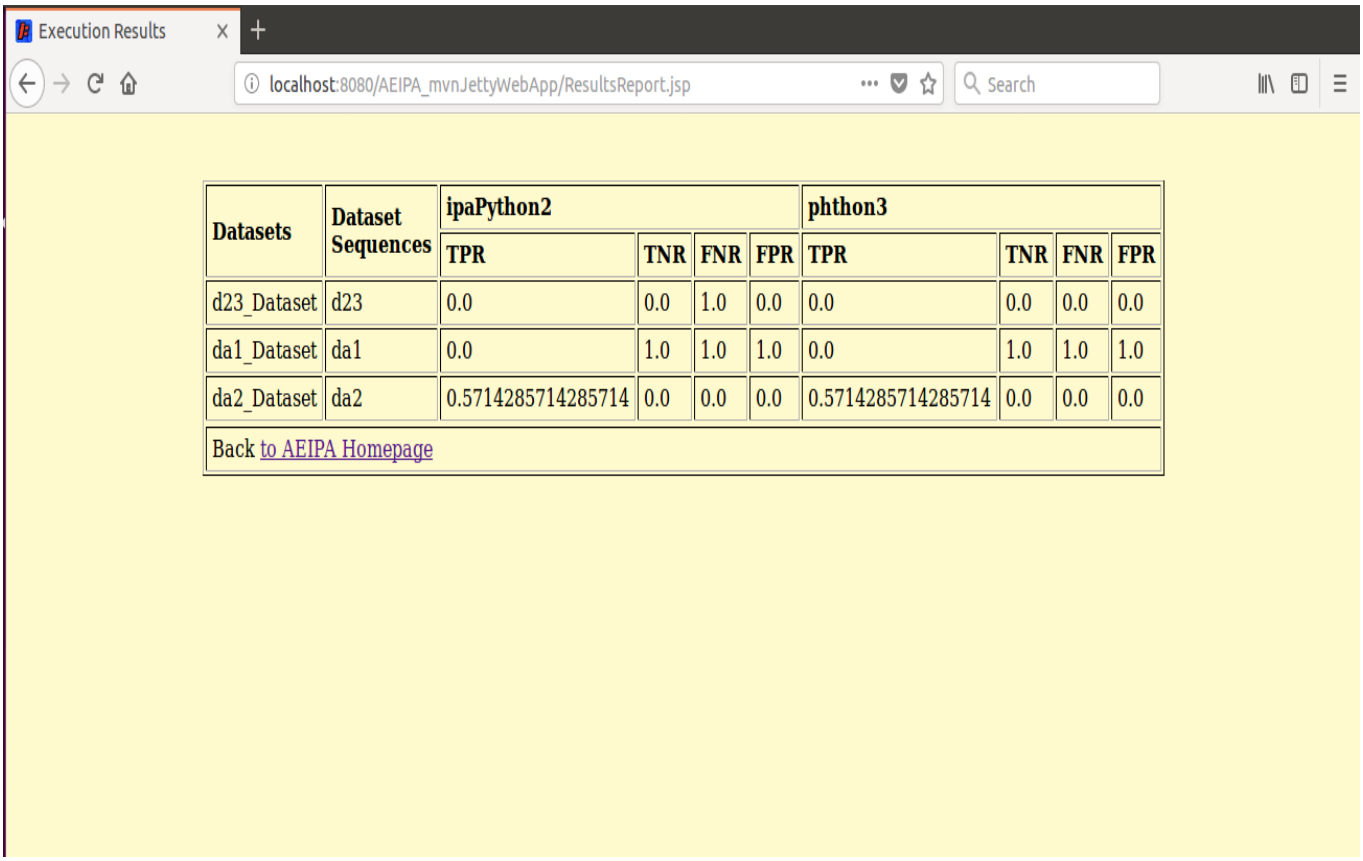
Please select the Dataset(s) that you want to execute with each IPA:

Dataset Name	Input Dataset Description	Input Dataset Evaluation Description	Keywords	Dataset Type	User Name	
☐ a2				INPUT	a1	Edit Delete
☐ Acil				INPUT	a3	Edit Delete
☐ D1				INPUT	a1	Edit Delete
☐ d2				INPUT	a3	Edit Delete
☑ d23				INPUT	a3	Edit Delete
☑ da1				INPUT	a3	Edit Delete
☑ da2				INPUT	a3	Edit Delete
☐ sdsdqwq				INPUT	a3	Edit Delete

Execute IPA(s) using each Dataset




Loading
localhost:8080/AEIPA_mvnJettyWebApp/Counter.jsp
Search
100%



Datasets	Dataset Sequences	ipaPython2				phthon3			
		TPR	TNR	FNR	FPR	TPR	TNR	FNR	FPR
d23_Dataset	d23	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
da1_Dataset	da1	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0
da2_Dataset	da2	0.5714285714285714	0.0	0.0	0.0	0.5714285714285714	0.0	0.0	0.0

Back to AEIPA Homepage

Chapter 5

5. Conclusions and Future Work

5.1. Summary

5.2. System Evaluation

Performance analysis shows that scientific workloads for both Docker and Singularity based containers can achieve near-native performance.

5.3. Research Contributions

5.4. System Limitations

5.5. Generalizability

5.6. Future Work

Chapter 6

6. References

[1] E. Kit, Software Testing in the Real World: Improving the Process, Addison-Wesley, Reading, Mass, USA, 1995.

[2] G. Tassey, “The economic impacts of inadequate infrastructure for software testing,” RTI Project 7007.011, U.S. National Institute of Standards and Technology, Gaithersburg, Md, USA, 2002.

[3] R. Ramler and K. Wolfmaier, “Observations and lessons learned from automated testing,” in Proceedings of the International Workshop on Automation of Software Testing (AST ’06), pp. 85–91, Shanghai, China, May 2006.

[4] K. Karhu, T. Repo, O. Taipale, and K. Smolander, “Empirical observations on software testing automation,” in Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation (ICST ’09), pp. 201–209, Denver, Colo, USA, April 2009.

[5] O. Taipale and K. Smolander, “Improving software testing by observing causes, effects, and associations from practice,” in Proceedings of the International Symposium on Empirical Software Engineering (ISESE ’06), Rio de Janeiro, Brazil, September 2006.

[6] B. Shea, “Sofware testing gets new respect,” InformationWeek, July 2000.

[7] E. Dustin, J. Rashka, and J. Paul, Automated Software Testing: Introduction, Management, and Performance, Addison-Wesley, Boston, Mass, USA, 1999.

[8] S. Berner, R. Weber, and R. K. Keller, “Observations and lessons learned from automated testing,” in Proceedings of the 27th International Conference on Software Engineering (ICSE ’05), pp. 571–579, St. Louis, Mo, USA, May 2005.

[9] J. A. Whittaker, “What is software testing? And why is it so hard?” IEEE Software, vol. 17, no. 1, pp. 70–79, 2000.

[10] L. J. Osterweil, “Software processes are software too, revisited: an invited talk on the most influential paper of ICSE 9,” in Proceedings of the 19th IEEE International Conference on Software Engineering, pp. 540–548, Boston, Mass, USA, May 1997.

[11] J. Kasurinen, O. Taipale, K. Smolander, “Software Test Automation in Practice: Empirical Observations,” Advances in Software Engineering, vol. 2010, Article 4 (January 2010), 13 pages. DOI=10.1155/2010/620836.

[12] Ian Sommerville. 2004. Software Engineering (7th Edition). Pearson Addison Wesley.

[13] D. Powers, Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation, Journal of Machine Learning Technologies, vol. 2, no. 1, pp. 3763, 2011.

[14] Steven S. Skiena. 2008. The Algorithm Design Manual (2nd ed.). Springer Publishing Company, Incorporated.

[15] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd ed.). The MIT Press.

[16] Karsten Weihe. 1997. Reuse of algorithms: still a challenge to object-oriented programming. In Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '97), A. Michael Berman (Ed.). ACM, New York, NY, USA, 34-48. DOI=http://dx.doi.org.qe2a-proxy.mun.ca/10.1145/263698.263704

[17] Padhy, Neelamadhab & Singh, R.P. & Satapathy, Suresh. (2017). Software reusability metrics estimation: Algorithms, models and optimization techniques. Computers & Electrical Engineering. 69. 10.1016/j.compeleceng.2017.11.022.

[18] R. Dua, A. R. Raja, and D. Kakadia, ”Virtualization vs containerization to support paas,” in Cloud Engineering (IC2E), 2014 IEEE International Conference on, 2014, pp. 610-614.

[19] T. Bui, ”Analysis of docker security,” arXiv preprint arXiv:1501.02967, 2015.

[20] A. Azab, "Enabling Docker Containers for High-Performance and Many-Task Computing," *2017 IEEE International Conference on Cloud Engineering (IC2E)*, Vancouver, BC, 2017, pp. 279-285. doi: 10.1109/IC2E.2017.52

[21] Pankaj Saha, Angel Beltre, Piotr Uminski, and Madhusudhan Govindaraju. 2018. Evaluation of Docker Containers for Scientific Workloads in the Cloud. In Proceedings of the Practice and Experience on Advanced Research Computing (PEARC '18). ACM, New York, NY, USA, Article 11, 8 pages.

[22] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. Linux Journal 2014, 239 (2014), 2.

[23] A. Naby, M. S. Shehata and T. S. Norvell, "AEIPA: Docker-based system for Automated Evaluation of Image Processing Algorithms," *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, Windsor, ON, 2017, pp. 1-4.

[24] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107-113.

[25] J. Albrecht and J. E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," in *IEEE Transactions on Software Engineering*, vol. SE-9, no. 6, pp. 639-648, Nov. 1983.

[26] K. Christensen, G. P. Fitsos, and C. P. Smith, "A perspective on software science," IBM Syst. J., vol. 20, no. 4, pp. 372-387, 1981.

[27] OSMAN, Hafeez; CHAUDRON, Michel R.V.. Correctness and Completeness of CASE Tools in Reverse EngineeringSource Code into UML Model. GSTF Journal on Computing (JoC), [S.l.], v. 2, n. 1, Jan. 2018. ISSN 2010-2283.

[28] Robert C. Martin. 2000. The open-closed principle. In More C++ gems, Robert C. Martin (Ed.). Cambridge University Press, New York, NY, USA 97-112.

[29] A. Eberlein, G. Succi and J. W. Paulson, "An Empirical Study of Open-Source and Closed-Source Software Products," in *IEEE Transactions on Software Engineering*, vol. 30, no. , pp. 246-256, 2004. doi:10.1109/TSE.2004.1274044.

[30] R. S. Freedman, "Testability of software components," in *IEEE Transactions on Software Engineering*, vol. 17, no. 6, pp. 553-564, June 1991. doi: 10.1109/32.87281.

[31] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and Linux containers,” in Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw., 2015, pp. 171–172.

[32] R. Dua, A. R. Raja, and D. Kakadia, “Virtualization versus containerization to support PaaS,” in Proc. IEEE Int. Conf. Cloud Eng., 2014, pp. 610–614.

[33] J. Bhimani *et al*., "Docker Container Scheduler for I/O Intensive Applications Running on NVMe SSDs," in *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 3, pp. 313-326, 1 July-Sept. 2018. doi: 10.1109/TMSCS.2018.2801281

[34] C. Anderson, “Docker software engineering,” IEEE Softw., vol. 32, no. 3, p. 102–c3, 2015.

[35] ~~P. Di Tommaso, E. Palumbo, M. Chatzou, P. Prieto, M. L. Heuer, and C. Notredame, “The impact of docker containers on the performance of genomic pipelines,” PeerJ, vol. 3, 2015, Art. no. e1273~~.

[36] J. Fink, “Docker: A software as a service, operating system-level virtualization framework,” Code4Lib J., vol. 25, p. 29, 2014.

[37] A. M. Joy, "Performance comparison between Linux containers and virtual machines," *2015 International Conference on Advances in Computer Engineering and Applications*, Ghaziabad, 2015, pp. 342-346. doi: 10.1109/ICACEA.2015.7164727.

[38] W. Felter, A. Ferreira, R. Rajamony and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," *2015 IEEE International*

Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, 2015, pp. 171-172.

[39] S. Kamijo, Y. Matsushita, K. Ikeuchi, and M. Sakauchi, ”Traffic monitoring and accident detection at intersections,” IEEE transactions on Intelligent transportation systems, vol. 1, pp. 108-118, 2000.

[40] T. Brosnan and D.-W. Sun, ”Improving quality inspection of food products by computer vision - a review,” Journal of food engineering, vol. 61, pp. 3-16, 2004.

[41] M. Trajkovic, A. J. Colmenarez, S. Gutta, and K. I. Trovato, ”Computer vision based parking assistant,” ed: Google Patents, 2004.

[42] W. W. Boles and B. Boashash, ”A human identification technique using images of the iris and wavelet transform,” IEEE transactions on signal processing, vol. 46, pp. 1185-1188, 1998.

[43] CVonline: Image Databases. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/Imagedbase.htm

[44] M. J. McAuliffe, F. M. Lalonde, D. McGarry, W. Gandler, K. Csaky and B. L. Trus, "Medical Image Processing, Analysis and Visualization in clinical research," *Proceedings 14th IEEE Symposium on Computer-Based Medical Systems. CBMS 2001*, Bethesda, MD, USA, 2001, pp. 381-386.

[45] A. Leff and J. T. Rayfield, "Web-application development using the Model/View/Controller design pattern," *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, Seattle, WA, USA, 2001, pp. 118-127.

[46] E. Curry and P. Grace, "Flexible Self-Management Using the Model-View-Controller Pattern," in *IEEE Software*, vol. 25, no. 3, pp. 84-90, May-June 2008.

[47] J. Wojciechowski, B. Sakowicz, K. Dura and A. Napieralski, "MVC model, struts framework and file upload issues in web applications based on J2EE platform," *Proceedings of the International Conference Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2004.*, Lviv-Slavsko, Ukraine, 2004, pp. 342-345.

[48] Nikolas Havrikov, Alessio Gambi, Andreas Zeller, Andrea Arcuri, and Juan Pablo Galeotti. 2017. Generating unit tests with structured system interactions. In Proceedings of the 12th International Workshop on Automation of Software Testing (AST '17). IEEE Press, Piscataway, NJ, USA, 30-33.

[49] M. Rodríguez-Martinez, "Experiences with the Twitter Health Surveillance (THS) System," *2017 IEEE International Congress on Big Data (BigData Congress)*, Honolulu, HI, 2017, pp. 376-383.

[50] K. Tüzes-Bölöni, Z. Borsay, C. Sulyok and K. Simon, "Diaspora Mapping and Collaboration Platform for Expatriates," *2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)*, Subotica, 2018, pp. 000027-000032.

[51] H. Pu, S. Su, M. Lee and C. Lin, "Generation of personalized learning paths with a concept map: A case study of the IEEE floating-point standard," *2018 IEEE International Conference on Applied System Invention (ICASI)*, Chiba, 2018, pp. 184-187.

[52] C. Pan and C. Lin, "Designing and implementing a computerized adaptive testing system with an MVC framework: A case study of the IEEE floating-point standard," *2018*

IEEE International Conference on Applied System Invention (ICASI), Chiba, 2018, pp. 609-612.

[53] Anurag Dwarakanath, Upendra Chintala, Shrikanth N. C., Gurdeep Virdi, Alex Kass, Anitha Chandran, Shubhashis Sengupta, and Sanjoy Paul. 2015. CrowdBuild: a methodology for enterprise software development using crowdsourcing. In Proceedings of the Second International Workshop on CrowdSourcing in Software Engineering (CSI-SE '15). IEEE Press, Piscataway, NJ, USA, 8-14.

[54] G. Orsini, D. Bade and W. Lamersdorf, "Computing at the Mobile Edge: Designing Elastic Android Applications for Computation Offloading," *2015 8th IFIP Wireless and Mobile Networking Conference (WMNC)*, Munich, 2015, pp. 112-119.

[55] M. Shetty and S. R. Naik, "Ontology representation of Amazon S3 objects for metadata enrichment," *2016 International Conference on Computing, Analytics and Security Trends (CAST)*, Pune, 2016, pp. 507-512.

[56] Verginadis, Y., Michalas, A., Gouvas, P. et al. J Grid Computing (2017) 15: 219. https://doi.org/10.1007/s10723-017-9394-2

[57] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," in *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347-2376, Fourthquarter 2015.

[58] Manjunath R., Tejus, Channabasava R.K and Balaji S., "A Big Data MapReduce Hadoop distribution architecture for processing input splits to solve the small data problem," *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, Bangalore, 2016, pp. 480-487.

[59] Wenwen Li, Sizhe Wang, Vidit Bhatia, PolarHub: A large-scale web crawling engine for OGC service discovery in cyberinfrastructure, Computers, Environment and Urban Systems, Volume 59, 2016, Pages 195-207, ISSN 0198-9715.

Appendix

A. Haarcascade Face Detection Application using C and OpenCV

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
 #include "opencv2/imgproc/imgproc.hpp"
#include <fstream>
 #include <iostream>
 #include <stdio.h>

 using namespace std;
 using namespace cv;

 /** Function Headers */
 void detectAndDisplay( String outputFramePath, Mat frame , const char* outputtxtFilePath);

 /** Global variables */
 String face_cascade_name = "haarcascade_frontalface_alt.xml";
 //String eyes_cascade_name = "haarcascade_eye_tree_eyeglasses.xml";
 CascadeClassifier face_cascade;
 //CascadeClassifier eyes_cascade;
// string window_name = "Capture - Face detection";
// RNG rng(12345);

 /** @function main */
```

```
int main( int argc, const char** argv )
{
  CvCapture* capture;
  Mat frame;

  //-- 1. Load the cascades
  if( !face_cascade.load( face_cascade_name ) ){ printf("--(!)Error loading haarcascade_frontalface_alt.xml\n"); return -1; };
 //  if( !eyes_cascade.load( eyes_cascade_name ) ){ printf("--(!)Error loading haarcascade_eye_tree_eyeglasses.xml\n"); return -1; };
//printf("Hello Acil");
 frame = imread(argv[1], CV_LOAD_IMAGE_COLOR); // here we'll know the method used (allocate matrix)
detectAndDisplay(argv[2], frame, argv[3]);
  return 0;
 }

/** @function detectAndDisplay */
void detectAndDisplay(String outputFramePath , Mat frame, const char* outputtxtFilePath)
{
  std::vector<Rect> faces;
  Mat frame_gray;

  cvtColor( frame, frame_gray, CV_BGR2GRAY );
  equalizeHist( frame_gray, frame_gray );

  //-- Detect faces
```

```
 face_cascade.detectMultiScale( frame_gray, faces, 1.1, 2, 0|CV_HAAR_SCALE_IMAGE, Size(30, 30) );

std::ofstream myfile;

 myfile.open (outputtxtFilePath, std::ios_base::app);

 if (myfile.is_open())

 {

 for( size_t i = 0; i < faces.size(); i++ )

 {

  // Point center( faces[i].x + faces[i].width*0.5, faces[i].y + faces[i].height*0.5 );

   //ellipse( frame, center, Size( faces[i].width*0.5, faces[i].height*0.5), 0, 0, 360, Scalar( 255, 0, 255 ), 4, 8, 0 );

          Rect r=Rect(faces[i].x,faces[i].y,faces[i].width, faces[i].height);

          rectangle(frame, r,Scalar( 255, 0, 255 ), 4, 8, 0  );

          Mat faceROI = frame_gray( faces[i] );

myfile << faces[i].x << " " <<  faces[i].y << " " << faces[i].width << " " << faces[i].height <<"\n";

 /*       std::vector<Rect> eyes;


   //-- In each face, detect eyes

   eyes_cascade.detectMultiScale( faceROI, eyes, 1.1, 2, 0 |CV_HAAR_SCALE_IMAGE, Size(30, 30) );


  for( size_t j = 0; j < eyes.size(); j++ )

    {

     Point center( faces[i].x + eyes[j].x + eyes[j].width*0.5, faces[i].y + eyes[j].y + eyes[j].height*0.5 );

     int radius = cvRound( (eyes[j].width + eyes[j].height)*0.25 );

     circle( frame, center, radius, Scalar( 255, 0, 0 ), 4, 8, 0 );
```

```
    }
*/
 }

//myfile << "----------------------------\n";
   myfile.close();
 }
 else cout << "Unable to open the output file";
 imwrite(outputFramePath,frame);

 //-- Show what you got
 //imshow( window_name, frame );
}
```

B. A Sample for the haarcascade_frontalface_alt.xml

The xml file needed for the previous Software Application for face detection using haarcascade using C and OpenCV. These are only parts of the file as it is a huge file.

```
<opencv_storage>
<cascade type_id="opencv-cascade-classifier"><stageType>BOOST</stageType>
 <featureType>HAAR</featureType>
 <height>20</height>
 <width>20</width>
 <stageParams>
  <maxWeakCount>213</maxWeakCount></stageParams>
 <featureParams>
  <maxCatCount>0</maxCatCount></featureParams>
 <stageNum>22</stageNum>
 <stages>
  <_>
   <maxWeakCount>3</maxWeakCount>
   <stageThreshold>8.2268941402435303e-01</stageThreshold>
   <weakClassifiers>
    <_>
     <internalNodes>
      0 -1 0 4.0141958743333817e-03</internalNodes>
     <leafValues>
      3.3794190734624863e-02 8.3781069517135620e-01</leafValues></_>
    <_>
     <internalNodes>
      0 -1 1 1.5151339583098888e-02</internalNodes>
     <leafValues>
      1.5141320228576660e-01 7.4888122081756592e-01</leafValues></_>
    <_>
     <internalNodes>
      0 -1 2 4.2109931819140911e-03</internalNodes>
     <leafValues>
```

9.0049281716346741e-02 6.3748198747634888e-01</leafValues></_></weakClassifiers></_>

```
<_>
 <maxWeakCount>16</maxWeakCount>
 <stageThreshold>6.9566087722778320e+00</stageThreshold>
 <weakClassifiers>
  <_>
   <internalNodes>
    0 -1 3 1.6227109590545297e-03</internalNodes>
   <leafValues>
    6.9308586418628693e-02 7.1109461784362793e-01</leafValues></_>
  <_>
   <internalNodes>
    0 -1 4 2.2906649392098188e-03</internalNodes>
   <leafValues>
    1.7958030104637146e-01 6.6686922311782837e-01</leafValues></_>
  <_>
   <internalNodes>
    0 -1 5 5.0025708042085171e-03</internalNodes>
   <leafValues>
    1.6936729848384857e-01 6.5540069341659546e-01</leafValues></_>
```

.

.

.

.

.

```
<_>
 <rects>
  <_>
   4 3 8 3 -1.</_>
  <_>
   8 3 4 3 2.</_></rects></_>
<_>
 <rects>
  <_>
```

```
      0 4 20 6 -1.</_>
     <_>
      0 4 10 6 2.</_></rects></_>
   <_>
    <rects>
     <_>
      9 14 1 3 -1.</_>
     <_>
      9 15 1 1 3.</_></rects></_>
   <_>
    <rects>
     <_>
      8 14 4 3 -1.</_>
     <_>
      8 15 4 1 3.</_></rects></_>
   <_>
    <rects>
     <_>
      0 15 14 4 -1.</_>
     <_>
      0 17 14 2 2.</_></rects></_>
   <_>
    <rects>
     <_>
      1 14 18 6 -1.</_>
     <_>
      1 17 18 3 2.</_></rects></_>
   <_>
    <rects>
     <_>
      0 0 10 6 -1.</_>
     <_>
      0 0 5 3 2.</_>
     <_>
      5 3 5 3 2.</_></rects></_></features></cascade>
</opencv_storage>
```

C. Result of Haarcascade Face Detection Application using C and OpenCV

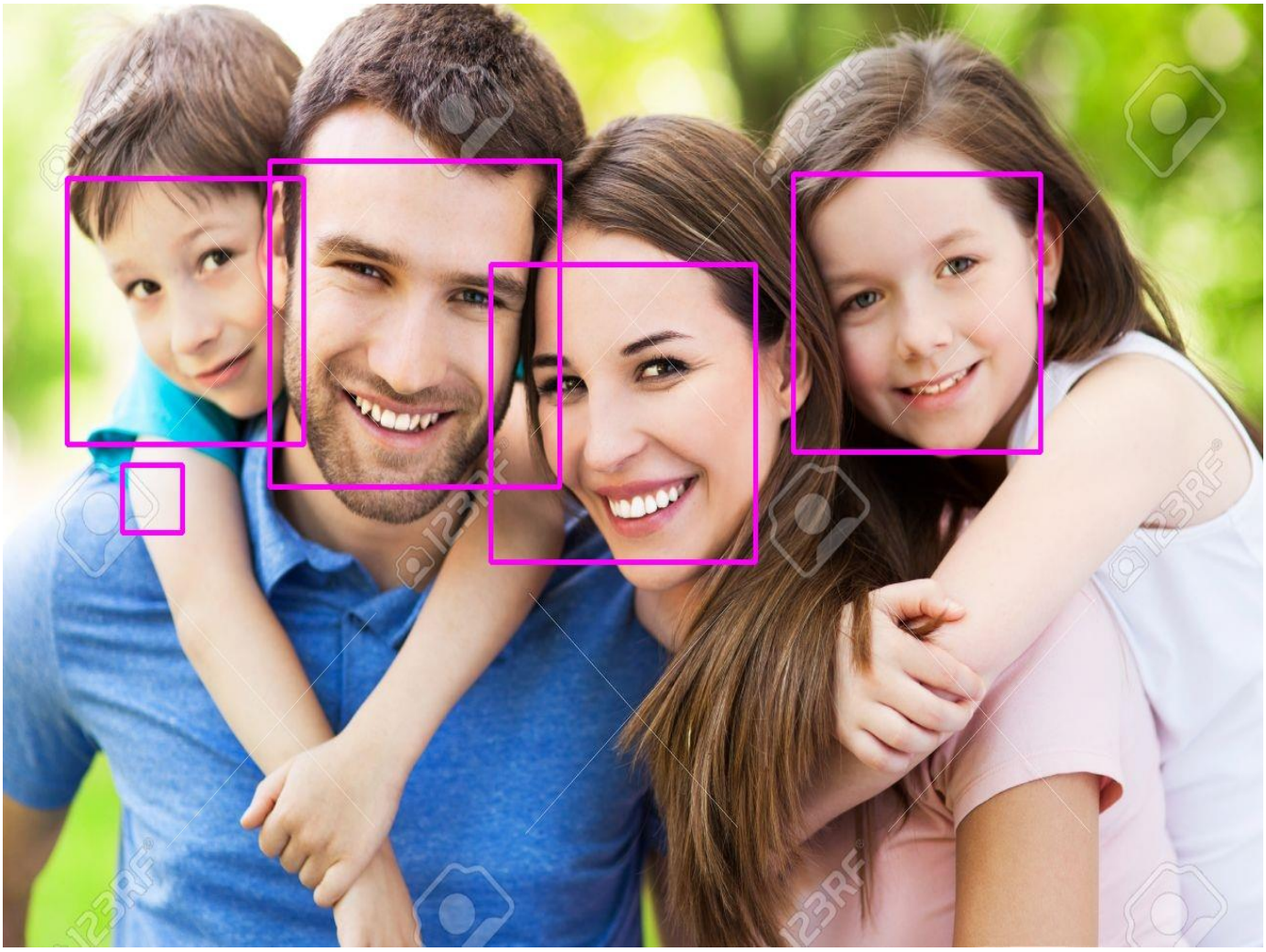


Figure C: Result of Haarcascade Face Detection Application using C and OpenCV

D. Haarcascade Face Detection Application using Python

```
import cv2
import sys

# Get user supplied values
imagePath = sys.argv[1]
cascPath = "haarcascade_frontalface_default.xml"

# Create the haar cascade
faceCascade = cv2.CascadeClassifier(cascPath)

# Read the image
image = cv2.imread(imagePath)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect faces in the image
faces = faceCascade.detectMultiScale(
    gray,
    scaleFactor=1.1,
    minNeighbors=5,
    minSize=(30, 30),
    flags = cv2.cv.CV_HAAR_SCALE_IMAGE
)
```

```
print("Found {0} faces!".format(len(faces)))
f = open(sys.argv[3],"a+b")
#f = open("output.txt","a+b")
# Draw a rectangle around the faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
    f.write("%d %d %d %d \r\n" % (x , y, w, h) )
    #cv2.circle(image, (x+w/2, y+h/2), (h/2), (0, 255, 0), 3)
#cv2.imwrite("imageoutput.png", image);

cv2.imwrite(sys.argv[2], image);
#f.write("-----------------------------\n");
#cv2.imshow("Faces found", image)


f.close()
cv2.waitKey(0)
```

E. A Sample of haarcascade_frontalface_default.xml


```
<opencv_storage>
<cascade type_id="opencv-cascade-classifier"><stageType>BOOST</stageType>
 <featureType>HAAR</featureType>
 <height>24</height>
 <width>24</width>
 <stageParams>
  <maxWeakCount>211</maxWeakCount></stageParams>
 <featureParams>
  <maxCatCount>0</maxCatCount></featureParams>
 <stageNum>25</stageNum>
 <stages>
  <_>
   <maxWeakCount>9</maxWeakCount>
   <stageThreshold>-5.0425500869750977e+00</stageThreshold>
   <weakClassifiers>
    <_>
     <internalNodes>
      0 -1 0 -3.1511999666690826e-02</internalNodes>
     <leafValues>
      2.0875380039215088e+00 -2.2172100543975830e+00</leafValues></_>
    <_>
     <internalNodes>
      0 -1 1 1.2396000325679779e-02</internalNodes>
     <leafValues>
      -1.8633940219879150e+00 1.3272049427032471e+00</leafValues></_>
    <_>
     <internalNodes>
      0 -1 2 2.1927999332547188e-02</internalNodes>
     <leafValues>
      -1.5105249881744385e+00 1.0625729560852051e+00</leafValues></_>
    <_>
     <internalNodes>
      0 -1 3 5.7529998011887074e-03</internalNodes>
     <leafValues>
```

-8.7463897466659546e-01 1.1760339736938477e+00</leafValues></_>
<_>
<internalNodes>
0 -1 4 1.5014000236988068e-02</internalNodes>
<leafValues>
-7.7945697307586670e-01 1.2608419656753540e+00</leafValues></_>
<_>
<internalNodes>
0 -1 5 9.9371001124382019e-02</internalNodes>
<leafValues>
5.5751299858093262e-01 -1.8743000030517578e+00</leafValues></_>
<_>
<internalNodes>
0 -1 6 2.7340000960975885e-03</internalNodes>
<leafValues>
-1.6911929845809937e+00 4.4009700417518616e-01</leafValues></_>
<_>
<internalNodes>
0 -1 7 -1.8859000876545906e-02</internalNodes>
<leafValues>
-1.4769539833068848e+00 4.4350099563598633e-01</leafValues></_>
.
.
.
.
.
<_>
<rects>
<_>
0 18 18 2 -1.</_>
<_>
0 19 18 1 2.</_></rects></_>
<_>
<rects>
<_>
3 15 19 3 -1.</_>

```
    <_>
      3 16 19 1 3.</_></rects></_>
  <_>
   <rects>
    <_>
      0 13 18 3 -1.</_>
    <_>
      0 14 18 1 3.</_></rects></_>
  <_>
   <rects>
    <_>
      15 17 9 6 -1.</_>
    <_>
      15 19 9 2 3.</_></rects></_>
  <_>
   <rects>
    <_>
      0 17 9 6 -1.</_>
    <_>
      0 19 9 2 3.</_></rects></_>
  <_>
   <rects>
    <_>
      12 17 9 6 -1.</_>
    <_>
      12 19 9 2 3.</_></rects></_>
  <_>
   <rects>
    <_>
      3 17 9 6 -1.</_>
    <_>
      3 19 9 2 3.</_></rects></_>
  <_>
   <rects>
    <_>
      16 2 3 20 -1.</_>
    <_>
      17 2 1 20 3.</_></rects></_>
```

<_>
<rects>
<_>
0 13 24 8 -1.</_>
<_>
0 17 24 4 2.</_></rects></_>
<_>
<rects>
<_>
9 1 6 22 -1.</_>
<_>
12 1 3 11 2.</_>
<_>
9 12 3 11 2.</_></rects></_></features></cascade>
</opencv_storage>

F. Image Result of Haarcascade Face Detection Application using Python

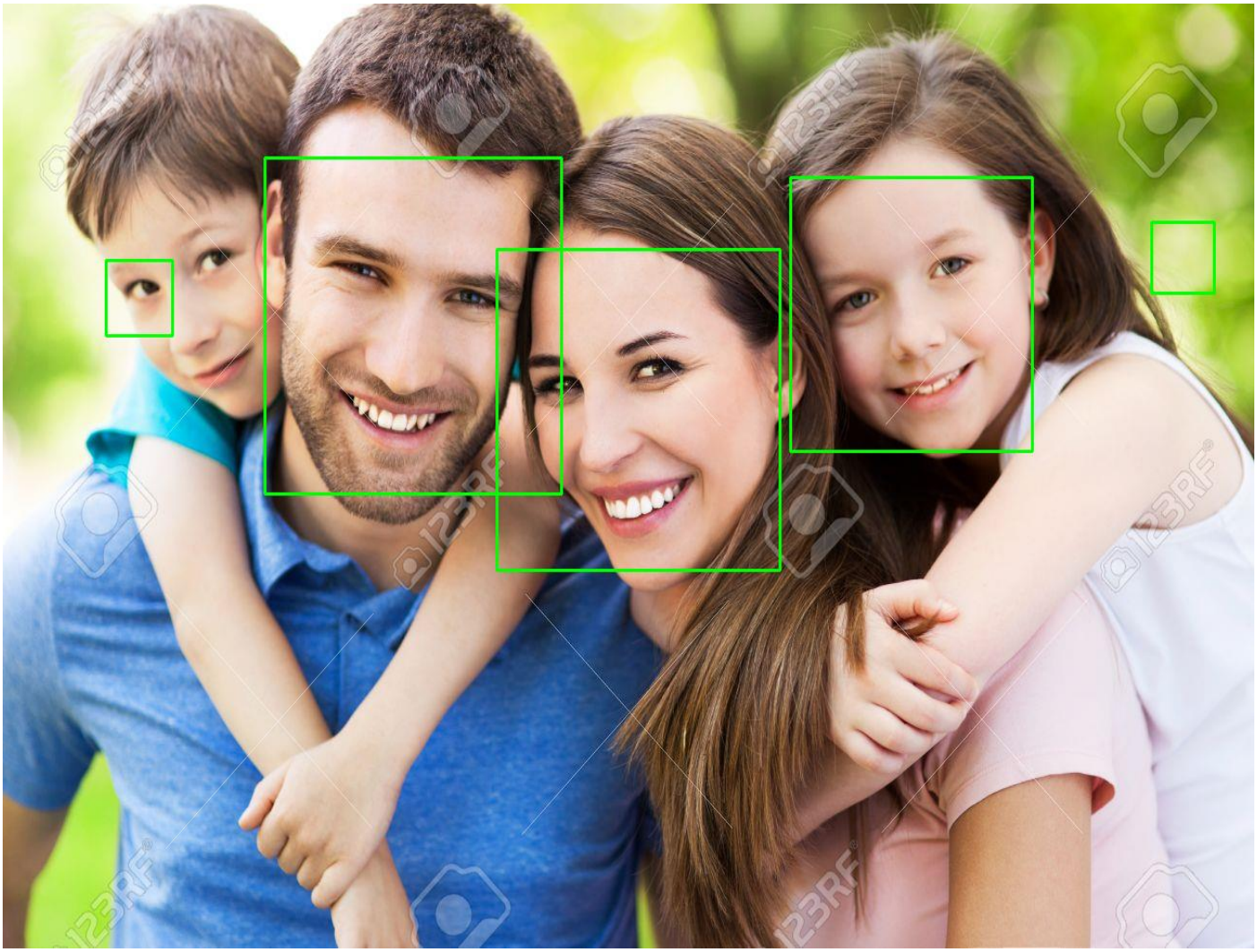


Figure F: Result of Haarcascade Face Detection Application using Python