(12)Indian Patent Application

(21) Application Number: 202141028961

(22) Filing Date: 28/06/2021 (43) Publication Date: 30/12/2022

(71) Applicant(s): L&T TECHNOLOGY SERVICES LIMITED

(72) Inventor(s): Madathil, Nithin
Khanum, Zaibanousheen
Chavan, Anusha
Chand, Sanjat

(51) International Classifications: G06K 9/00 G08B 21/24 G06F 16/335 G06F 16/248 G06F 16/2458

(30) Priority: 17/06/2022 WO PCT/IN2022/050554

(54) Title: SYSTEM AND METHOD FOR AUTOMATED SOFTWARE TESTING

(57) Abstract: The present disclosure describes a method (1000) and a device (110) for evaluating one or more functional attributes of an application. The device (110) comprises a processor (204) configured to capture screenshot(s) of the application and use a trained model (210) to detect object(s) within the application. The processor (204) is further configured to generate a first report comprising a label and a pair of coordinates corresponding to each of the detected object(s) and a second report comprising the label and a locator corresponding to each of the detected object(s). The processor (204) is configured to generate a third report comprising the label, the pair of coordinates, and the locator corresponding to each of the one detected object(s). The processor (204) is further configured to execute pre-generated test script(s) (214) based at least on the content of the third report to evaluate the functional attributes of the application.

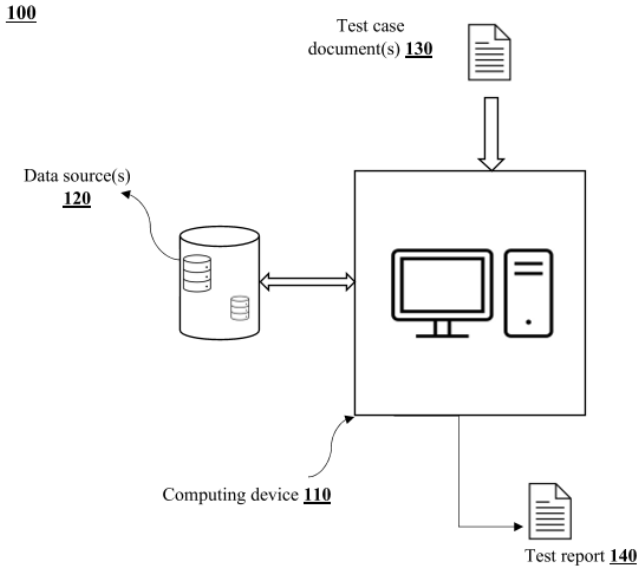

Figure 1

FORM 2

THE PATENTS ACT 1970
(39 OF 1970)
&
The Patent Rules, 2003

Complete Specification

(See Section 10 and Rule 13)

1. TITLE OF THE INVENTION

SYSTEM AND METHOD FOR AUTOMATED SOFTWARE TESTING

2. APPLICANT(S)

(a) NAME : **L&T TECHNOLOGY SERVICES LIMITED**

(b) NATIONALITY : **INDIAN**

(c) ADDRESS : **DLF IT SEZ Park, 2nd Floor – Block 3**

1/124, Mount Poonamallee Road,

Ramapuram, Chennai – 600 089,

INDIA.

3. PREAMBLE TO THE DESCRIPTION

COMPLETE

The following specification describes the invention and the manner in which it is to be performed

TECHNICAL FIELD

[0001] The present disclosure generally relates to the field of software testing. Particularly, the present disclosure relates to a method and system for automated software testing.

BACKGROUND

[0002] In the process of software development, software testing plays an important role for developing an effective and bug-free software product. In general, software testing may be defined as the technique of evaluating a software product or application to check if the software product/application is functioning as expected or not. Software testing is important because it helps in early identification and fixing of bugs, errors etc. before the software product is launched. There are different ways to test a software application, one way is to manually test the software application. In manual testing, an operator/tester (i.e., human being) manually enters data on application screens and checks if the software application is responding and/or functioning as expected or not. However, manual testing typically consumes a lot of time and may not be accurate enough because of the possibilities of human errors (i.e., the testers may make mistakes).

[0003] Nowadays, because of stiff business competition, organizations prefer faster releases and quality software to meet increasing demand of their services and products. To achieve the faster releases and quality requirements along with achieving quicker return on investment, organizations are moving towards automated software testing. The shift towards automated software testing has been majorly driven by Artificial Intelligence (AI) and Machine Learning (ML). In the automated software testing, a software or a test script automatically executes various test cases on the application screens. A test script may be defined as a set of actions that may be performed on an application under test to assess that the application is functioning as expected.

[0004] Nowadays, in order to implement the automated software testing for an application, an automation engineer manually writes test scripts. For that, the automation engineer has to go through the software application, analyze test case document(s) provided by customer(s), identify locators which identify various objects in the application, and then manually write the test scripts. This process of manually writing the test scripts is less efficient as it is time consuming, prone to human errors, and requires skilled/proficient automation engineers for writing the test scripts. Moreover, to test different types of software applications, professionals with different skillsets may be required or a professional needs to be trained for different skillsets to test different software applications, which increases overall cost and time associated with the software testing. Manually

writing the test scripts becomes more difficult when there are a large number of test cases that are to be executed to test the application.

[0005] Software development is a continuous process and with time there are continuous enhancements/modifications in software applications (e.g., user interface changes, functional changes, etc.). Conventionally, whenever there is a change in the software application, the automation engineer has to manually update the test scripts. For example, if there is a change in user interface of an application, the automation engineer has to manually update the locators for identifying objects in the application. Hence, considerable efforts and resources are required to keep the test scripts updated. Thus, it becomes tedious to perform automated software testing for applications that are under constant enhancements.

[0006] Thus, with the huge and rapidly growing demand of automation in the field of software testing, there exists a need for further improvements in the existing techniques of software testing. Particularly, there exists a need for time and resource efficient techniques that can automatically generate and execute test scripts for testing software applications. Further, there exists a need for techniques that can reduce overall cost and time required in training the professionals for testing the software applications. Furthermore, there exists a need for techniques that are capable of reusing previously generated test scripts.

[0007] The information disclosed in this background section is only for enhancement of understanding of the general background of the disclosure and should not be taken as an acknowledgement or any form of suggestion that this information forms the prior art already known to a person skilled in the art.

SUMMARY

[0008] The following presents a simplified summary to provide a basic understanding of some aspects of the disclosed automated software testing. This summary is not an extensive overview and is intended to neither identify key or critical elements nor delineate the scope of such elements. Its purpose is to present some concepts of the described features in a simplified form as a prelude to the more detailed description that is presented later.

[0009] According to an aspect of the present disclosure, methods and systems are provided for automatically testing/evaluating one or more functionalities or functional attributes of an application.

[0010] Various example embodiments described herein relate to a method for evaluating one or more functional attributes of an application. The method comprises capturing one or more screenshots of the application and processing, using a trained model, the one or more captured screenshots to detect at least one object within the application. The method further comprises generating, using the trained model, a first report comprising a label and a pair of coordinates corresponding to each of the at least one detected object. The method further comprises generating a second report comprising the label and a locator corresponding to each of the at least one detected object and generating a third report based on the first report and the second report. The third report comprises the label, the pair of coordinates, and the locator corresponding to each of the at least one detected object. The method further comprises executing one or more pre-generated test scripts based at least on the content of the third report to evaluate the one or more functional attributes of the application.

[0011] Various example embodiments described herein relate to a method for evaluating one or more functional attributes of an application, where the method comprises receiving a plurality of screenshots corresponding to a plurality of applications and augmenting the plurality of screenshots by performing one or more image augmentation operations on the plurality of screenshots and by adjusting one or more parameters of the plurality of screenshots. The method further comprises annotating the plurality of augmented screenshots by assigning labels to different objects present in the plurality of augmented screenshots and training the model by iteratively processing the plurality of annotated screenshots using machine learning techniques.

[0012] Various example embodiments described herein relate to a method for evaluating one or more functional attributes of an application, where the method comprises receiving a test case document. The test case document comprises data related to one or more test cases for evaluating the one or more functional attributes of the application. The method further comprises processing the received test case document to identify one or more keywords relevant to said one or more test cases and generating the one or more test scripts by corelating the identified one or more keywords with a plurality of predefined keywords.

[0013] Various example embodiments described herein relate to a method for evaluating one or more functional attributes of an application, where the locator uniquely locates the detected object within the application, and wherein the locator comprises an xpath of the detected object.

[0014] Various example embodiments described herein relate to a method for evaluating one or more functional attributes of an application, where evaluating the one or more functional attributes of the application comprises comparing result of the execution of the one or more pre-generated test scripts with an expected result and determining that the one or more functional attributes of the application are working as expected based on the comparing.

[0015] Various example embodiments described herein relate to a computing device for evaluating one or more functional attributes of an application. The computing device comprises a memory and at least one processor communicatively coupled with the memory. The at least one processor is configured to capture one or more screenshots of the application and process the one or more captured screenshots using a trained model to detect at least one object within the application. The at least one processor is further configured to generate, using the trained model, a first report comprising a label and a pair of coordinates corresponding to each of the at least one detected object and generate a second report comprising the label and a locator corresponding to each of the at least one detected object. The at least one processor is further configured to generate a third report based on the first report and the second report, where the third report comprises the label, the pair of coordinates, and the locator corresponding to each of the at least one detected object. The at least one processor is further configured to execute one or more pre-generated test scripts based at least on the content of the third report to evaluate the one or more functional attributes of the application.

[0016] Various example embodiments described herein relate to a computing device for evaluating one or more functional attributes of an application, where the at least one processor is further configured to receive a plurality of screenshots corresponding to a plurality of applications; augment the plurality of screenshots by performing one or more image augmentation operations on the plurality of screenshots and by adjusting one or more parameters of the plurality of screenshots; annotate the plurality of augmented screenshots by assigning labels to different objects present in the plurality of augmented screenshots; and train the model by iteratively processing the plurality of annotated screenshots using machine learning techniques.

[0017] Various example embodiments described herein relate to a computing device for evaluating one or more functional attributes of an application, where the at least one processor is further configured to receive a test case document. The test case document comprising data related to one or more test cases for evaluating the one or more functional attributes of the application. The at least one processor is further configured to process the received test case document to identify

one or more keywords relevant to said one or more test cases and generate the one or more test scripts by corelating the identified one or more keywords with a plurality of predefined keywords.

[0018] Various example embodiments described herein relate to a computing device for evaluating one or more functional attributes of an application, where the locator uniquely locates the detected object within the application, and wherein the locator comprises an xpath of the detected object.

[0019] Various example embodiments described herein relate to a computing device for evaluating one or more functional attributes of an application, where to evaluate the one or more functional attributes of the application, the at least one processor is configured to compare result of the execution of the one or more pre-generated test scripts with an expected result; and determine that the one or more functional attributes of the application are working as expected based on the comparing.

[0020] The foregoing summary is illustrative only and is not intended to be in any way limiting. In addition to the illustrative aspects, embodiments, and features described above, further aspects, embodiments, and features will become apparent by reference to the drawings and the following detailed description.

BRIEF DESCRIPTION OF DRAWINGS

[0021] The embodiments of the disclosure itself, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings. One or more embodiments are now described, by way of example only, with reference to the accompanying drawings in which:

[0022] **Figure 1** shows an exemplary communication system **100** for implementing the techniques of automated software testing, in accordance with some embodiments of the present disclosure.

[0023] **Figure 2** shows a detailed block diagram **200** of the communication system **100** illustrated in **Figure 1**, in accordance with some embodiments of the present disclosure.

[0024] **Figures 3(a)** shows a flow diagram **300-1** illustrating a process flow for training a machine learning model, in accordance with some embodiments of the present disclosure.

[0025] Figures 3(b) shows an exemplary cluster **300-2** where different types of login buttons are grouped under same label, in accordance with some embodiments of the present disclosure.

[0026] Figures 4 shows a flow diagram **400** illustrating a process flow for automatically generating one or more test scripts, in accordance with some embodiments of the present disclosure.

[0027] Figures 5(a) shows an exemplary test case document **500-1** for testing login functionality of an exemplary software application, in accordance with some embodiments of the present disclosure.

[0028] Figures 5(b) shows an exemplary screenshot **500-2** of the exemplary software application, in accordance with some embodiments of the present disclosure.

[0029] Figures 6 shows a flow diagram **600** illustrating a process flow for executing one or more test scripts for testing one or more functionalities of a software application, in accordance with some embodiments of the present disclosure.

[0030] Figures 7 shows an exemplary report **700** generated while testing the one or more functionalities of the software application, in accordance with some embodiments of the present disclosure.

[0031] Figures 8 shows another exemplary report **800** generated while testing the one or more functionalities of the software application, in accordance with some embodiments of the present disclosure.

[0032] Figures 9 shows yet another exemplary report **900** generated while testing the one or more functionalities of the software application, in accordance with some embodiments of the present disclosure.

[0033] Figure 10 shows a flowchart illustrating a method **1000** for evaluating one or more functionalities of a software application, in accordance with some embodiments of the present disclosure.

DETAILED DESCRIPTION

[0034] Exemplary embodiments are described with reference to the accompanying drawings. Wherever convenient, the same reference numbers are used throughout the drawings to refer to the

same or like parts. While examples and features of disclosed principles are described herein, modifications, adaptations, and other implementations are possible without departing from the spirit and scope of the disclosed embodiments. It is intended that the following detailed description be considered as exemplary only, with the true scope and spirit being indicated by the following claims. Additional illustrative embodiments are listed below.

[0035] Various embodiments of the present invention now will be described more fully hereinafter with reference to the accompanying drawings, in which some, but not all embodiments of the invention are shown. Indeed, the invention may be embodied in many different forms and should not be construed as limited to the embodiments set forth herein. Rather, these embodiments are provided so that this disclosure will satisfy applicable legal requirements. The term "or" is used herein in both the alternative and conjunctive sense, unless otherwise indicated. The terms "illustrative," "example," and "exemplary" are used to be examples with no indication of quality level. Like numbers refer to like elements throughout.

[0036] The phrases "in an embodiment," "in one embodiment," "according to one embodiment," and the like generally mean that the particular feature, structure, or characteristic following the phrase may be included in at least one embodiment of the present disclosure and may be included in more than one embodiment of the present disclosure (importantly, such phrases do not necessarily refer to the same embodiment).

[0037] The word "exemplary" is used herein to mean "serving as an example, instance, or illustration." Any implementation described herein as "exemplary" is not necessarily to be construed as preferred or advantageous over other implementations.

[0038] If the specification states a component or feature "can," "may," "could," "should," "would," "preferably," "possibly," "typically," "optionally," "for example," "often," or "might" (or other such language) be included or have a characteristic, that particular component or feature is not required to be included or to have the characteristic. Such component or feature may be optionally included in some embodiments, or it may be excluded.

[0039] Although aspects of some embodiments described in the disclosure will focus, for the purpose of illustration, on particular examples of software applications, test case documents, and machine learning models, the examples are illustrative only and are not intended to be limiting.

Various aspects of the disclosure will now be described with regard to certain examples and embodiments, which are intended to illustrate but not limit the disclosure.

[0040] Throughout this specification, the term 'machine learning model' or 'deep learning model' may be used interchangeably, and these terms shall not be taken in a sense to limit the scope of the present disclosure. Further, the term 'machine learning model' or 'deep learning model' may refer to a model which is trained or is subject to training by a processor. Further the term 'trained machine learning model' may refer to a model which is trained and ready to be used in the automated software testing.

[0041] Throughout this specification, examples of the automated software testing are provided in the context of web-applications. However, such examples are used for explaining the one or more embodiments for the purpose of understanding the present disclosure and not for purposes of limitation. A person skilled in the art will understand that the embodiments described may be well suited for any testing any type of software application including mobile applications. Therefore, the examples mentioned throughout the specification shall not be taken in a sense to limit the scope of the present disclosure.

[0042] The terms like "at least one" and "one or more" may be used interchangeably throughout the description. The terms like "a plurality of" and "multiple" may be used interchangeably throughout the description. Further, the terms like "object", "element", and "web element" may be used interchangeably throughout the description. Further, the terms like "functionalities" and "functional attributes" may be used interchangeably throughout the description.

[0043] In the following detailed description of the embodiments of the disclosure, reference is made to the accompanying drawings that form a part hereof, and in which are shown by way of illustration of specific embodiments in which the disclosure may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the disclosure, and it is to be understood that other embodiments may be utilized and that changes may be made without departing from the scope of the present disclosure. The following description is, therefore, not to be taken in a limiting sense. In the following description, well known functions or constructions are not described in detail since they would obscure the description with unnecessary detail.

[0044] In the context of the present disclosure, a webpage may be defined as a hypertext document delivered by a web server and displayed to a user using a web browser. A web page may consist of

various objects (also called elements or web elements). Examples of objects may include any visually displayed function on the webpage including, but not limited to, Edit box, Link, Button, Image, image link, image button, Text area, Checkbox, Radio button, Dropdown list, etc. To interact with the objects or to perform any operation (action) like writing in a text box, selecting a checkbox, selecting an option from a drop-down etc., the objects may need to be located first. For the purpose of uniquely locating/identifying the objects within a webpage, locators may be used.

[0045] In the context of present disclosure, a locator may be defined as an address that uniquely identifies/locates an object within a web application or a webpage. In short, a locator finds an object within the webpage. Examples of locators may include: id, name, className, tagName, linkText, partialLinkText, cssSelector, xpath etc. Among various locators, xpath is most commonly used locator which may help in navigating through a webpage. All objects within the webpage may be located/identified uniquely using their xpaths. In general, xpath is a path written using html tags (or XML tags) and their attributes to navigate to an object on the webpage. Typically, there are two types of xpaths: Absolute xpath and Relative xpath.

[0046] Absolute xpath is the simplest form of xpath which provides an absolute path of an object within the webpage. The absolute xpath starts from the very beginning of the webpage (i.e., from root object) and goes down to the desired object one by one. Since in an html page the first tag is HTML, the absolute xpath starts from the html tag. In absolute xpath, "/"is used to access an immediate child of a parent tag. On the other hand, relative xpath generally starts from anywhere on the webpage and may search objects anywhere on the webpage. In relative xpath, "//" is used to access any child of the parent tag. Generally, relative xpaths are preferred for testing software applications because absolute xpaths needs to be updated whenever there is any change in the html structure of the webpage.

[0047] In the context of present disclosure, a label may be defined as a name assigned to an object. Consider an example of a webpage, where a user clicks on a "*click button*" and is redirected to a login screen thereafter. In this example, the element "*click button*" is an object and the name assigned to the button (e.g., "*Sign In*" or "*click here to Sign In*") is a label. Hence, the label may be defined as a name describing the object. Furthermore, the address that uniquely identifies/locates the object "*click button*" within the webpage may be named as the locator of the object in the present disclosure. The label may also be referred as 'class name' in the present disclosure.

[0048] The present disclosure provides techniques (methods and systems) for automated software testing using automatically generated test scripts. As described in the background section,

conventionally an automation engineer has to manually write the test scripts for an application. Also, if there is any modification/enhancement in the application, the automation engineer has to manually update the test scripts, which requires considerable efforts and resources. Thus, it becomes tedious to perform automated software testing for applications that are under constant enhancements.

[0049] To overcome these and other problems, the present disclosure proposes using Artificial Intelligence (AI) and/or Machine Learning (ML) techniques for automatically generating and executing one or more test scripts at run time for testing one or more functional attributes of a software application.

[0050] The present disclosure proposes to automate the process of software testing with the help of AI and ML. Automating the process of software testing with the help of AI/ML has many potential benefits such as, but limited to, improved accuracy, reduced cost, minimal resource requirement, reduced testing time. AI in the software testing helps an organization in reducing the time needed for software testing, so the organization can focus more on complex tasks.

[0051] Referring now to **Figure 1**, which illustrates a communication system **100** for implementing the embodiments consistent with the present disclosure. It must be understood to a person skilled in art that techniques of the present disclosure may also be implemented in various other systems, other than as shown in Figure 1. In accordance with some embodiments of the present disclosure, the communication system **100** may be used for evaluating/testing one or more functionalities or functional attributes of a software application. The communication system **100** may comprise a computing device **110** which may be in communication with one or more data sources **120**. Additionally, the computing device **110** may be in communication with other computing devices (not shown) via at least one network (not shown).

[0052] The network may comprise a data network such as, but not restricted to, the Internet, Local Area Network (LAN), Wide Area Network (WAN), Metropolitan Area Network (MAN), etc. In certain embodiments, the network may include a wireless network, such as, but not restricted to, a cellular network and may employ various technologies including Enhanced Data rates for Global Evolution (EDGE), General Packet Radio Service (GPRS), Global System for Mobile Communications (GSM), Internet protocol Multimedia Subsystem (IMS), Universal Mobile Telecommunications System (UMTS) etc. In one embodiment, the network may include or otherwise cover networks or subnetworks, each of which may include, for example, a wired or wireless data pathway.

[0053] The one or more data sources **120** may comprise a plurality of images/screenshots of one or more software applications or webpages. In one embodiment, the plurality of screenshots may be captured using inbuilt Application Programming Interfaces (APIs) or other image capturing mechanisms provided inside the computing device **110**. The captured screenshots may be stored in the one or more data sources **120**. In another embodiment, plurality of screenshots may be captured using an external device (not shown) and then stored in the one or more data sources **120**. The plurality of screenshots may be used for the purpose of training a machine learning (ML) or deep learning (DL) model. For the sake of explanation, it may be considered that the one or more applications are web applications. However, the present disclosure is not limited thereto, and in general the one or more applications may be any software applications.

[0054] In one non-limiting embodiment of the present disclosure, the computing device **110** may receive one or more test case documents **130**. The test case documents **130** may be received from an external device or the computing device **110** may fetch the one or more case documents **130** from the external device. In another embodiment, the one or more test case documents **130** may be provided by a customer and then uploaded on the computing device **110** by an operator of the computing device **110**. The computing device **110** may perform software testing and thereafter may generate an output file or test report or log file **140**. The generated test report **140** may be provided as an output to the operator.

[0055] In an exemplary embodiment, the computing device **110** may be communicatively coupled with the one or more data sources **120** comprising the plurality of screenshots. The computing device **110** may receive/fetch the plurality of screenshots from the one or more data sources **120** to train a machine learning model. During real-tome implementation, the computing device **110** may receive at least one test case document **130** from an external entity and process the received at least one test case document **130** based at least on the trained ML model to generate the test report **140**.

[0056] Now, **Figure 1** is explained in conjunction with **Figure 2**, which is a detailed block diagram **200** of the communication system **100** for evaluating/testing one or more functional attributes of an application, in accordance with some embodiments of the present disclosure. According to an embodiment of the present disclosure, the computing device **110** may comprise one or more interfaces **202**, at least one processor **204**, a memory **206**, and various units or modules **218**. The units **218** may comprise a capturing unit **220**, a processing unit **222**, a generating unit **224**, a combining unit **226**, an executing unit **228**, a receiving unit **230**, a transmitting unit **232**, and various other units **234**. In one non-limiting embodiment, the units **220**-**234** may be dedicated hardware

units capable of executing one or more instructions stored in the memory **206** for performing various operations of the computing device **110**. In another non-limiting embodiment, the units **220**-**234** may be software modules stored in the memory **206** which may be executed by the at least one processor **204** for performing the operations of the computing device **110**.

[0057] The interfaces **202** may include a variety of software and hardware interfaces, for example, a web interface, a graphical user interface, an input device-output device (I/O) interface, a network interface, and the like. The I/O interfaces may allow the computing device **110** to interact with other computing devices directly or through other devices. The network interface may allow the computing device **110** to interact with one or more data sources **120** either directly or via a network.

[0058] The memory **206** may comprise various types of data. For example, the memory may comprise one or more keywords **208**, trained model(s) **210**, one or more reports **212**, one or more test scripts **214**, and other various types of data **216** (such as one or more screenshots, one or more test case documents **130**, output test reports **140** etc.). The memory **206** may comprise the one or more data sources **120**. The memory **206** may further store one or more instructions executable by the at least one processor **204**. The memory **206** may be communicatively coupled to the at least one processor **204**. By way of example but not limitation, the memory **206** may include random access memory (RAM), read-only memory (ROM), electrically erasable programmable read-only memory (EEPROM), FLASH memory, disk storage, magnetic storage devices, or the like. Disk storage, as used herein, includes compact disc (CD), laser disc, optical disc, digital versatile disc (DVD), floppy disk, and Blu-ray Disc™, or other storage devices that store data magnetically or optically with lasers. Combinations of the above types of media are also included within the scope of the terms non-transitory computer-readable and processor-readable media.

[0059] The at least one processor **204** may include, but not restricted to, a general-purpose processor, a Field Programmable Gate Array (FPGA), an Application Specific Integrated Circuit (ASIC), a Digital Signal Processor (DSP), microprocessors, microcomputers, micro-controllers, central processing units, state machines, logic circuitries, and/or any devices that manipulate signals based on operational instructions. A processor may also be implemented as a combination of computing devices, e.g., a combination of a DSP and a microprocessor, a plurality of microprocessors, one or more microprocessors in conjunction with a DSP core, or any other such configuration. Among other capabilities, the processor **204** may be configured to fetch and execute computer-readable instructions stored in the memory.

[0060] In the forthcoming paragraphs, the implementation details of the present disclosure are explained by means of **Figures 3-9** in conjunction with **Figures 1 and 2** (discussed above). Referring now to **Figure 3(a)**, which shows a flow diagram **300-1** illustrating a process flow for training a machine learning model, in accordance with some embodiments of the present disclosure. In one non-limiting embodiment of the present disclosure, the processing unit **222** may be configured to train the machine learning model for detecting one or more objects in a software application. The forthcoming paragraphs now describe the detailed process of training the machine learning model.

[0061] Referring to **Figure 3(a)**, the processing unit **222** in conjunction with the receiving unit **230** may be configured to receive or fetch a plurality of screenshots of one or more applications from the at least one data source **120**. In the context of present disclosure, a screenshot may refer to any picture of a computer, smartphone, or tablet screen (on which a software application is open) captured and saved as an image file. The plurality of screenshots may be captured by the computing device **110** or by an external device (not shown) and then stored in the one or more data source **120**. The plurality of screenshots may be automatically captured during an execution of the one or more applications. In one non-limiting embodiment, one or more inbuilt tools/libraries of a framework may be used for capturing the plurality of screenshots. In another non-limiting embodiment, third party APIs or applications may be used for capturing the plurality of screenshots.

[0062] Typically, a machine learning model works well when it has a huge amount of training data i.e., more the amount of training data, better will be the performance of the machine learning model. Sometimes it may not be possible to have a large amount of data to train a machine learning network, which may result in poor performance of the machine learning model. To overcome such issues, the present disclosure utilizes image data augmentation for expanding the size of training dataset in order to improve performance and ability of the machine learning model (block **310**).

[0063] In one non-limiting embodiment of the present disclosure, for augmenting the plurality of screenshots, different image augmentation techniques may be used such as image rotation, image flipping, image noising, image shifting, image padding, image cropping, image blurring, affine transformation, but not limited thereto. Image rotation is the technique of creating multiple images by rotating an image at different angles. Image shifting may comprise changing position of different objects present in the image. Image flipping may comprise flipping an image in the left-right direction as well as in the up-down direction. Image noising may comprise adding noise to an image. Image cropping may refer to cropping a part of an image to create a random subset of the

image Affine transformation may refer to changing geometric structure of an image while preserving lines and parallelism (but not necessarily distances and angles). Additionally, one or more parameters/features of an image such as brightness, contrast, saturation etc. may be adjusted to augment the screenshots. The present disclosure is not limited to these image augmentation techniques and in general any image augmentation technique may be used for augmenting the plurality of screenshots.

[0064] At block **310**, the processing unit **222** may be configured to augment the plurality of received screenshots using one or more of the above-described image augmentation techniques/operations and/or by adjusting one or more parameters (e.g., brightness, contrast, saturation etc.) of the plurality of screenshots. Once the plurality of screenshots are augmented, the processing unit **222** may be configured to annotate the plurality of augmented screenshots (block **320**). Image annotation may be defined as assigning class names or labels to different objects present in the plurality of augmented screenshots. Usually, labels are predetermined by an automation engineer and are chosen to give the training model information about what is shown in a screenshot. The various objects present in the plurality of augmented screenshots may be assigned same or different labels depending on their functionality. Particularly, the objects which have same/similar functionality may be assigned a same label and may be grouped into one group irrespective of their appearance/shape. For example, different types of login buttons present in different screenshots may be grouped together and assigned a same label called "SIGNIN", as shown in **Figure 3(b)**. **Figure 3(b)** shows an exemplary cluster **300-2** where different types of login buttons having different shapes/appearances are grouped under the same label.

[0065] In one non-liming embodiment, the automation engineer may manually assign labels to the different objects present in the plurality of augmented screenshots. In another non-limiting embodiment, the processing unit **222** may automatically assign the labels to different objects using any conventionally known annotation techniques such as, but not limited to, 2D/3D bounding box annotation, polygon annotation, semantic segmentation, cuboidal annotation.

[0066] In one non-limiting embodiment of the present disclosure, at block **330**, the processing unit **222** may be configured to train the machine learning model by iteratively processing the plurality of annotated screenshots using machine learning techniques to generate a trained model **210**. For example, the processing unit **222** may use AI and ML techniques for training the machine learning model by iteratively processing the plurality of annotated screenshots. The processing unit **222** may use YOLO V3 model for generating the trained model. YOLO V3 model employs convolutional neural networks (CNN) to detect and recognize various objects in a screenshot in real-time. For

training the model, the processing unit **222** may divide the plurality of annotated screenshots into training screenshots and testing screenshots. In an exemplary aspect, the training screenshots may comprise 70-90% of the annotated screenshots and the testing screenshots may comprise 10-30% of the annotated screenshots such that no screenshot is a part of the both the training and the testing screenshots. For training the machine learning model, the processing unit **222** may use one or more parameters of the training screenshots such as height, width, pixel density, color etc. A number of training iteration to train the machine learning model may be based on a learning score indicative of valid detection of objects in webpages. The learning score is indicative of a learning loss encountered when training the model. If it is identified during training that the model is subjected to less learning loss (i.e., the learning score is high) then such a model may be deployed in the software testing. In some examples, the model may be trained iteratively until the learning loss is less than a threshold value indicative a high learning score. Thus, the machine learning model may be trained iteratively until the learning loss is less than the benchmark value. The trained model **210** may be stored in the memory **206**. In one embodiment, the trained model **210** may be in the form of an *.h5* file format which contains multi-dimensional arrays of data. In general, *.h5* file format is designed to store and organize large amounts of data in Hierarchical Data Format (HDF).

[0067] Once the trained model **210** is generated, the proposed techniques may be used in real time for evaluating/testing/assessing one or more functionalities or functional attributes of the software application by automatically generating and executing one or more test scripts to determine whether the one or more functionalities of the application are working as intended or not. The one or more functionalities may comprise any existing or newly added feature of the application.

[0068] Referring now to **Figure 4**,which illustrates a flow diagram **400** illustrating a process flow for automatically generating one or more test scripts **214** in real time from a test case document **130**, in accordance with some embodiments of the present disclosure. In one embodiment, the test case document may be in the form of an xlsx file. The test case document **130** may be defined as a document comprising an organized list of one or more test cases for different test scenarios that verify whether the one or more functionalities of the application are working as intended or not. In an exemplary aspect, the test case document **130** may comprise title of the test case, test steps (i.e., steps to execute the one or more test cases), actions associated with each test step, input or variable data or test data associated with the test steps, and expected result or output, as shown in Figure 5(a).

[0069] Referring now to **Figure 5(a)**,which shows an exemplary test case document **500-1** for testing login functionality of an exemplary software application whose one exemplary screenshot

500-2 is shown in **Figure 5(b)**, in accordance with some embodiments of the present disclosure. As shown in Figure 5(a), the exemplary test case document **500-1** may comprise one test case with title "As a user, I want to verify if I can login the application". The title of the test case indicates the intent of the user/customer for performing the software testing. For instance, in Figure 5(a), the intent of the user is to check if the login functionality of the software application is working as intended or not. Each test case may comprise one or more test steps associated with it and each test step may have an associated action. For instance, the exemplary test case document **500-1** of Figure 5(a), comprises two test steps where the first test step is for launching the software application and second test step is for enter email and password for testing the login functionality. The test steps may require variable data or input for testing the software application. For instance, in the exemplary embodiment of Figure 5(a), the first test step associated with launching the application may require a URL for opening the application. Similarly, the second test step may require email and password to check if the login functionality is working as expected or not. Each test step may have an expected result/output associated with it. The forthcoming paragraphs now describe the process for automatically generating one or more test scripts **214** from the input test case document **500-1**.

[0070] Initially, the processing unit **222** in conjunction with the receiving unit **230** may be configured to receive/fetch the test case document **500-1** (or test case template). The processing unit **222** may be configured to fetch one or more library or user-defined keywords **208** from the memory **206**. At block **410**, the generating unit **224** may be configured to generate the one or more test scripts **214** based on the received test case document **500-1** and the one or more keywords **208**.

[0071] In one non-limiting embodiment of the present disclosure, a test script may be defined as a set of instructions to test any functionality of the software application. The test script is generally written in a programming language. The purpose of the test script is to provide the implementation of test cases in an efficient and effective manner. In one non-limiting embodiment of the present disclosure, each test script may be made up of one or more keywords. Just as a test case is conceptually made up of at least one step, a test script is made up of at least one keyword. Keywords serve as a key to something which is not in view.

[0072] For example, consider a test case with 10 steps, if the functionality associated with those 10 steps is written somewhere and referred by a variable 'CLICK' then 'CLICK' is a keyword. In the present disclosure, the keywords are the foundation upon which all the test scripts are built. The keywords may be reused across the test scripts i.e., one or more test scripts may use same keyword

to test same/similar functionality in one or more software applications. The keyword based testing provides various advantages such as, but not limited to, less maintenance cost, reusability of scripts, achieve more testing with lesser effort, reduced testing time etc.

[0073] The keywords may be divided into two categories: library keywords or low-level keywords; and user-defined keywords or high-level keywords. The library keywords are made available by the libraries used with a framework (built-in and external libraries). For example, "Open Browser" is an example of a library keyword. On the other hand, user-defined keywords are defined and created by the user. In one embodiment, the user-defined keywords may also comprise library keywords in addition to use-defined actions.

[0074] Coming back to **Figure 4**, in one non-limiting embodiment of the present disclosure, the processing unit **222** may process the received test case document **500-1** to read and extract data present inside the test case document **500-1**. For example, the processing unit **222** may extract the one or more test cases, steps to execute the one or more test cases, actions associated with the steps, and variable data associated with the test cases from the test case document **500-1**. Additionally, the processing unit **222** may process the test case document **500-1** to identify one or more keywords or actions words based on the actions associated with the steps. The identified keywords may be corelated/mapped with the library or user-defined keywords to find matches using semantic analysis. The matched keywords may then be used by the generating unit **224** for generating the one or more test scripts **214**. In order the understand the same, reference may be made to below example.

Example 1

[0075] Consider an example of testing 'login feature' of a software application whose exemplary screenshot **500-2** is shown in **Figure 5(b)**. The customer may provide the test case document **500-1** for testing the login functionality of the application. The processing unit **222** may extract one or more test cases, steps to execute the one or more test cases, actions associated with the steps, and variable data associated with the test cases from the test case document **500-1**. Consider that there is a test step in the test case document **500-1** with action as "*Enter email and password and hit the login button*" and variable data associated for this test step may be an email id 'testsite@gmail.com' and a password 'test@123'. The processing unit **222**, upon parsing through the test case document **500-1** using semantic analysis, may determine that this test step describes about entering password. So, it may associate library keyword "Enter Password" with this test step. Similarly, the processing

unit **222** may associate each test step with corresponding library keyword(s). In one non-limiting embodiment, one or more test steps may be associated with a single library keyword.

[0076] After processing the entire test case document **500-1** as shown in Figure 5(a), the processing unit **222** may determines following keywords and actions for testing the 'login feature' of the software application.

a. *Open* Brower
b. *Enter* Email
c. *Enter* Password
d. *Click* Button
e. *Close* Browser

[0077] Each of the above keywords has its associated functionality defined in a separate folder/file stored in the memory **206**. The determined keywords and their associated functionalities may be arranged in a particular order to form a test script. In this manner, the test scripts **214** are generated based on test case documents for testing one or more functionalities of the software applications. It may be noted that the test case documents provided by customers may have different formats. In one non-limiting embodiment, before processing the test case documents, the processing unit **222** may convert the test case documents into a predefined format.

[0078] Referring now to **Figure 6**, which illustrates a flow diagram **600** illustrating a process flow for executing the generated one or more test scripts **214** in real time for testing one or more functionalities of an application, in accordance with some embodiments of the present disclosure.

[0079] In one non-limiting embodiment of the present disclosure, the executing unit **228** may be configured to execute the one or more auto-generated test scripts **214**. For example, in the above described Example 1, when the test script is executed, initially the functionality corresponding to the keyword "Open Browser" is run, a web browser is opened, and a website is opened based on the URL provided in the variable data. Once the application is launched, the capturing unit **220** may be configured to capture one or more screenshots of the application. The one or more screenshots may then be used for the purpose of detecting one or more objects within the application.

[0080] Coming to the specifics, at block **610**, the processing unit **222** may be configured to process the captured screenshots using the trained model **210** to detect at least one object in the captured screenshots. For each of the plurality of screenshots, the processing unit **222** may divide an input

screenshot into a grid of cells. The size of the cells may vary depending on the size of the input screenshot. For example, if input screenshot size is 416×416 and the grid is of 13x13, then the cell size is 32×32.

[0081] Each cell may be responsible for predicting a number of boxes in the screenshot. For each bounding box, the trained model **210** or neural network(s) may predict the confidence that the bounding box actually encloses an object, and the probability of the enclosed object being associated with a particular label. Most of the bounding boxes may get eliminated because their confidence is low or because they are enclosing the same object as another bounding box with very high confidence score. Remaining bounding boxes may be identified as the enclosing objects. Consider that in the above described Example 1, the processing unit **222** may detect at least two text boxes **T1**, **T2** (one for username and one for password) and one click button **B1** as objects within the application along with their coordinates.

[0082] Based on the object detection, the generating unit **224** may generate a first report (or xlsx file) comprising at least one label and at least a pair of coordinates corresponding to the at least one detected object, as shown in **Figure 7**. In one non-limiting embodiment of the present disclosure, the position of the object on a screen or dimensions of the object may be defined in terms of coordinates (or a coordinate pair). Any object has its own screen position on a webpage known as x-y coordinates. In one embodiment, the x-y coordinates of the object may be measured in terms of x and y pixels, where x pixel means the horizontal position of the object on the webpage from the left side and y pixel means the vertical position of the object on the webpage from the top. In one non-limiting embodiment of the present disclosure, the screen position, dimensions, size of the object may be measured using one or more predefined libraries of a framework.

[0083] Referring to **Figure 7**, an exemplary report **700** is shown comprising labels and coordinates corresponding to the detected objects, in accordance with some embodiments of the present disclosure. Consider that the capturing unit **220** captures one screenshot (as shown in Figure 5(b) of a software application whose login functionality is to be tested. The processing unit **222** may process the captured screenshot using the trained model **210** to detect objects present the captured screenshots. In the above Example 1, the processing unit **222** may detect three objects (i.e.. two textboxes **T1**, **T2** and one click button **B1**). The processing unit **222** may also determine the coordinates of the detected objects along with confidence scores. The generating unit **224** may then generate the first report **700** comprising the labels of the detected objects, their coordinates, and confidence scores, as shown in **Figure 7**. It may be noted that the report **700** is only for exemplary purpose and in general the report **700** may comprise various other entries as well.

[0084] Sometimes, the screenshots of the software applications may comprise objects in the forms simple texts instead of check boxes, radio buttons, click buttons etc. To detect such objects, the present disclosure utilizes an Optical Character Recognition (OCR) engine which detects the textual objects in the screenshots (block **620**). Similar to generating the first report, the generating unit **224** may generate another report (or xlsx file) based on output from the OCR engine. For instance, the generating unit **224** may generate a report **800** comprising the labels/name of exemplary textual object (e.g., 'Click Here', 'Privacy Policy', 'Home', etc.) and their coordinates, as shown in **Figure 8**.

[0085] Coming back to **Figure 6**, at block **630**, the generating unit **224** may be configured to determine/extract locators of the different objects of the software application based on a source file or HTML source code of the application using prebuilt libraries (e.g., BeautifulSoup 4). The generating unit **224** may be configured to generate a second report (or xlsx file) comprising at least one label and at least one locator corresponding to the at least one detected object. For example, in above described Example 1, the generating unit **224** may generate an exemplary report **900** comprising various objects and their locators (i.e., xpaths), as shown in **Figure 9**.

[0086] At block **640** of Figure 6, the generating unit **224** in conjunction with the combining unit **226** may be configured to generate a third report (or xlsx file) based on the first report **700**, second report **900**, and the report **800**. The third report may be referred to as a feature table. The feature table may comprise a label, a pair of coordinates, and a locator for each of the at least one detected object. In one non-limiting embodiment, the third report may be a logical report. The reports **700, 800, 900** may be same as the reports **212** of Figure 2.

[0087] At block **650**, the executing unit **228** may be configured to execute the generated one or more test scripts **214** for the content of the feature table to evaluate the one or more functionalities of the software application. In the above-described Example 1 of testing login functionality, after detecting the objects (text boxes and click button), the processing unit **222** may enter credentials (username and password) from the variable data and click on the login button. Once the user is logged in, the processing unit **222** may verify whether the login functionality is working as intended or not. Particularly, the processing unit 222 may compare the output or a result of execution of the generated test scripts **214** with the expected result and may determine that the login functionality is working as intended when the result of execution of the test scripts **214** matches with the expected result. When the result of execution of the test scripts **214** does not match with the expected result,

the processing unit **222** may determine that the login functionality is not working as intended and may indicate the same to the operator. The processing unit **222** may then close the browser.

[0088] In one non-limiting embodiment of the present disclosure, the generating unit **224** may generate a test report or a log file **140** based on the result of execution of the test scripts **214** over the one or more functionalities of the application. The test report **140** may comprise detailed information related to execution of the one or more generated test scripts **214** over the one or more functionalities of the application. The test report **140** may provide detailed information regarding the success of each test performed to validate the quality, performance, and functionality of the application. The test report **140** may provide a detailed summary of overall test run and indicates passed and failed tests. This information may be very helpful for the automation engineer in drilling down on data to gain insight and perform root cause analysis in case of any failures during execution of the test scripts **214**. In one non-limiting embodiment, the data present inside the test report **140** may be displayed on a display or may be automatically sent to intended users via email.

[0089] Thus, the present disclosure provides techniques for automatically generating and executing test scripts in real time for testing software applications, thereby eliminating the need of manual software testing. Specifically, the present disclosure provides techniques for automatically testing the software application that undergo continuous modifications /enhancements. For example, when there is change in user interface (UI) of an application, the automation engineer will not have to update the locators to identify objects in application as they are identified dynamically at run time (i.e., during execution of the test scripts). Since the objects in the application are identified using object detection, change in object locators will not affect the execution of the test scripts.

[0090] Further, every time a test case is queued for test run, first the test scripts are generated automatically and are executed on the go. Thus, there is no need to keep/store any test scripts at any point of time. Also, since the test scripts are automatically generated, an automation engineer will not have to write all the test scripts manually. Since most of the functions are automated, the cost involved in maintenance is minimal. Hence, the proposed techniques are efficient in terms of time, cost, and resources.

[0091] The proposed techniques may be used for testing different applications in a domain. Although the disclosure has been described considering the software application as a web application. However, the present disclosure is not limited thereto, and the proposed techniques may be used for different applications across different domains with some modifications.

[0092] In one non-limiting embodiment of the present disclosure, the at least one processor **204** in conjunction with the memory **206** may be configured to perform some or all operations of the computing device **110**. Particularly, the at least one processor **204** may be configured to performs the various operations of the units **220-234**, as discussed above.

[0093] Referring now to **Figure 10**, a flowchart is described illustrating an exemplary method **1000** for testing/evaluating one or more functionalities of an application, according to an embodiment of the present disclosure. The method **1000** is merely provided for exemplary purposes, and embodiments are intended to include or otherwise cover any methods or procedures for evaluating one or more functionalities of an application.

[0094] The method **1000** may include, at block **1002**, capturing one or more screenshots of the application. For example, the at least one processor **204** may be configured to capture the one or more screenshots of the application. The operations of block **402** may also be performed by the capturing unit **220** of **Figure 2**.

[0095] In one non-limiting embodiment of the present disclosure, though not exclusively disclosed, the method may include the step of training an object detection model (or machine learning model) over one or more software applications. The training may comprise receiving a plurality of screenshots corresponding to a plurality of applications and augmenting the plurality of screenshots by performing one or more image augmentation operations (such as image rotation, image flipping, image noising, image shifting, image padding, image cropping, image blurring, affine transformation, but not limited thereto) on the plurality of screenshots and/or by adjusting one or more parameters (such as brightness, contrast, saturation, , but not limited thereto) of the plurality of screenshots. The training may further comprise annotating the plurality of augmented screenshots by assigning labels to different objects present in the plurality of augmented screenshots and training the object detection model by iteratively processing the plurality of annotated screenshots using machine learning techniques. The operations of training the model may be performed by the at least one processor **204** of **Figure 2** or by the processing unit **222** of **Figure 2**.

[0096] At block **1004**, the method **1000** may include processing the one or more captured screenshots using the trained model (210) to detect at least one object within the application. For example, the at least one processor **204** may be configured to processing the one or more captured

screenshots using the trained model (210) to detect the at least one object within the application. The operations of block **1004** may also be performed by the processing unit **222** of **Figure 2**.

[0097] At block **1006**, the method **1000** may include generating, using the trained model (210), a first report comprising a label and a pair of coordinates corresponding to each of the at least one detected object. For example, the at least one processor **204** may be configured generate, using the trained model (210), the first report comprising the label and the pair of coordinates corresponding to each of the at least one detected object. The operations of block **1006** may also be performed by the generating unit **224** of **Figure 2**.

[0098] At block **1008**, the method **1000** may include generating a second report comprising the label and a locator corresponding to each of the at least one detected object. For example, the at least one processor **204** may be configured to generate the second report comprising the label and the locator corresponding to each of the at least one detected object. The operations of block **1008** may also be performed by the generating unit **224** of **Figure 2**.

[0099] In one non-limiting embodiment of the present disclosure, the locator may uniquely locate the detected objects within the application and the locator may comprise an xpath of the detected object.

[0100] At block **1010**, the method **1000** may include generating a third report based on the first report and the second report. For example, the at least one processor **204** may be configured to generate the third report based on the first report and the second report. The operations of block **1010** may also be performed by the generating unit **224** of **Figure 2**. The third report may comprise the label, the pair of coordinates, and the locator corresponding to each of the at least one detected object.

[0101] In one non-limiting embodiment of the present disclosure, the method **1000** may further comprise generating one or more test scripts **214** by processing a test case document **130** in real-time. For example, the method **1000** may comprise receiving the test case document **130** comprising data related to one or more test cases for evaluating the one or more functional attributes of the application. The method may further comprise processing the received test case document **130** to identify one or more keywords relevant to said one or more test cases and generating the one or more test scripts **214** by corelating the identified one or more keywords with a plurality of predefined keywords. The steps of generating the one or more test scripts **214** may be performed

by the at least one processor **204**; or by the processing unit **222** in conjunction with the generating unit **224** and receiving unit **230** of **Figure 2**.

[0102] At block **1012**, the method **1000** may include executing the one or more pre-generated test scripts 214 based at least on the content of the third report to evaluate the one or more functional attributes of the application. For example, the at least one processor **204** may be configured to execute the one or more pre-generated test scripts 214 based at least on the content of the third report to evaluate the one or more functional attributes of the application. The operations of block **1012** may also be performed by the executing unit **228** of **Figure 2**.

[0103] In one non-limiting embodiment of the present disclosure, the operation of evaluating the one or more functional attributes of the application may comprise comparing result of the execution of the one or more pre-generated test scripts **214** with an expected result and determining that the one or more functional attributes of the application are working as expected based on the comparing (i.e., when the result of the execution of the one or more pre-generated test scripts **214** matches with the expected result). The operations of evaluating may be performed by the at least one processor **204**.

[0104] In one non-limiting embodiment, the method may further comprise generating a test report based on the result of execution of the one or more pre-generated test scripts **214** over the one or more functional attributes of the application. The test report may comprise detailed information related to execution of the one or more pre-generated test scripts **214** over the one or more functional attributes of the application.

[0105] The above method **1000** may be described in the general context of computer executable instructions. Generally, computer executable instructions can include routines, programs, objects, components, data structures, procedures, modules, and functions, which perform specific functions or implement specific abstract data types.

[0106] The order in which the various operations of the method are described is not intended to be construed as a limitation, and any number of the described method blocks can be combined in any order to implement the method. Additionally, individual blocks may be deleted from the methods without departing from the spirit and scope of the subject matter described herein. Furthermore, the methods can be implemented in any suitable hardware, software, firmware, or combination thereof.

[0107] The various operations of method described above may be performed by any suitable means capable of performing the corresponding functions. The means may include various hardware and/or software component(s) and/or module(s), including, but not limited to the processor **204** and the various units of **Figure 2**. Generally, where there are operations illustrated in Figures, those operations may have corresponding counterpart means-plus-function components.

[0108] It may be noted here that the subject matter of some or all embodiments described with reference to **Figures 1-9** may be relevant for the method **1000** and the same is not repeated for the sake of brevity.

[0109] In one non-limiting embodiment of the present disclosure, one or more computer-readable storage media may be utilized in implementing embodiments consistent with the present disclosure. A computer-readable storage medium refers to any type of physical memory on which information or data readable by a processor may be stored. Thus, a computer-readable storage medium may store instructions for execution by one or more processors, including instructions for causing the processor(s) to perform steps or stages consistent with the embodiments described herein. The term "computer-readable medium" should be understood to include tangible items and exclude carrier waves and transient signals, i.e., non-transitory. Examples include Random Access Memory (RAM), Read-Only Memory (ROM), volatile memory, nonvolatile memory, hard drives, Compact Disc (CD) ROMs, Digital Video Disc (DVDs), flash drives, disks, and any other known physical storage media.

[0110] Certain aspects may comprise a computer program product for performing the operations presented herein. For example, such a computer program product may comprise a computer readable media having instructions stored (and/or encoded) thereon, the instructions being executable by one or more processors to perform the operations described herein. For certain aspects, the computer program product may include packaging material.

[0111] Various components, modules, or units are described in this disclosure to emphasize functional aspects of devices configured to perform the disclosed techniques, but do not necessarily require realization by different hardware units. Rather, as described above, various units may be combined in a hardware unit or provided by a collection of interoperative hardware units, including one or more processors as described above, in conjunction with suitable software and/or firmware.

[0112] Finally, the language used in the specification has been principally selected for readability and instructional purposes, and it may not have been selected to delineate or circumscribe the inventive subject matter. It is therefore intended that the scope of the invention be limited not by this detailed description, but rather by any claims that issue on an application based here on. Accordingly, the embodiments of the present invention are intended to be illustrative, but not limiting, of the scope of the invention, which is set forth in the appended claims.

WE CLAIM:

1. A method (1000) for evaluating one or more functional attributes of an application, the method (1000) comprising:

capturing (1002), by a processor (204) of a computing device (110), one or more screenshots of the application;

processing (1004), by the processor (204) using a trained model (210), the one or more captured screenshots to detect at least one object within the application;

generating (1006), by the processor (204) using the trained model (210), a first report comprising a label and a pair of coordinates corresponding to each of the at least one detected object;

generating (1008), by the processor (204), a second report comprising the label and a locator corresponding to each of the at least one detected object;

generating (1010), by the processor (204), a third report based on the first report and the second report, wherein the third report comprises the label, the pair of coordinates, and the locator corresponding to each of the at least one detected object; and

executing (1012), by the processor (204), one or more pre-generated test scripts (214) based at least on the content of the third report to evaluate the one or more functional attributes of the application.

2. The method of claim 1, further comprising:

receiving a plurality of screenshots corresponding to a plurality of applications;

augmenting the plurality of screenshots by performing one or more image augmentation operations on the plurality of screenshots and by adjusting one or more parameters of the plurality of screenshots;

annotating the plurality of augmented screenshots by assigning labels to different objects present in the plurality of augmented screenshots; and

training the model by iteratively processing the plurality of annotated screenshots using machine learning techniques.

3. The method of claim 1, further comprising:

receiving a test case document, wherein the test case document comprises data related to one or more test cases for evaluating the one or more functional attributes of the application;

processing the received test case document to identify one or more keywords relevant to said one or more test cases; and

generating the one or more test scripts by correlating the identified one or more keywords with a plurality of predefined keywords.

4. The method of claim 1, wherein the locator uniquely locates the detected object within the application, and wherein the locator comprises an xpath of the detected object.

5. The method of claim 1, wherein evaluating the one or more functional attributes of the application comprises:

comparing result of the execution of the one or more pre-generated test scripts with an expected result; and

determining that the one or more functional attributes of the application are working as expected based on the comparing.

6. A computing device (110) for evaluating one or more functional attributes of an application, the computing device (110) comprising:

a memory (206); and

at least one processor (204) communicatively coupled with the memory (206), wherein the at least one processor (204) is configured to:

capture one or more screenshots of the application;

process the one or more captured screenshots using a trained model (210) to detect at least one object within the application;

generate, using the trained model (210), a first report comprising a label and a pair of coordinates corresponding to each of the at least one detected object;

generate a second report comprising the label and a locator corresponding to each of the at least one detected object;

generate a third report based on the first report and the second report, wherein the third report comprises the label, the pair of coordinates, and the locator corresponding to each of the at least one detected object; and

execute one or more pre-generated test scripts (214) based at least on the content of the third report to evaluate the one or more functional attributes of the application.

7. The computing device of claim 6, wherein the at least one processor is further configured to:

receive a plurality of screenshots corresponding to a plurality of applications;

augment the plurality of screenshots by performing one or more image augmentation operations on the plurality of screenshots and by adjusting one or more parameters of the plurality of screenshots;

annotate the plurality of augmented screenshots by assigning labels to different objects present in the plurality of augmented screenshots; and

train the model by iteratively processing the plurality of annotated screenshots using machine learning techniques.

8. The computing device of claim 6, wherein the at least one processor is further configured to:

receive a test case document, wherein the test case document comprises data related to one or more test cases for evaluating the one or more functional attributes of the application;

process the received test case document to identify one or more keywords relevant to said one or more test cases; and

generate the one or more test scripts by corelating the identified one or more keywords with a plurality of predefined keywords.

9. The computing device of claim 6, wherein the locator uniquely locates the detected object within the application, and wherein the locator comprises an xpath of the detected object.

10. The computing device of claim 6, wherein to evaluate the one or more functional attributes of the application, the at least one processor is configured to:

compare result of the execution of the one or more pre-generated test scripts with an expected result; and

determine that the one or more functional attributes of the application are working as expected based on the comparing.

Dated this 17th day of June 2022

-- Digitally Signed--
Bhanu Prasad
(INPA No: 3253),
Manager, IPR Dept.,
L&T Technology Services Limited,
DLF 3rd Block, 2nd Floor,
Manapakkam, Chennai - 600089.

ABSTRACT

SYSTEM AND METHOD FOR AUTOMATED SOFTWARE TESTING

The present disclosure describes a method (1000) and a device (110) for evaluating one or more functional attributes of an application. The device (110) comprises a processor (204) configured to capture screenshot(s) of the application and use a trained model (210) to detect object(s) within the application. The processor (204) is further configured to generate a first report comprising a label and a pair of coordinates corresponding to each of the detected object(s) and a second report comprising the label and a locator corresponding to each of the detected object(s). The processor (204) is configured to generate a third report comprising the label, the pair of coordinates, and the locator corresponding to each of the one detected object(s). The processor (204) is further configured to execute pre-generated test script(s) (214) based at least on the content of the third report to evaluate the functional attributes of the application.

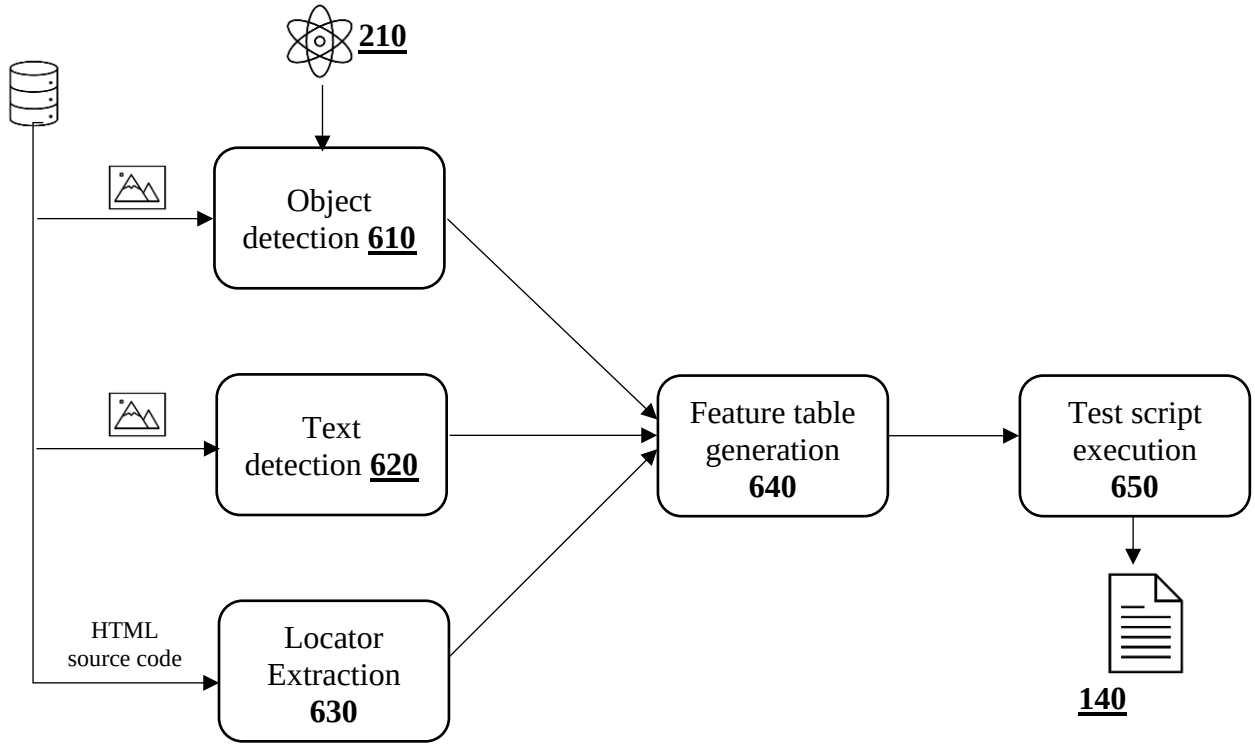

Figure 6

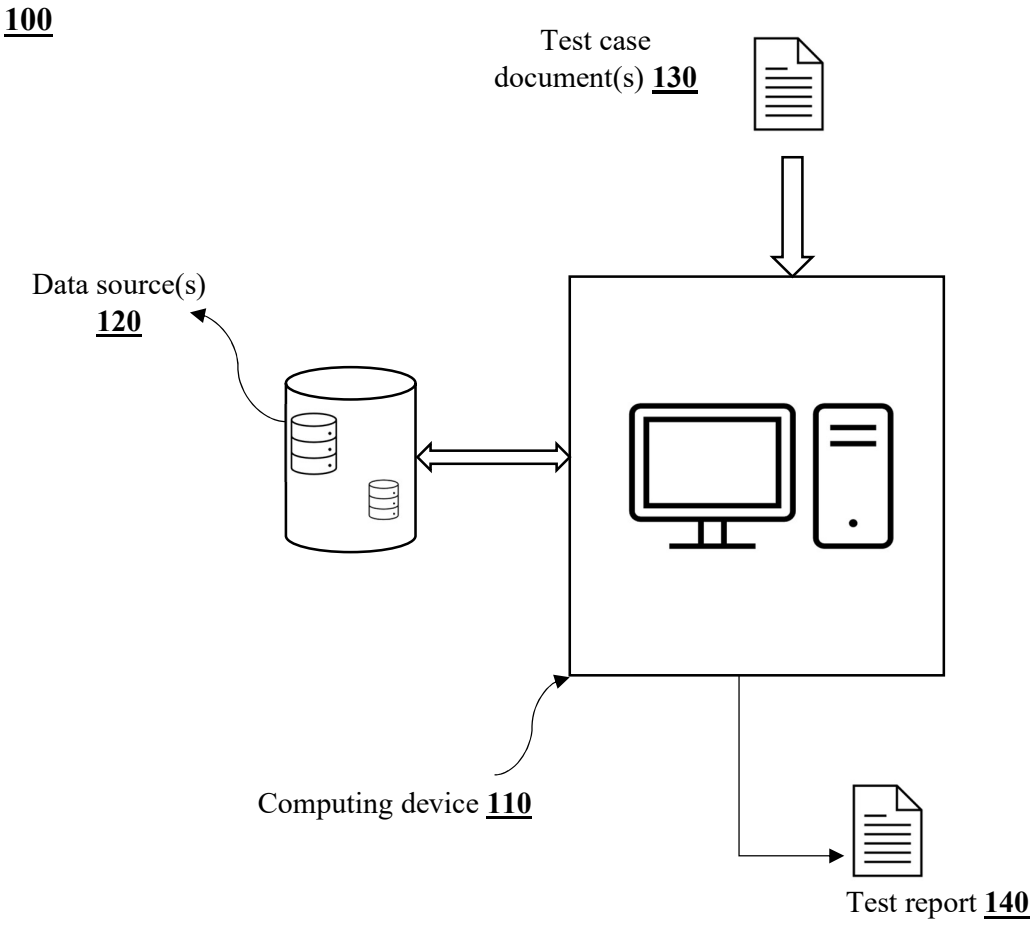

Figure 1

-- Digitally Signed--
Bhanu Prasad
(INPA No: 3253)
Manager, IPR Dept.,
L&T Technology Services Limited,
DLF 3rd Block, 2nd Floor,
Manapakkam, Chennai - 600089.

200

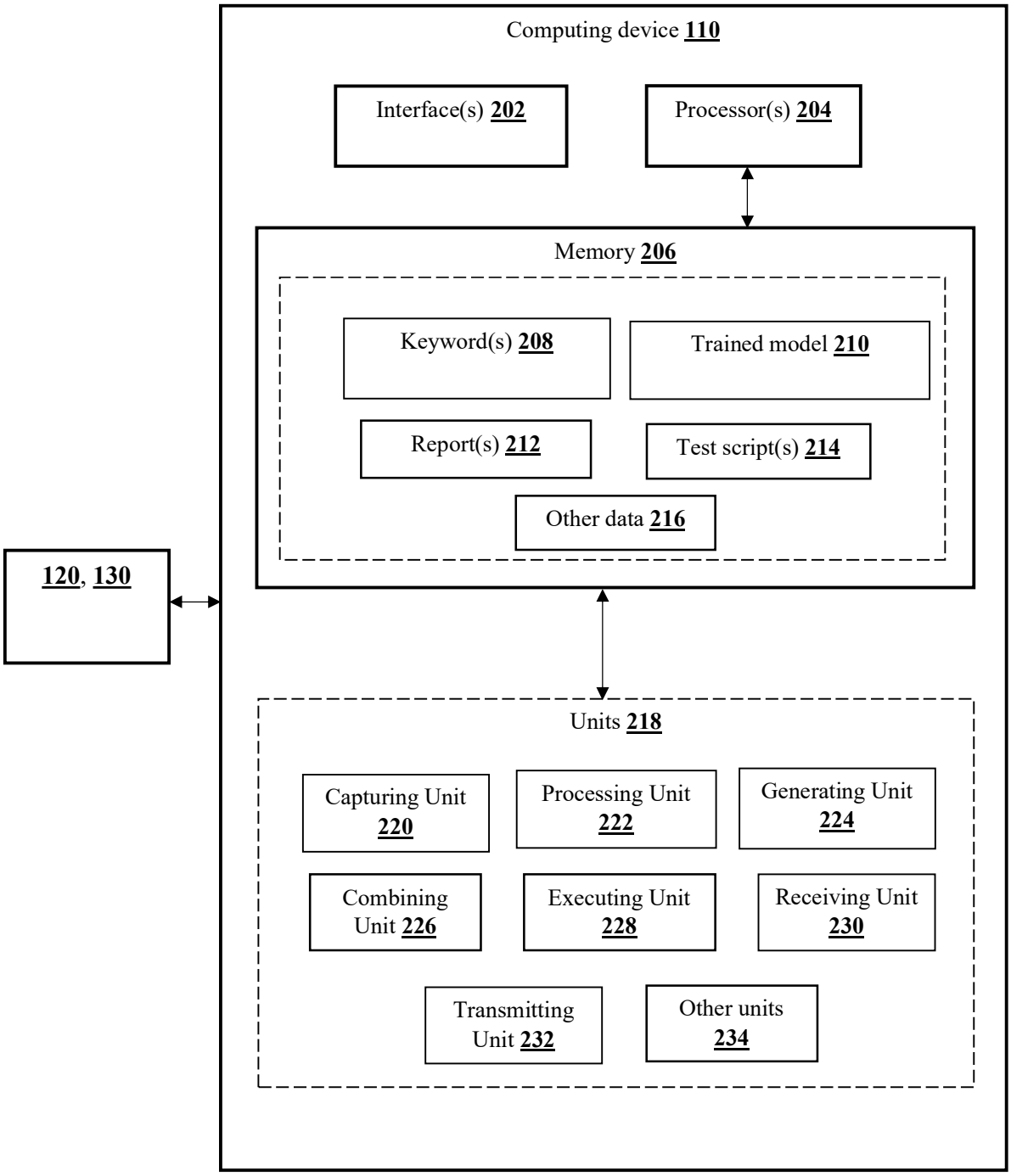

Figure 2

-- Digitally Signed--
Bhanu Prasad
(INPA No: 3253)
Manager, IPR Dept.,
L&T Technology Services Limited,
DLF 3rd Block, 2nd Floor,
Manapakkam, Chennai - 600089.

300-1

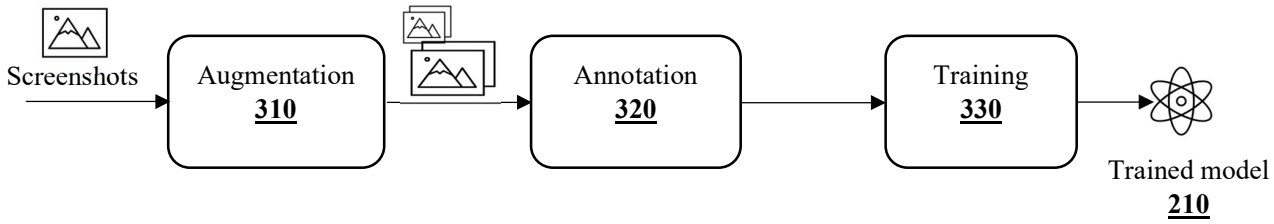

Figure 3(a)

300-2

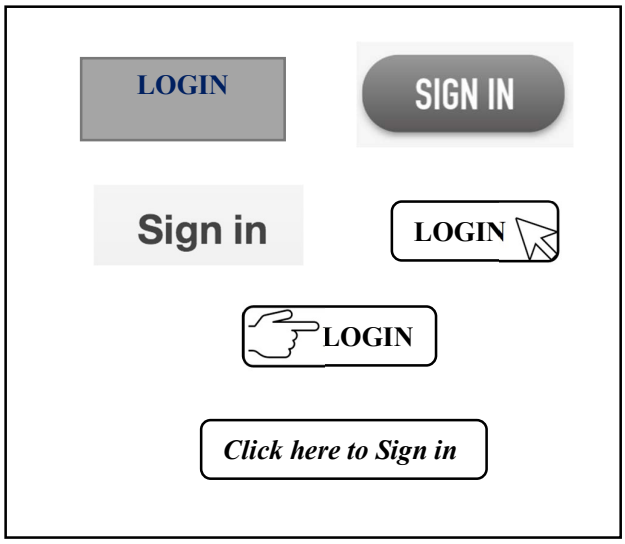

Figure 3(b)

400

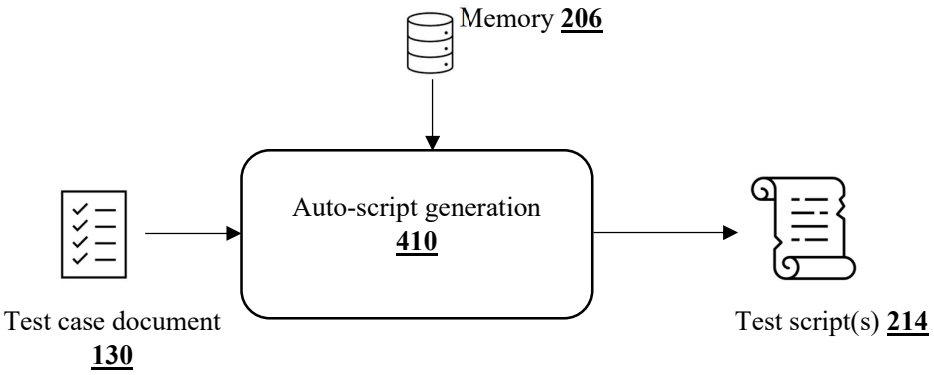

Figure 4

-- Digitally Signed--
Bhanu Prasad
(INPA No: 3253)
Manager, IPR Dept.,
L&T Technology Services Limited,
DLF 3rd Block, 2nd Floor,
Manapakkam, Chennai - 600089.

500-1

Sr. No.	Title	Test Step	Action	Input	Expected Result
1.	As a user, I want to verify if I can login the application				
		1	Launch Application	https://www.testsite.com/	Home Page
		2	Enter email Enter password Hit the login button	Email id: testsite@gmail.com Password: test@123	Login Successful

Figure 5(a)

500-2

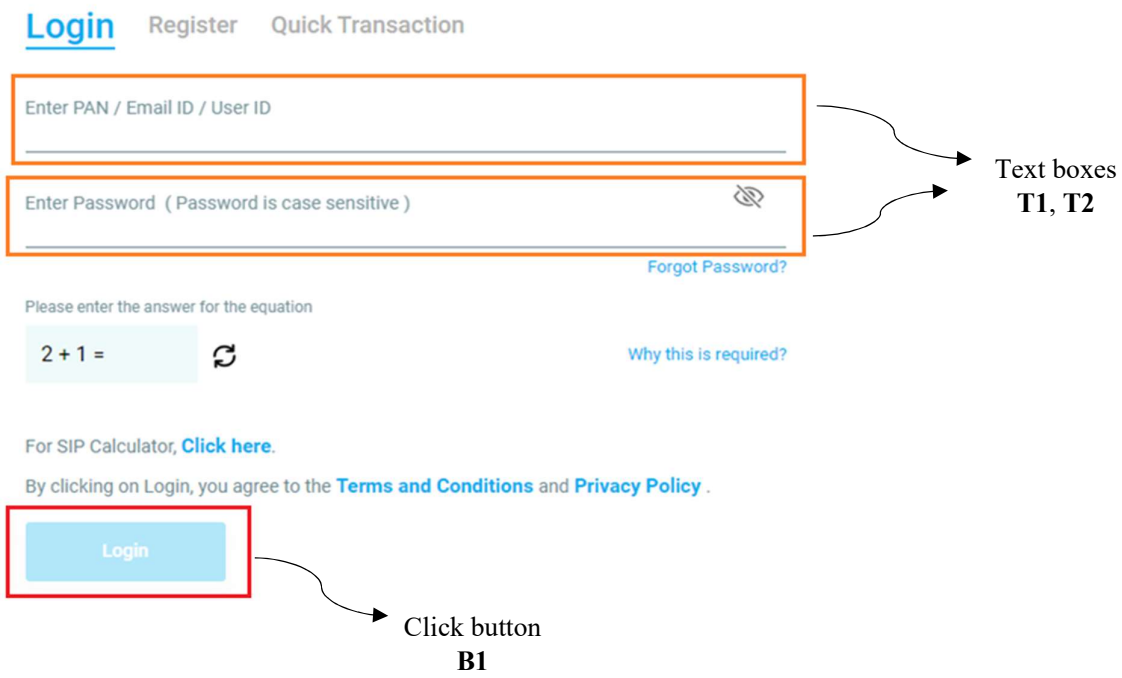

Figure 5(b)

-- Digitally Signed--
Bhanu Prasad
(INPA No: 3253)
Manager, IPR Dept.,
L&T Technology Services Limited,
DLF 3rd Block, 2nd Floor,
Manapakkam, Chennai - 600089.

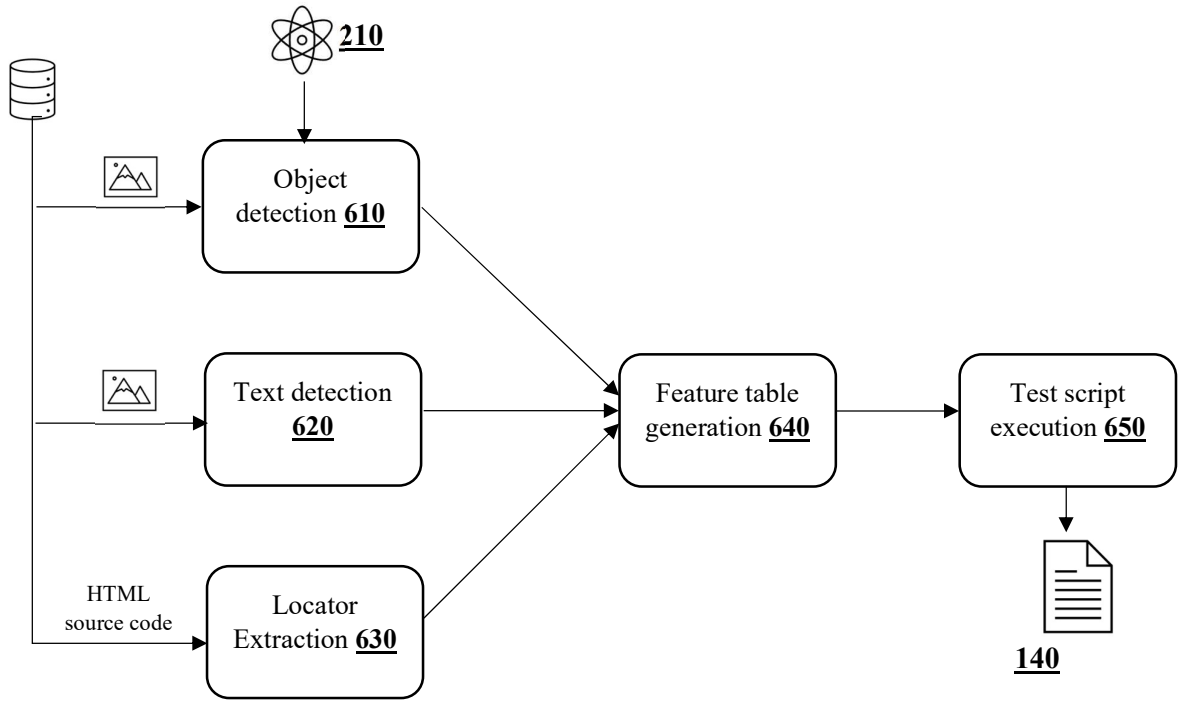

Figure 6

-- Digitally Signed--
Bhanu Prasad
(INPA No: 3253)
Manager, IPR Dept.,
L&T Technology Services Limited,
DLF 3rd Block, 2nd Floor,
Manapakkam, Chennai - 600089.

700

Screenshot	Object Label	Xmin	Ymin	Xmax	Ymax	Confidence
input1.png	textbox1	598	14	677	63	0.86225
input1.png	textbox2	1752	13	1811	69	0.89926
input1.png	clickbutton	1214	14	1575	66	0.86029

Figure 7

800

Object label	Coordinates
clickhere	133.0,192.0
privacypolicy	345.0,193.0

Figure 8

900

Object label	xpath
textbox1	//input[@id='userName']
textbox2	//*[@id=’password’]
clickbutton	//input[@id='clickbutton']
clickhere	//*[@id='clickhere']

Figure 9

-- Digitally Signed--
Bhanu Prasad
(INPA No: 3253)
Manager, IPR Dept.,
L&T Technology Services Limited,
DLF 3rd Block, 2nd Floor,
Manapakkam, Chennai - 600089.

1000

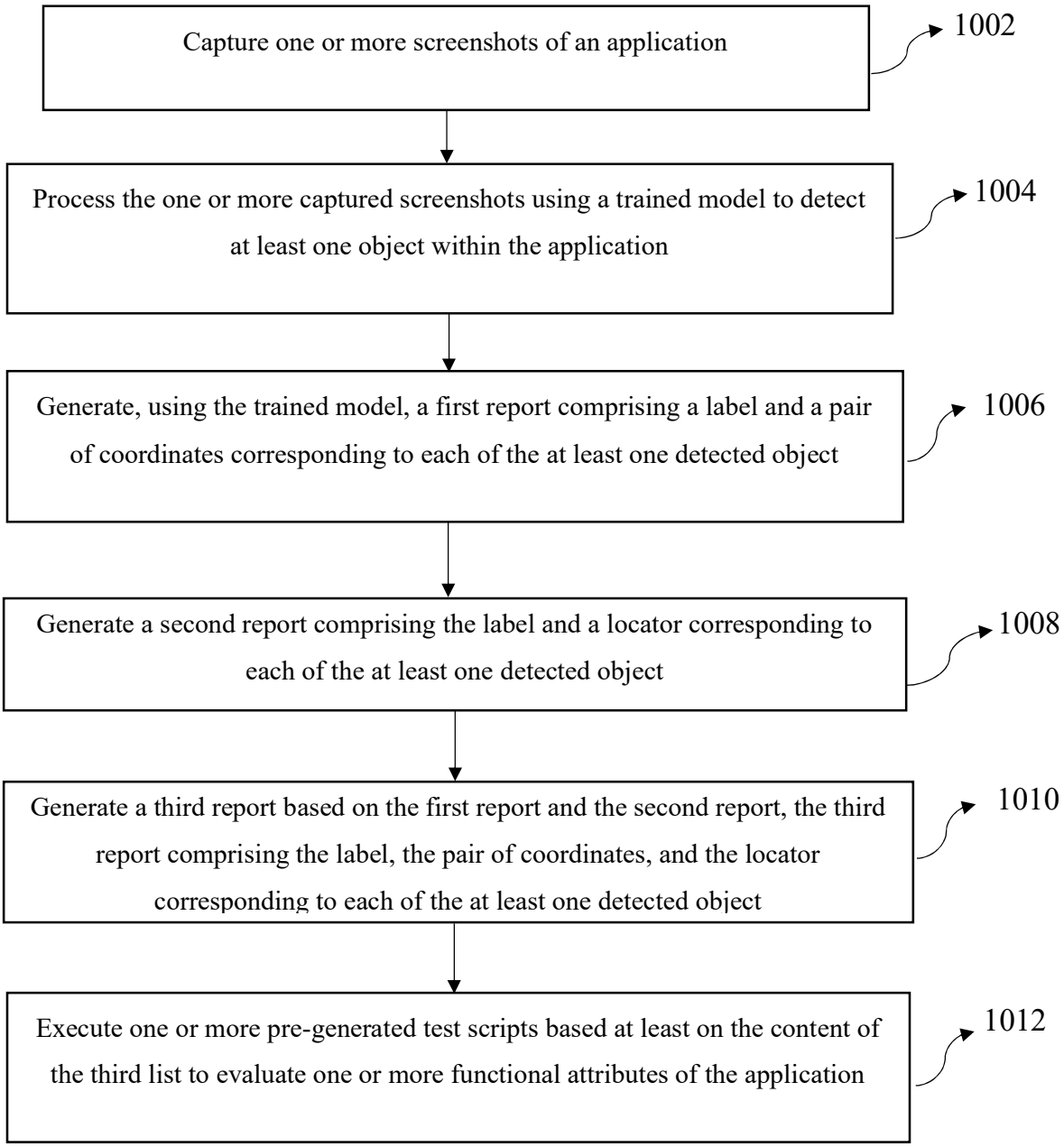

Figure 10

-- Digitally Signed--
Bhanu Prasad
(INPA No: 3253)
Manager, IPR Dept.,
L&T Technology Services Limited,
DLF 3rd Block, 2nd Floor,
Manapakkam, Chennai - 600089.