Automatic Construction of Graphical User Interfaces Semantic Models Using Robots for Mobile Application Testing

Feng XUE[1,2], Qingying LIU[2], Tao ZHANG[2]*, Shaoying LIU[3], Jing CHENG[4], Chunyan MA[2]

[1] School of Information, Xi'an University of Finance and Economics,
360 Changning Street, Changan District, Xi'an, 710100, China
120150792@qq.com

[2] School of Software, Northwestern Polytechnical University,
1 Dongxiang Road, Changan District, Xi'an, 710129, China
qyliu@mail.nwpu.edu.cn, tao_zhang@nwpu.edu.cn (**Corresponding author*),
machunyan@nwpu.edu.cn

[3] Graduate School of Advanced Science and Engineering, Hiroshima University, 1 Chome-3-2 Kagamiyama, Higashi Hiroshima City, 739-8527, Hiroshima, Japan
sliu@hiroshimau.ac.jp

[4] School of Computer Science and Engineering, Xi'an Technological University,
2 Xuefuzhong Road, Weiyang District, Xi'an, 710021, China
chengjing@xatu.edu.cn

Abstract: With the growing adoption of rich GUIs (Graphical User Interfaces) in mobile applications, researchers and practitioners have focused on GUI-based testing of the applications. Constructing a GUI model for AUT (Application Under Test) is a conventional strategy; nevertheless, automated modelling is typically grounded in source code logic, often lacking representation of functional semantics. However, manually constructing a GUI model enriched with semantics is inefficient. In this work, a GUI semantic model is proposed, namely FSM-ES (Finite State Machine with Extended Semantics), which not only reflects the syntactical structure of the GUI, but also the semantics of its elements. Moreover, visual technology is used to capture GUI information and use semantic ontology to guide robots in order to complete gesture actions such as clicking and sliding on the GUI, achieving the establishment of FSM-ES. The case study demonstrates that, while ensuring coverage of the core functions of the AUT, GUI semantic model proposed in this paper is 52% faster than manual modelling. Moreover, it facilitates the generation of test cases focused on functional semantics.

Keyword: Robotic testing, GUI semantic model, Automated modelling.

1. Introduction

With the rise of data-driven computational methods for modelling user interactions with GUIs (Graphical User Interfaces), the GUIs have become not only interfaces for human users to interact with the underlying computing services, but also valuable data sources that encode the underlying task flow, the supported user interactions, and the design patterns of the corresponding applications (Li et al., 2021; Abulhaija et al., 2022). Hence, researchers and practitioners have focused on GUI-based testing of the applications, including the GUI itself.

Manual GUI-based testing is inefficient, costly and difficult to ensure test coverage. By contrast, automated GUI-based testing is more effective (Mao, Harman & Jia, 2017). GUI models support automatic test case generation for GUI-based automated testing. These models are typically created after the application has been built. The ability to automatically generate large sets of different test cases, which have a greater possibility of identifying defects under deep paths, is one advantage of this automated test case generation over manual test case creation. Moreover, these models can serve as documentation to help stakeholders understand the scope of the tests, and what is being tested, because they capture the desired behaviour of the AUT (Application Under Test). Hence, creating accurate GUI models for AUTs is a critical step in GUI-based automated testing.

Despite the numerous advantages of the GUI models, their adoption in industry remains slow. One obstacle that stands in the way of wide adoption is that testers and software engineers are reluctant to manually create models, because it is time-consuming and costly (Kong et al., 2018). Hence, automation of modelling approaches has become essential.

Existing automated modelling approaches are mainly based on application demonstrations,

https://doi.org/10.24846/v33i1y202407

test cases, or source code. The approach based on application demonstrations (Xu et al., 2021; Kong et al., 2018) extracts information using a GUI framework that requires platform-specific customization. The approach based on test cases (Lin, Jabbarvand & Malek, 2019) requires the tester to create a set of test cases and automatically construct models from the set. The approach based on source code (Reis & Mota, 2018; Bera et al., 2021) reverse engineers the model from the source code, and the generated model shows only a minimal relationship with the mobile application GUI.

The above-mentioned modelling techniques are invasive to the devices under test. These techniques require support from the underlying OS (Operating System) or GUI framework of an AUT to obtain GUI states and trigger GUI actions. If the AUT is running on a closed system with no underlying OS or GUI framework support accessible or an uncommon platform whose underlying system support is hard to access, then such a modelling approach may be difficult to apply (Qian et al., 2020). Although the underlying systems might be accessible for some AUTs, hacking into these systems to enable modelling may change the tested environments to a certain extent, and the GUI actions triggered via internal system facilities may not closely emulate the experience of real users. Under such circumstances, the modelling results of the AUT may not be trustworthy.

Non-invasive modelling techniques are necessary for mobile applications that do not require or provide access to underlying system support. Given that robots have the potential to perform human-like operations (He et al., 2023; Li, Shi & Hwang, 2023; Shi et al., 2023) and achieve non-invasive exploration of the AUT, a possible way is to utilize them to undertake modelling. This idea stems from the fact that robots have been widely used in the field of GUI-based testing in recent years (Pan et al., 2020).

Robotic testing inherently provides a fully non-invasive approach to automated GUI-based testing of mobile applications. In practice, robots with a high-definition camera that simulates the human eye and a robotic arm that simulates the human arm are available for testing, as shown in Figure 1, as well as intelligent algorithms that emulate the human brain (Pan et al., 2020). However, in non-invasive modelling and testing based on robots, where reliance on source code is absent, a heightened necessity exists to comprehend the GUI semantics of the AUT. Traditional GUI testing models, such as FSM (Finite State Machines) (Miao & Yang, 2010), EFG (Event Flow Graphs) (Huang & Lu, 2012), unified modelling language (Lafi, Alrawashed & Hammad, 2011), etc., lack the expression of functional semantic information of the AUT (Kull, 2012; Liu et al., 2018). Moreover, manually constructing a semantically rich testing model is inefficient (Salihu et al., 2019).

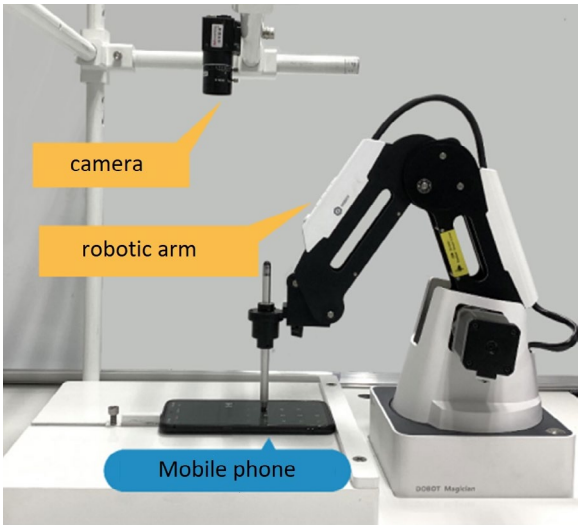

Figure 1. Robotic testing system

This paper addresses these limitations by extending the semantics for the GUI model. The DAOM (Domain Application Ontology Model) is built for specific domains based on the textual semantic description of mobile applications (Song, 2021). Additionally, AFG (Action Flow Graph) is incorporated to delineate the interactive behaviour of mobile applications, thereby shaping the FSM-ES model (Finite State Machine with Extended Semantics). Moreover, the mapping from AFG to FSM-ES produces a TSG (Task Sub-Graph), which can support the generation of function-related test cases, thus effectively shortening the test sequence. A case study is also conducted on an application running on a touch screen device. The result shows that using robots to construct FSM-ES provides an efficient way to create initial GUI models with extended semantics. This research from this paper provides the following major contributions:

- FSM-ES is proposed as a language and mechanism for representing mobile application GUI models that can facilitate non-invasive, automated GUI-based testing of the GUI itself and the related applications;

- A specific approach is presented to automatically constructing an FSM-ES for the AUT using robots and a case study to demonstrate its feasibility.

The remainder of this paper is organized as follows. Section 2 introduces the related work. Section 3 discusses the proposed model. Section 4 presents the FSM-ES modelling approach using robots. Section 5 exhibits the case study. Finally, Section 6 presents the conclusion of the present work.

2. Related Work

2.1 Reverse Engineering-based GUI Modelling

A number of GUI modelling approaches have emerged in the automated GUI testing. The most common approach is based on reverse modelling engineering of system demonstrations. Nguyen et al. (2014) developed the GUITAR tool, which dynamically reverse engineers the EFG of a system by automatically capturing the GUI. The obtained model represents all possible actions that the user can perform in that GUI system. Imparato (2015) used the SlumDroid tool, which uses the Robotium framework to simulate the user's interaction with the Android device and extracts the executable tasks stored in the task list. A challenge of this approach is to provide a specific input to text fields without user instructions.

2.2 Test Cases-based GUI Modelling

GUI models can also be constructed using test cases. Schulze et al. (2015) developed a GUI model construction approach that allows testers to first create and debug a set of test cases. The proposed method automatically constructs a model from test cases that the tester is satisfied with. The constructed model is derived from the test cases, which are the actions that the system can perform (e.g., clicking a button) and their expected outputs in the form of assertion statements (e.g., asserting data inputs). Lin, Jabbarvand & Malek (2019) proposed CRAFTDROID, a framework that uses information retrieval, static analysis, and dynamic analysis techniques to extract from an application's existing test suite human knowledge and to pass test cases and predictions to other applications with similar functionality for testing. The test case-based approaches require testers to write and maintain a set of executable test cases (Tan et al., 2022).

2.3 Source Code-based GUI Modelling

Another popular approach used to construct GUI models is to observe the state of a system or its abstract representation based on a source code. Marchetto et al. (2008) proposed an approach to abstract the document object model of a web application into a state model and generate additional test cases. They applied their technique on an open-source to-do list manager application with seed defects. Reis & Mota (2018) used Socket to create GUI models from a Java/Swing source code. However, the state in the proposed source code-based approach has only a small relationship with GUIs.

The above testing methods greatly promote the automation of mobile application testing. However, with the diversity of applications system platforms and the diversity of its numbers, the above methods cannot be effectively applied to modelling and testing of different types of applications. For example, for the same application, it is still highly complex to model it multiple times on different platform versions (Android, iOS, web, etc.). The method proposed in the present paper does not rely on the application platform, but revolves around the functional meaning reflected in the GUI to complete application modelling. It can better support cross platform testing while achieving functional test case generation.

3. Model

The traditional FSM-based GUI models are characterized by using nodes to represent GUI states and edges to describe interactions. However, these representations do not contain semantics of the nodes and edges. This work organizes all the semantic information related to a domain, expresses it in DAOM and proposes an FSM with extended semantics, called FSM-ES. The model structure is illustrated in Figure 2.

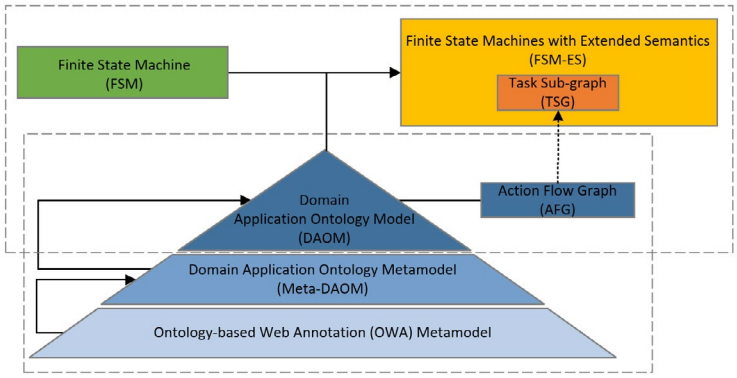

Figure 2. Model structure

Domain application ontology metamodel (Meta-DAOM) is constructed on the basis of the OWA (Ontology-based Web Annotation) metamodel (Naing, Lim & Goh, 2003; Mandić, 2022) to provide a complete formal definition of DAOM. Such a DAOM is an instance of Meta-DAOM oriented to a certain domain. To generate function-related test cases, the tasks are described in the DAOM using AFGs. Then, the AFGs are mapped to FSM-ES to generate TSGs. According to the TSG, function-related test cases can be generated automatically. The details of the model are discussed in the following subsections.

3.1 Ontology-based Web Annotation (OWA) Metamodel and Meta-Domain Application Ontology Model (DAOM)

The ontology is an explicit formal specification of a shared conceptual model (Gelfert, 2017). The central role of ontology is to define the specialized vocabulary within a domain and the relationships between things within the domain represented by the vocabulary. The OWA metamodel is a typical formal definition of the ontology, and it provides a complete formal definition for describing the semantics of more complex objects. In this work, the OWA metamodel is used to describe the ontology, as defined below.

Definition 1: Ontology O is a six-tuple, $O=\{C, A^C, R, A^R, H, X\}$, where:

- C is the set of concepts related to the domain;
- A^C is the set of attributes of each concept. The set of concept attributes $A^C(c_i)$ means that each concept c_i in the set of concepts C is used to represent a set of objects of similar kind and can be described by the same set of attributes;
- R is the set of relations between concepts. Relation $r_i(c_p, c_q)$ means that each relation r_i in relation R represents a binary relation between concepts C_p and C_q. The instances of this relation are a pair of concept objects (c_p, c_q);
- A^R is the set of attributes of each relation. The set of relation attributes $A^R(r_i)$ represents the attributes of relation r_i;
- H is the hierarchy of concept set C, which represents a set of parent–child relations between concepts in C;
- X is the set of axioms. Each axiom in X is a constraint on the attribute values of concepts and the attribute values of relations, or a constraint on the relationship between concepts.

Meta-DAOM is a domain-independent and generic model that mainly provides formal definitions for domain-dependent DAOMs and guidance for constructing DAOMs based on the OWA metamodel.

Definition 2: The domain application ontology metamodel (Meta-DAOM) O_{Meta} is a six-tuple, $O_{Meta}=\{C_{Meta}, A^C_{Meta}, R_{Meta}, A^R_{Meta}, H_{Meta}, X_{Meta}\}$, where:

- $C_{Meta}=\{c_E, c_A, c_T\}$;
- $A^C_{Meta}=\{A^C_{Meta}(c_E), A^C_{Meta}(c_A), A^C_{Meta}(c_T)\}$;
- $R_{Meta}=\{\{INCL(c_p,c_q)\mid c_p\in c_E\wedge c_q\in c_E\},$
  $\{ASSC(c_p,c_q)\mid c_p\in c_E\wedge c_q\in c_E\},$
  $\{IN(c_p,c_q)\mid c_p\subseteq c_E\wedge c_q\in c_A\},$
  $\{OUT(c_p,c_q)\mid c_p\in c_A\wedge c_q\subseteq c_E\},$
  $\{INCL(c_p,c_q)\mid c_p\in c_T\wedge c_q\in c_A\},$
  $\{SEQ(c_p,c_q)\mid c_p\in c_T\wedge c_q\in c_T\}\}$;
- $A^R_{Meta}=\{A^R_{Meta}(IN), A^R_{Meta}(OUT), A^R_{Meta}(INCL),$
  $A^R_{Meta}(ASSC), A^R_{Meta}(SEQ)\}$;

- $H_{Meta} = \{(c_E, c_A), (c_A, c_T)\}$;
- $X_{Meta} = \{X_{Meta}(c_E), X_{Meta}(c_A), X_{Meta}(c_T)\}$.

c_E is a set of conceptual nouns common to domain applications, called entities.

c_A is a set of atomic operations for mobile applications. Atomic operations of this type are called actions. Once an action occurs, it cannot be interrupted. Precisely, it is defined as follows.

Definition 3: For $\forall \alpha \in c_A$, action α is a triplet, $\alpha = \{s_k^s, Op, ac^s\}$, where:

- s_k^s is the semantics of the GUI state s_k in which the action is executed;
- O_p is the operation necessary to implement the action, such as a click;
- ac_s is the semantics of the GUI element operated in the action.

The semantics of GUI states and elements will be described in detail in subsection *3.4 Finite State Machine with Extended Semantics (FSM-ES)*.

c_T is the set of tasks that correspond to the functional requirements.

Definition 4: For $\forall \tau \in c_T$, task τ is a quaternion, $\tau = \{N, s_I^s, D, s_F^s\}$, where:

- N is the name of the task;
- s_I^s is the semantics of the initial GUI state s_I of the task;
- D is the corresponding AFG for the task, which describes a set of actions and the relationships between the actions;
- s_F^s is the semantics of the finish GUI state s_F of the task.

The other involved notations are explained as follows:

$INCL(c_p, c_q)$ indicates that entity C_p includes entity C_q for $\forall c_p \in c_E$, $\forall c_q \in c_E$. For example, $c_{username} \in c_E$, $c_{user} \in c_E$, $INCL(c_{user}, c_{username})$ indicate that username is an included concept in user.

$ASSC(c_p, c_q)$ indicates that entity C_p is associated with entity C_q for $\forall c_p \in c_E$, $\forall c_q \in c_E$. All other relationships between entities that are not inclusion relationships are association relationships. $ASSC(c_p, c_q) = ASSC(c_q, c_p)$ and $INCL(c_p, c_q) \cap ASSC(c_p, c_q) = \varnothing$.

$IN(c_p, c_q)$ indicates that entity C_p is an input to action C_q for $\forall c_p \in c_E$, $\forall c_q \in c_A$, $c_q = \{s_k^s, Op, ac^s\}$, that is, $Op = input$ and $ac^s = c_p$.

$OUT(c_p, c_q)$ indicates that the output of action C_p is the entity C_q for $\forall c_p \in c_A$, $\forall c_q \in c_E$.

$INCL(c_p, c_q)$ indicates that the task C_p contains the action C_q for $\forall c_p \in c_T$, $\forall c_q \in c_A$.

$SEQ(c_p, c_q)$ indicates that task C_q can only be executed after the task C_q for $\forall c_p \in c_T$, $\forall c_q \in c_T$.

3.2 Action Flow Graph (AFG)

To generate function-related test cases, a new structure is used in DAOM to model the execution behaviour of GUIs, called action flows, each of which corresponds to a task and is represented by AFGs. The structure is described as follows.

The AFG describes the control flow from action to action. Actions are indivisible, and transitions between actions are triggered by the completion of previous actions. After the completion of an action, the control flow will immediately reach the next action. Transition is used to represent this flow from one action to another.

Definition 5: For $\forall \tau \in c_T, \tau = \{N, s_I^s, D, s_F^s\}$, and the AFG D is a six-tuple, $D = \{A, T, F, G, \alpha_I, \alpha_F\}$, where:

- $A = \{\alpha_1, \alpha_2, ..., \alpha_m\}$ is a finite set of action;
- $T = \{t_1, t_2, ..., t_n\}$ is a finite set of transitions;
- $F \subseteq (A \times T) \cup (T \times A)$ is a set of flow relations;
- $G(t)$ is the conditional expression for the transition t;
- $\alpha_I \in A$, is the initial action, $\alpha_I = \{s_I^s, Op, ac^s\}$;
- $\alpha_F \in A$, is the initial action, $\alpha_F = \{s_F^s, Op, ac^s\}$.

Only one transition t satisfies $(\alpha_I, t) \in F$ and for any $t' \in T$, $(t', \alpha_I) \notin F$ and $(\alpha_I, t') \notin F$.

Definition 6: $D = \{A, T, F, G, \alpha_I, \alpha_F\}$ is an AFG. The current action μ in D is an arbitrary subset of A. For $\forall t \in T$, $\bar{t} = \{a \in A \mid (a, t) \in F\}$ and $\vec{t} = \{a \in A \mid (t, a) \in F\}$ denote the precursor action and the successor action of t, respectively.

A transition t is triggered for action μ if $\vec{t} \subseteq \mu$, and the value of $G(t)$ is true; otherwise it is not triggered. The set of μ-triggerable transitions is denoted by $enabled(\mu)$.

At a certain point in time, a triggerable transition must satisfy the following conditions:

Definition 7: $D = \{A, T, F, G, \alpha_I, \alpha_F\}$ is an AFG. The transition $t \in T$ can be triggered from the action μ when and only when $t \in enabled(\mu)$ and $(\mu - \bar{t}) \cap \vec{t} = \varnothing$, at which point the new state $\mu' = (\mu - \bar{t}) \cup \vec{t}$. When an action that a GUI can receive is executed, the state will transit.

Definition 8: $D = \{A, T, F, G, \alpha_I, \alpha_F\}$ is an AFG. A run σ of D is a sequence of actions and transitions $\sigma = \mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{n-1}} \mu_n$, where:

- $\mu_0 = \{\alpha_I\}$;
- $\mu_n = \{\alpha_F\}$;
- $t_i \in enable(\mu), i \geq 0$;
- $\mu_i = (\mu_{i-1} - \bar{t}_{i-1}) \cup \vec{t}_{i-1}, i \geq 1$.

According to the above definition, the basic elements of AFG are initial action, finish action, intermediate action, transition, branch, bifurcation, and convergence. In the AFG, the hollow circle indicates the initial action; the hollow ring is the finish action; the rectangular box denotes the intermediate action; and the solid line with arrows represents the transition (Figure 3). The branches in the AFG are represented by diamonds, and they can have one incoming degree and multiple outgoing degrees. On each outgoing transition, a Boolean expression is indicated, and the diamond in AFG without Boolean expression is indicated as a merge node. Synchronization bars are used in the AFG to handle the bifurcation and convergence of the parallel control flow. This flow means that the actions are executed in no sequence. The synchronization bar is represented by a thick line. The bifurcation has one incoming degree and multiple outgoing degrees, each representing a separate control flow. Actions in different control flows behind the fork are concurrently executed. Convergence can have multiple degrees and one outgoing degree. Before the convergence, actions on different control flows are concurrently executed. When the merge is reached, the concurrent control flows are synchronized, that is, each control flow waits until all control flows have arrived before triggering the next action.

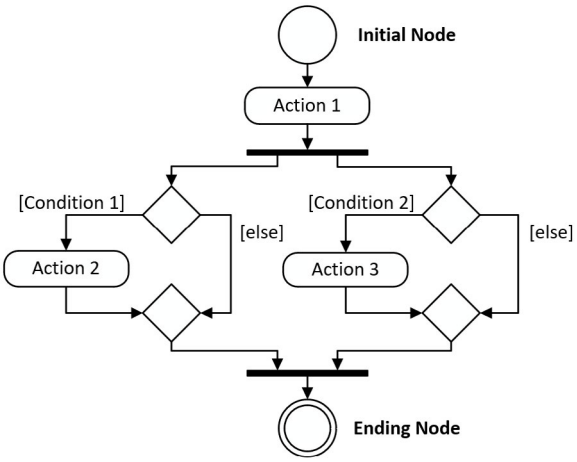

Figure 3. Schematic of AFG

3.3 Domain Application Ontology Model (DAOM)

DAOM is an instance of Meta-DAOM oriented to a certain domain. Meta-DAOM serves as a fundamental model, defining solely the essential elements required by the domain model. In contrast, DAOM constitutes semantic modelling tailored for specific application scenarios. The semantics of GUI elements are defined manually based on Meta-DAOM. A partial example of a DAOM is shown in Figure 4, corresponding to the login function. Three entities are involved, which are user, username, and password with the user containing the username and password. Moreover, three actions are involved, entering the username in the GUI state s_1 with semantics s_1^s, entering the password in the GUI state s_2 with semantics s_2^s and clicking the login button in the GUI state s_3 with semantics s_3^s. Username and password are the inputs for the first two actions. A task, which is Login, contains three actions. AFG(Login) indicates that the inputting the username and password are simultaneously executed in no sequence. When entering a password, the two options are correct and incorrect. Then, Login is clicked after completing the input. AFG defines possible behavioural pathways rather than specific processes. For example, login may not require the order of entering usernames and passwords.

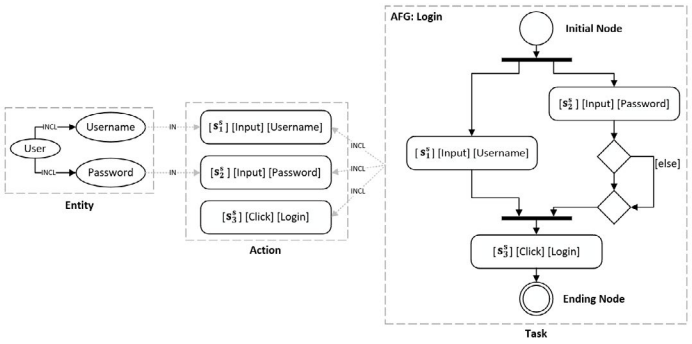

Figure 4. DAOM corresponding to the login function

3.4 Finite State Machine with Extended Semantics (FSM-ES)

The FSM-ES consists of GUI state nodes and interaction edges with their semantics. Semantics are provided by DAOM, where the entities and their relationships provide semantics for nodes, actions provide semantics for edges, and tasks provide semantics for TSGs. The GUI state is an abstract representation of the GUI of the tested application, and in FSM-ES, it is composed of a state number, the GUI elements it contains, and the semantics formed by these elements. In short, the semantics of GUI states and events refer to a textual representation (entity) of unique states and events. Furthermore, the connection between GUI states and events is established at the functional meaning level through entity-to-entity relationships, rather than just simple state transition relationships.

Definition 9: FSM-ES is a quaternion, $FSM\_ES = \{S, \Sigma, \delta, s_0\}$, where:

- S is an infinite nonempty set of states of the GUI, consisting of all possible states of the AUT;
- $\sum$ is the finite nonempty set of all possible input events of the AUT;
- δ is the state transfer function that maps $S \times \Sigma$ to S, $\delta : S \times \Sigma \to S$. For $\forall s \in S$, $e \in \Sigma$ and $\delta(s,e)$ indicate the states that can be reached from GUI state s, along the edge labeled e;
- s_0 is the initial GUI state, $s_0 \in S$.

Definitions 10 to 15 are supplementary definitions to FSM-ES.

Definition 10: For $\forall s \in S$, GUI state s is a triplet, $s = \{AC, CC, s^s\}$, where AC is the set of atomic elements, which are non-divisible elements in the GUI. CC is the set of composite elements, which are obtained by combining the atomic elements according to Rule 1.

Definition 11: For $\forall as \in AC$, atomic element ac is a quadruplet, $ac = \{ac^t, ac^v, ac^s, ac^p\}$, where:

- ac^t indicates the type of ac. The types are text, icon, image, and input box. The semantic type is determined by the type of GUI element;
- ac^v indicates the value of ac. The elements whose ac^t is text have this attribute, and ac^v is the value of the text;
- ac^s is the semantics of ac, which is the mapping of ac^v to the corresponding entity c_e in DAOM, $ac^s : ac^v \to \{c_e \mid c_e \in c_E\}$;
- ac^p is the position of ac, which is represented by the upper-left and lower-right coordinates of the element, $ac^p = \left(\left(x_{min}^{ac}, y_{min}^{ac}\right), \left(x_{max}^{ac}, y_{max}^{ac}\right)\right)$.

Rule 1: If an atomic element of a text type is a description of other atomic elements, then the combination is performed.

Figure 5 lists the composite elements that comply with this rule. In this case, the text is a description of an icon, a radio box, a checkbox, and a switch.

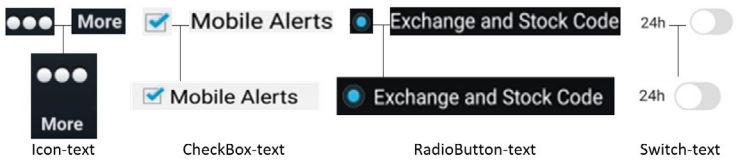

Figure 5. Composite elements

Definition 12: For $\forall s \in S$, $s = \{AC, CC, s^s\}$, $cc \in CC$, composite element cc is a triplet, $cc = \{cc^{AC}, cc^s, cc^p\}$, where:

- cc^{AC} is the set of atomic elements, $cc^{AC} \subseteq AC$;
- cc^S is the semantics of cc, which is the semantics of text type elements in cc^{AC}, that is $cc^s = \{ac^s \mid ac \in cc^{AC} \wedge ac^t = \text{text}\}$;
- cc^P is the position of cc, which is represented by the upper-left and lower-right coordinates of the composite element, $cc^p = \left(\left(x^{ac}_{min}, y^{ac}_{min}\right), \left(x^{ac}_{max}, y^{ac}_{max}\right)\right)$.

Definition 13: For $\forall cc \in CC$, if $ac \in cc^{AC}$, then $ac^s = cc^s$.

Definition 14: S^s is the semantics of GUI states, $s^s = \{cc^s, ac^s \mid cc \in CC \wedge ac \in AC \wedge ac \notin cc^{AC}\}$, which is the set of elements in GUI state s mapped to the entities in DAOM.

Definition 15: For $\forall e \in \Sigma$, event e is a quadruple, $e = \{e^c, e^a, e^p, e^s\}$, where:

- e^c is the element that the e needs to trigger;
- e^a is the trigger operation of e^c, such as a click;
- e^p is the parameter of e;
- e^s is the semantics of e, which defines the functional operations supporting the state migration mapped to the action e_a in DAOM, $c_a \in c_A$, $c_a = \{s^s_k, Op, ac^s\}$, $e^s = \{c_a \mid e^c = ac, s^s_k \subseteq s^s\}$, and s^s is the semantics of current GUI state s.

3.5 Task Sub-Graph (TSG)

To automate the generation of function-related test cases from FSM-ES, the AFG must be mapped to the FSM-ES. The mapping result is called a TSG. Each AFG built according to the functional requirements has a unique corresponding one TSG in the AUT.

Definition 16: Let the FSM-ES $G = \{S, \Sigma, \delta, s_0\}$ and $G' = \{S', \Sigma', \delta', s'_I\}$. If $S' \subseteq S$, $\Sigma' \subseteq \Sigma$, and $\delta' \subseteq \delta$, then G' is called a sub-graph of G and is represented as $G' \subseteq G$.

Definition 17: Let the FSM-ES $G = \{S, \Sigma, \delta, s_0\}$, $G_p = \{S_p, \Sigma_p, \delta_p, s_p\}$, $G_q = \{S_q, \Sigma_q, \delta_q, s_q\}$, and $G_p \subseteq G$, $G_q \subseteq G$.

- If $S_p \subseteq S_q$, then G_p is smaller than G_q, or G_q is larger than G_p and is written as $G_p < G_q$ or $G_q > G_p$;
- If $S_p = S_q \wedge \Sigma_p \subseteq \Sigma_q$, then G_p is smaller than G_q, or G_q is larger than G_p and is written as $G_p < G_q$ or $G_q > G_p$;
- If $S_p = S_q \wedge \Sigma_p = \Sigma_q \wedge \delta_p \subseteq \delta_q$, then G_p is smaller than G_q, or G_q is larger than G_p and is written as $G_p < G_q$ or $G_q > G_p$.

Definition 18: Let the FSM-ES $G = \{S, \Sigma, \delta, s_0\}$, $G' = \{S', \Sigma', \delta', s'_I\}$, $G' \subseteq G$, task $\tau \in c_T$, $\tau = \{N, s^s_I, D, s^s_F\}$, and the AFG $D = \{A, T, F, G, \alpha_I, \alpha_F\}$. If for $\forall a \in A$, where $a = \{s^s_k, Op, ac^s\}$, there is $s_q \in S'$ such that $s^s_p \subseteq s^s_q$, and G' is the smallest sub-graph satisfying the above conditions, then G' is called the TSG of G with respect to task τ, represented as $G'(\tau) \subseteq G$.

Figure 6 shows the TSG of the task named login. $s_i \in S$ and $s_f \in S$ can be found in the FSM-ES $G = \{S, \Sigma, \delta, s_0\}$, satisfying $s^s_I \subseteq s^s_i$, $s^s_1 \subseteq s^s_i$, $s^s_2 \subseteq s^s_q$, $s^s_3 \subseteq s^s_i$, and $s^s_F \subseteq s^s_f$. The sub-graph G' marked in red is the TSG of G with respect to the Login task and is represented as $G'(Login) \subseteq G$. AFG can divide FSM-ES into functional submodels TSG. In addition, since AFG defines feasible actions for functionality, it can optimize state transition relationships, support more accurate generation of test cases, and reduce redundancy in test cases.

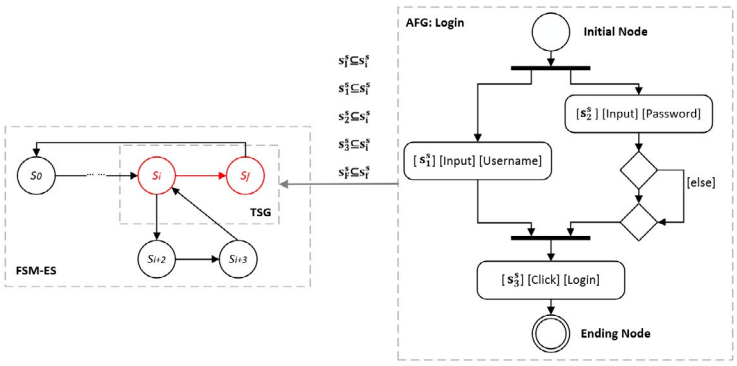

Figure 6. Mapping from the AFG of the login function to the FSM-ES results in TSG

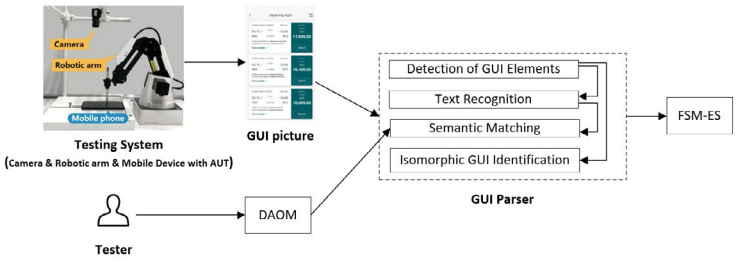

Figure 7. Modelling workflow

4. Automatic Construction of FSM-ES

Instead of manually creating a model, which is a common way to model the AUT, the proposed modelling approach heuristically constructs the FSM-ES using a robot exploring the mobile application. The modelling workflow is shown in Figure 7. The robot performs a series of processes on the images after capturing the current GUI of the AUT. First, GUI element detection is performed using the trained deep learning model. Then, text recognition is performed using the open-source API (Application Programming Interface). Finally, semantic matching is completed by referring to the DAOM manually created by the tester. To prevent node explosion in the model, it is necessary to complete isomorphic GUI recognition based on GUI element detection, and the above-mentioned steps are performed only once for isomorphic GUIs. The above-mentioned process is repeated until all GUIs have been explored, and the corresponding FSM-ES for the AUT is constructed. A three-axis robotic arm and an industrial high-definition camera are used. The robotic arm can simulate typical single finger actions such as clicking, double clicking, long pressing, sliding, and brushing by setting different position intervals and pause times. The camera is used to locate the interactive object elements on the GUI, thereby guiding the interaction between the robotic arm and the AUT. Therefore, it presents a non-invasive, platform independent exploration approach that supports generalized testing modelling.

The robot explores the AUT using a depth-first strategy to drive the robotic arm to operate the GUI elements and complete the state migration. Accordingly, exploring the next state does not require all GUI elements in the current state to be operated. The algorithm framework is shown in Algorithm 1. The robot first captures the current GUI picture (Line 2) and then determines whether it is isomorphic to the existing state in the model (Line 3). If the picture is isomorphic, then the robot updates the model edge set and state transition form (Lines 4 to 5). Otherwise, the robot processes the GUI to obtain modelling information and update the node set, edge set and state transition form (Lines 7 to 10). Thereafter, the robot will be directed to operate on the next GUI element (Line 12). The details of the techniques are as follows.

Algorithm 1. Build GUI Model

Input: Domain application ontology model D
Output: FSM-ES M of AUT, $M = \{S, \Sigma, \delta, s_0\}$
1: **while** not finish exploring **do**
2: capture the current state of the GUI picture
3: **if** s_{cur} is isomorphic to s_{pre} in S **then**
4: $s_{cur} = s_{pre}$
5: update Σ, δ
6: **else**
7: detect GUI elements in s_{cur}
8: recognize texts in s_{cur}
9: semantic match with D
10: update S, Σ, δ
11: **end if**
12: operate the next element in s_{cur}
13: **end while**

4.1 GUI Element Detection

The robot needs to obtain the type and position information of the elements from the GUI pictures captured with the camera. Currently, the deep learning-based technology has shown significant progress, especially in computer vision (Shi et al., 2020; Li et al., 2023), and deep learning-based GUI element detection methods have proven to be an effective solution (Xue, Wu & Zhang, 2022). Deep learning-based detection methods can be divided into one-stage and two-stage detectors. Detection speed is important in the present work. Accordingly, the one-stage detection method, YOLOv5 (Jocher, 2020) is employed for

real-time GUI element recognition. YOLOv5 extracts graphical features through convolutional neural networks and returns the categories and location information of the detection target. Compared to two-stage detection, YOLOv5 has a faster detection efficiency, while maintaining high accuracy, making it suitable for real-time interaction between robots and AUTs.

4.2 Text Recognition

The content of text elements is important for the acquisition of semantics. The text recognition service provided by Baidu is suitable for multi-scene, multi-lingual, and high-precision whole-image text detection and recognition tasks and is used for text recognition in the proposed modelling approach. Baidu provides OCR (Optical Character Recognition) methods that can recognize text on GUI interfaces and provide the position of characters in the interface. More details can be found in the case study section.

4.3 Semantic Matching

The recognized text is matched with entities in the DAOM to determine the semantics of the GUI elements. Currently, different methods focusing on semantic matching are used for three patterns of text. Methods such as distance similarity and cosine similarity are used to determine the semantic similarity of the text.

- For text with a fixed format, such as emails and phone numbers, regular expressions are used to obtain semantics;
- For text containing synonyms, such as buying a ticket and ordering a ticket, semantic matching is done by the semantic similarity calculation service provided by Baidu. Similarity is a real value between zero and one. The higher the output value is, the higher the relative semantic similarity becomes;
- For named entities, such as names of people and places, semantic matching is performed by named entity recognition technology provided by Baidu.

4.4 Isomorphic GUI identification

Isomorphic GUIs are GUIs that differ in appearance (text, image, colour, and size), but are identical in function, structure, and internal logic relationships. Figure 8 shows an example of isomorphic GUIs. Figures 8(a)–8(c) show

Figure 8. Example of isomorphic GUIs: (a)–(c) detailed interface of the information of three flights in the list; (d) interface of the flight information list

the detailed interface of the information of three flights in the list, and they differ only in terms of text. Figure 8(d) shows the interface of the flight information list.

Figure 9(a) shows an example of the corresponding model in Figure 8, where S_{2a}, S_{2b}, and S_{2c} can be regarded as the same state, and the simplified GUI model is shown in Figure 9(b). Hence, the robot can avoid the state explosion problem by identifying isomorphic GUIs. This work completes the experiment by using the isomorphic GUI recognition method proposed by Zhang et al. (2020). First, a GUI skeleton is built to exclude noise from the GUI style characters based on the result of GUI element detection. An autoencoder is used to extract the feature vectors from the reconstructed GUI skeleton. Finally, relative entropy is used in the identification of isomorphic GUIs. Isomorphic GUI identification can achieve simplification of the state model, avoiding redundant paths and repeated jumps in local paths.

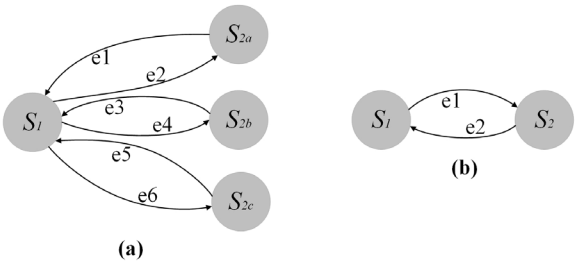

Figure 9. Simplified example of the GUI model by FSM: (a) original GUI model; (b) simplified version

5. Case Study

In this section, a case study of constructing the FSM-ES for a commercial mobile application is described in order to demonstrate how the modelling approach proposed in this paper works in practice, and its effectiveness is analyzed. The important issues identified during the case study are also discussed.

5.1 Creation of the DAOM

Given that DAOM is domain-related, a specific domain should be selected for the case study. A DAOM of the airline service mobile applications is chosen to be built, because mobile applications in airline services are rapidly spreading worldwide.

First, the DAOM is manually built to construct the FSM-ES. Figure 10 shows the entities, actions, and a typical task in the DAOM and their relationships. For the sake of space, only the task of searching flights is taken into consideration.

Figure 10 shows 21 common entities, including user, username, password, login information, email address, order, flight number, and flight duration. Some of these entities may be associated with each other, such as user and order. Meanwhile, some entities may contain other entities as their components, such as name containing first name and last name and order containing passenger, cabin class, and price.

The action flow diagram corresponding to the task of searching flights in DAOM contains four actions. Each action contains the semantics of the GUI state, the action type, and the semantics of the triggered GUI element. The task is first performed by selecting the departure (a_1), the destination (a_2), and the date (a_3) and then clicking the search button (a_4), where the entity departure, destination, and date are the inputs for actions a_1, a_2, and a_3, respectively.

5.2 Construction of the FSM-ES

Cathay Pacific airline services are used as example to explain in detail the steps of constructing FSM-ES using a robot. The application of the modelling approach proposed in this paper allows to demonstrate the practicality and effectiveness of the present approach, because Cathay Pacific is a

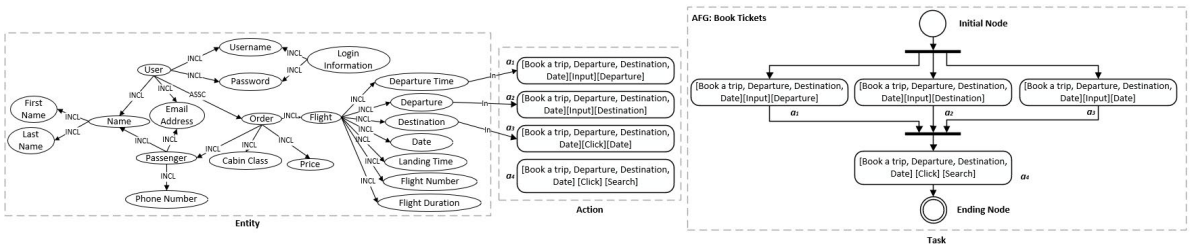

Figure 10. DAOM for airline service mobile applications

commercial application with more than 1 million downloads on Google Play. In this case study, the DOBOT Magician robotic arm and Hikvision industrial camera with model MV-CS200-10GC are used.

The steps of constructing FSM-ES are shown in Figure 11. The first step shows the GUI element detection. Figure 11(b) shows the result of the GUI element detection for the original GUI shown in Figure 11(a). The second step explains the text recognition. Figure 11(c) shows the result of the text recognition for the original GUI. The third step states the semantic matching. Figure 11(d) illustrates the semantics of the identified text for Figure 11(c). To prevent GUI state explosion, isomorphic GUI states are recognized and merged. Figure 11(e) shows the GUI skeleton required for isomorphic GUI recognition. The details of the above-mentioned steps will be discussed below.

5.2.1 GUI Element Detection

The GUI element detection is performed by using the YOLOv5 network. To this end, the network needs to be trained to become capable of fulfilling the task. 1901 labeled GUI images taken from Rico (Deka et al., 2017) were used to train the network and apply it to the Cathay Pacific application. The 36 GUI states in Cathay Pacific has a total of 840 elements, and the trained network identified a total of 694 elements, where 11 elements were incorrectly identified and 146 elements were missed, with an accuracy rate of 81.31%.

The robot detects the GUI elements in Figure 12(a) using the trained YOLOv5 network, and the detection results are shown in Figure 12(b), corresponding to the TXT file shown in Figure 12(c), which contains the coordinates and categories of the GUI elements. According to the result of GUI element detection, the set of states

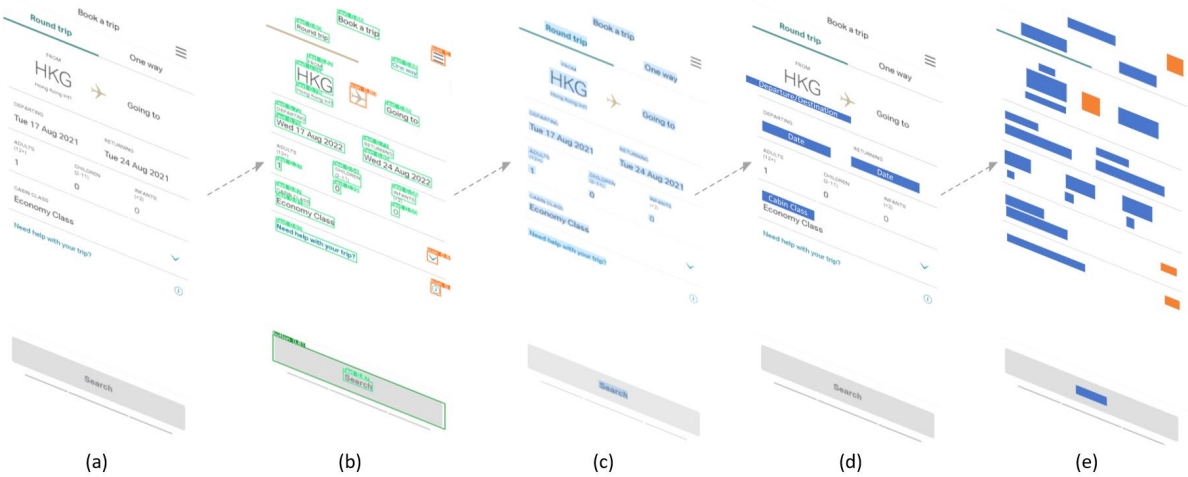

Figure 11. FSM-ES construction process: (a) GUI; (b) GUI element detection; (c) text recognition; (d) semantic matching; (e) GUI skeleton

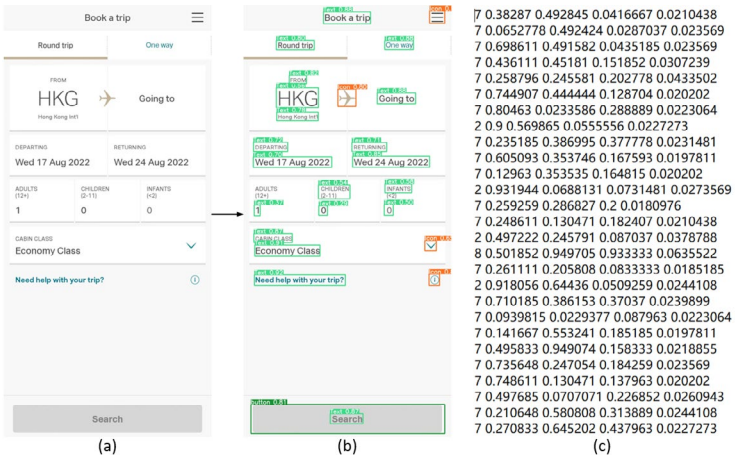

Figure 12. Result of GUI element detection: (a) original GUI; (b) detection result; (c) result details

in the FSM-ES is updated, and the robot can be driven to the next step.

5.2.2 Text Recognition

This case study introduces the OCR technology to recognize a text because the elements of text type need to obtain specific element values. Taking Cathay Pacific as example, the accuracy of text recognition based on the result of GUI element detection is compared with the accuracy of text recognition directly on the image. The former is 79.06%, and the latter is 92.29%. The former is lower due to the dependence on the accuracy of recognition and classification in GUI element detection, so direct performance on the image is taken for text recognition. After the OCR is used, the GUI element detection accuracy increased to 90.71%.

The robot uses OCR to directly perform text recognition on the GUI shown in Figure 12(a) and obtains the text recognition result, as shown in Figure 13, which contains the values of the text elements in the GUI and their coordinates. After the text recognition, the robot updates the state set in the FSM-ES. Performing text recognition directly on the GUI can also prevent text elements from being missed, thus improving the accuracy of constructing the model.

The dynamic GUIs and element rich GUIs in AUT can lead to a decrease in the accuracy of visual recognition of GUI. Further visual recognition training for apps in segmented fields may improve this issue.

5.2.3 Semantic Matching

The essential way to obtain the semantics of the recognized entities from the GUI is first to recognize the relevant texts and then try to match them with the similar texts stored in the DAOM. The semantics of the recognized texts from the GUI can be found, because the text stored in the DAOM is associated with a clear semantics. In the present case study, three patterns of texts, namely texts with a fixed format, texts containing synonyms in the DAOM and the named entities, are taken into consideration.

For fixed format text, regular expressions are used for text type detection, such as the date concept expressed by year, month, and day. For the proprietary names that DAOM may contain, direct matching is chosen, such as place names. In terms of named entity recognition, it is obtained

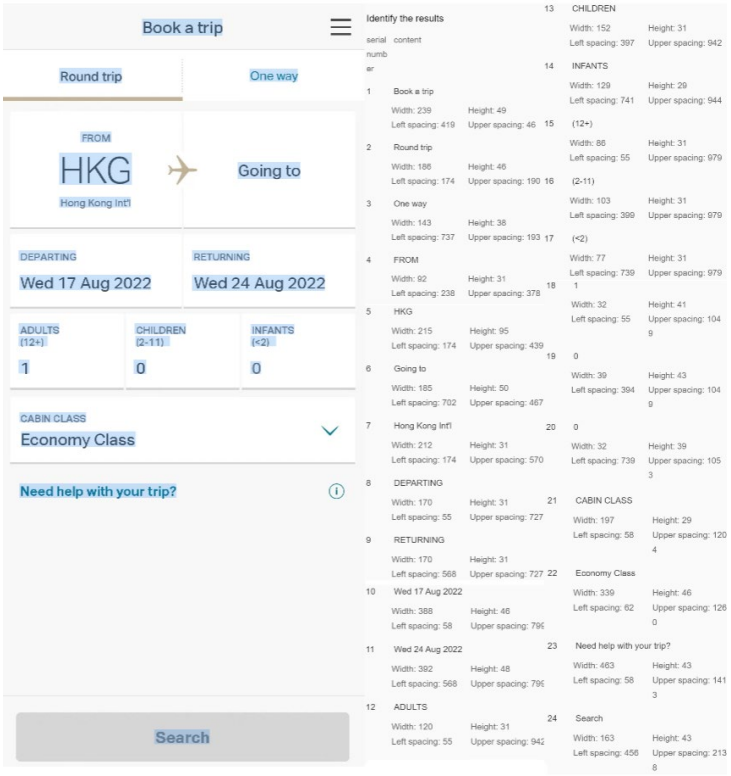

Figure 13. Result of text recognition

by converting entity concepts into vectors using an encoder and measuring the similarity of the vectors. In Cathay Pacific, a total of 85 element semantics were identified. The results of text recognition shown in Figure 13 are used for semantic acquisition, where Hong Kong is analyzed as a place name, which can be associated to departure or destination in DAOM entity. Wed 17 Aug 2022 and Wed 24 Aug 2022 are analyzed as dates, which can be associated to the dates in the DAOM entity. Meanwhile, cabin class can directly correspond to cabin class in the DAOM entity.

5.2.4 Isomorphic GUI Recognition

This case study merges nodes by identifying isomorphic GUIs to prevent GUI state explosion. Figures 14(a) and 14(c) show examples of isomorphic GUIs that appear in Cathay Pacific. The robot uses the corresponding GUI element detection results to obtain the GUI skeletons as shown in Figures 14(b) and 14(d) to exclude noise from UI style characters. An auto-encoder is then used to extract feature vectors from the reconstructed GUI skeleton. Finally, the relative entropy was used to identify isomorphic GUIs. The two interfaces in Figure 14 were identified as isomorphic. If the GUI state is the first occurrence, then no interface is isomorphic to it, and the number of states in the model increases; otherwise, the node is merged into an existing node, and the number of states in the model remains the same.

After the above-mentioned procedure is performed according to Algorithm 1, the corresponding FSM-ES of Cathay Pacific can be obtained, as shown in Figure 15.

Simultaneously with automated modelling, senior master's students specializing in software engineering are engaged to perform manual modelling. After multiple modelling rounds to achieve core functional coverage of Cathay Pacific, manual modelling requires an average of 122 minutes, whereas the robot completes the modelling in an average of 58 minutes. Automated modelling is 52% faster than manual modelling. However, besides covering the core functions, manual modelling has extended coverage to additional application states. This discrepancy arises from the richer semantic concepts possessed by humans, enabling the unlocking of additional application states.

After exploration by the robot, the FSM-ES with 36 nodes and 78 edges of the Cathay Pacific application is formed, as shown in Figure 16. Table 1 shows the keywords in the GUI corresponding to each node.

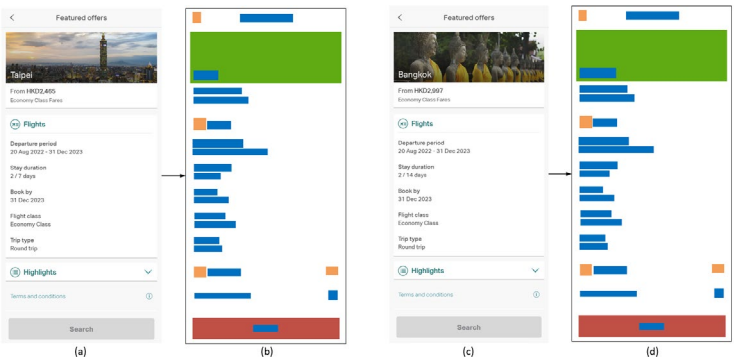

Figure 14. Isomorphic GUI recognition: (a) isomorphic GUI; (b) corresponding GUI skeleton for (a); (c) isomorphic GUI; (d) corresponding GUI skeleton for (c)

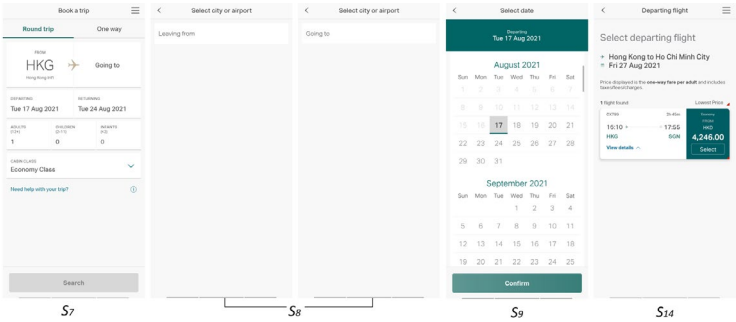

Figure 15. GUI states of Cathay Pacific about searching flights

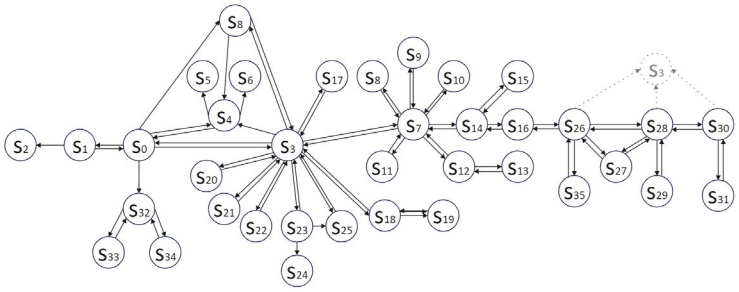

Figure 16. FSM-ES of Cathay Pacific

Table 1. Corresponding GUI for each node

Node	GUI	Node	GUI	Node	GUI
s_0	Home	s_1	Message Centre	s_2	Travel alerts
s_3	List Items	s_4	Sign in	s_5	Sign in with account
s_6	Learn more	s_7	Book a trip	s_8	Select city or airport
s_9	Select date	s_{10}	Select other information	s_{11}	Travel Help
s_{12}	Travel Restriction	s_{13}	See other sections	s_{14}	Departing flight
s_{15}	Flight Sort & Filter	s_{16}	Select fare	s_{17}	Mobile boarding passes
s_{18}	Flight status	s_{19}	Flight results	s_{20}	Timetable

5.3 Generate Test Cases

In this subsection, first the TSG that illustrates the search flight function in the Cathay Pacific application is described, and then the test cases generated based on the TSG in order to check the search flight function are explained.

Figure 16 shows the states of the GUI for searching flights in the Cathay Pacific application. Five screens are associated with the task of searching for flights. The GUI pages (or interfaces) shown in the diagram are Home, Select Departure, Select Destination, Select Date, and Flight List. The Select Departure and Select Destination interfaces are merged into the same node s_8 through isomorphic GUI recognition, and the other remaining interfaces are represented by nodes s_7, s_9, and s_{14}.

Figure 17 shows the TSG of Cathay Pacific regarding the searching flights task. This TSG involves a total of four states and eight events. Taking s_7 as example, the corresponding GUI semantics for this node is [Book a trip, Round trip, One way, Departure, Destination, Date, ADULTS, Children, INFANTS, (12+), (2-11), (<2), 1, 0, 0, Cabin Class, Need help with your trip, Search]. Taking e_1 as example, the event semantics for this edge is [Book a trip, Departure, Destination, Date][Input][Departure] corresponding to action a_1 shown in Figure10.

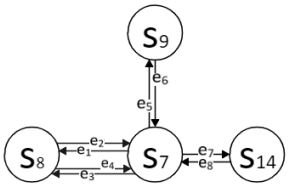

Figure 17. TSG corresponding to the task of searching the flights in Cathay Pacific

In this case study, test sequences are used to represent test cases. The test sequence generated from the TSG corresponding to the searching the flight task is shown in Table 2. Each test sequence corresponds to a test case related to the searching flight function.

Table 2. Test sequence generated from the TSG

No.	Test Sequence
1	$s_7 \xrightarrow{e_1} s_8 \xrightarrow{e_2} s_7 \xrightarrow{e_3} s_8 \xrightarrow{e_4} s_7 \xrightarrow{e_5} s_9 \xrightarrow{e_6} s_7 \xrightarrow{e_7} s_{14}$
2	$s_7 \xrightarrow{e_3} s_8 \xrightarrow{e_4} s_7 \xrightarrow{e_1} s_8 \xrightarrow{e_2} s_7 \xrightarrow{e_5} s_9 \xrightarrow{e_6} s_7 \xrightarrow{e_7} s_{14}$
3	$s_7 \xrightarrow{e_1} s_8 \xrightarrow{e_2} s_7 \xrightarrow{e_5} s_9 \xrightarrow{e_6} s_7 \xrightarrow{e_3} s_8 \xrightarrow{e_4} s_7 \xrightarrow{e_7} s_{14}$
4	$s_7 \xrightarrow{e_3} s_8 \xrightarrow{e_4} s_7 \xrightarrow{e_5} s_9 \xrightarrow{e_6} s_7 \xrightarrow{e_1} s_8 \xrightarrow{e_2} s_7 \xrightarrow{e_7} s_{14}$
5	$s_7 \xrightarrow{e_5} s_9 \xrightarrow{e_6} s_7 \xrightarrow{e_1} s_8 \xrightarrow{e_2} s_7 \xrightarrow{e_3} s_8 \xrightarrow{e_4} s_7 \xrightarrow{e_7} s_{14}$
6	$s_7 \xrightarrow{e_5} s_9 \xrightarrow{e_6} s_7 \xrightarrow{e_3} s_8 \xrightarrow{e_4} s_7 \xrightarrow{e_1} s_8 \xrightarrow{e_2} s_7 \xrightarrow{e_7} s_{14}$

Take No. 1 test sequence as example, the robot first clicks on HKG at s_7 to jump to s_8 to select the departure. Then, the robot gets the instance of departure in DAOM, enters it, and returns to s_7. Subsequently, it clicks on Going to at s_7 to jump to s_8 to select the destination. After that, the robot gets the instance of destination in DAOM, enters it, and returns to s_7. In s_7, it clicks on Wed 17 Aug 2022 to jump to s_9 to select the date. In DAOM, it gets the instance of date, selects it, and then returns to s_7. Finally, in s_7, the robot clicks on Search to jump to s_{14}.

5.4 Discussion

The case study allowed the gaining of experience in applying the present modelling and identifying two important aspects. This subsection discusses these issues to provide a full picture of the present approach.

5.4.1 Limitations on Equipment

The proposed modelling approach allows to build the FSM-ES for all mobile applications deployed on touchscreen devices. The only restrictions on the device are that the device must have a touch screen, and the test environment needs to have a programmable robot that can be used for testing. The method is independent of the underlying operating system of the device and theoretically supports compatibility testing. Mobile applications have adaptability and display differences at different resolutions and sizes. For example, a line of text on a large-sized screen may appear as multiple lines on a small-sized screen.

5.4.2 Human Intervention

Although the approach proposed in this paper aims to realize automatic construction of semantic model FSM-ES of mobile applications, the case study helped to find the necessity of involving human intervention during the modelling process using a robot. The main reasons are described as follows. First, exceptions may occur in the GUI during the modelling process. Given that this mechanism is still in the model construction process, there is no valid reference information, so the error cannot be detected, and the robot will only continue to generate the wrong model based on the current GUI state. Second, the error detection of the GUI elements cannot be avoided due to the limitation of the current vision-based GUI element recognition accuracy, and the error detection of the GUI elements may generate a wrong model. Hence, Testers need to intervene in key steps during the modelling and testing process to complete it. However, establishing a comprehensive exception handling mechanism will further promote the level of automation. Moreover, utilizing human-in-the-loop approaches and using robots as testing aids will still promote improved testing efficiency.

6. Conclusion

This paper proposes FSM-ES, a semantic GUI model automatically built by robot. FSM-ES enables efficient test case generation with reduced redundancy, focusing tests on functionality. In addition, the combination of visual technology and robotics was researched to achieve code-independent and platform-independent GUI modelling. Compared to manual modelling, the automated construction of FSM-ES is 52% faster.

However, optimizing the reduction of human intervention remains a worthwhile future direction, with a focus on enhancing the accuracy of visual algorithms and refining modelling execution strategies to augment automation. Additionally, the testing method relying on robot vision holds significant potential value for cross-platform testing and migration testing of applications.

REFERENCES

Abulhaija, S., Hattab, S., Abdeen, A. & Etaiwi, W. (2022) Predicting mobile apps performance using machine learning. *Journal of System and Management Sciences*. 12(6), 300-314. doi: 10.33168/JSMS.2022.0619.

Bera, D., Schuts, M., Hooman, J. & Kurtev, I. (2021) Reverse engineering models of software interfaces. *Computer Science and Information Systems*. 18(3), 657-686. doi: 10.2298/CSIS200131013B.

Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., Nichols, J. & Kumar, R. (2017) Rico: A mobile app dataset for building data-driven design applications. In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software*

and Technology, UIST 2017, 22-25 October 2017, Québec City, Canada. New York, NY, United States, Association for Computing Machinery. pp. 845-854.

Gelfert, A. (2017) The ontology of models. In: Magnani, L. & Bertolotti, T. (ed.) *Springer Handbook of Model-Based Science*. Cham, Springer Handbooks, Springer, pp. 5-23. doi: 10.1007/978-3-319-30526-4_1.

He, Z., Li, J., Wu F., Shi H. & Hwang, K. (2023) DeRL: Coupling Decomposition in Action Space for Reinforcement Learning Task. *IEEE Transactions on Emerging Topics in Computational Intelligence*. 8(1), 1030-1043. doi: 10.1109/TETCI.2023.3326551.

Huang, Y. & Lu, L. (2012) Apply ant colony to event-flow model for graphical user interface test case generation. *IET Software*. 6(1), 50-60. doi: 10.1049/iet-sen.2011.0012.

Imparato, G. (2015) A combined technique of GUI ripping and input perturbation testing for Android apps. In: *Proceedings of the IEEE/ACM 37th IEEE International Conference on Software Engineering, ICSE 2015, 16-24 May 2015, Florence, Italy*. Washington, DC, United States, IEEE Computer Society. pp. 760-762.

Jocher, G. (2020) *YOLOv5, code repository*. https://github.com/ultralytics/YOLOv5 [Accessed 15th October 2023].

Kong, P., Li, L., Gao, J., Liu, K., Bissyandé, T. F. & Klein, J. (2018) Automated testing of android apps: A systematic literature review. *IEEE Transactions on Reliability*. 68(1), 45-66. doi: 10.1109/TR.2018.2865733.

Kull, A. (2012) Automatic GUI model generation: State of the art. In: *Proceedings of the IEEE 23rd International Symposium on Software Reliability Engineering Workshops, ISSRE Wksp 2012, 27-30 November 2012, Dallas, TX, USA*. Washington, DC, United States, IEEE Computer Society. pp. 207-212.

Lafi, M., Alrawashed, T. & Hammad, A. M. (2021). Automated test cases generation from requirements specification. In: *Proceedings of the 2021 International Conference on Information Technology, ICIT 2021, 14-15 July 2021, Amman, Jordan*. Washington, DC, United States, IEEE Computer Society. pp. 852-857.

Li, J., Shi, H., Chen, W., Liu, N. & Hwang, K. (2023) Semi-Supervised Detection Model Based on Adaptive Ensemble Learning for Medical Images. *IEEE Transactions on Neural Networks and Learning Systems*. 1-12. doi: 10.1109/TNNLS.2023.3282809.

Li, J., Shi H. & Hwang, K. (2023) Using Goal-Conditioned Reinforcement Learning with Deep Imitation to Control Robot Arm in Flexible Flat Cable Assembly Task. *IEEE Transactions on Automation Science and Engineering*. 1-12. doi: 10.1109/TASE.2023.3323307.

Li, T. J. J., Popowski, L., Mitchell, T. & Myers, B. A. (2021) Screen2vec: Semantic embedding of GUI screens and GUI components. In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, CHI 2021, 8-13 May 2021, Yokohama, Japan*. New York, NY, United States, Association for Computing Machinery. pp. 1-15.

Lin, J. W., Jabbarvand, R. & Malek, S. (2019) Test transfer across mobile apps through semantic mapping. In: *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, 10-15 November 2019, San Diego, CA, USA*. New Jersey, USA, IEEE Press. pp. 42-53.

Liu, T. F., Craft, M., Situ, J., Yumer, E., Mech, R. & Kumar, R. (2018) Learning design semantics for mobile apps. In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology, UIST 2018, 11-14 October 2018, Berlin, Germany*. New York, NY, United States, Association for Computing Machinery. pp. 569-579.

Mandić, M. (2022) Semantic web based platform for the harmonization of teacher education curricula. *Computer Science and Information Systems.* 19(1), 229-250. doi: 10.2298/CSIS210207050M.

Mao, K., Harman, M. & Jia, Y. (2017) Robotic testing of mobile apps for truly black-box automation. *IEEE Software*. 34(2), 11-16. doi: 10.1109/MS.2017.49.

Marchetto, A., Tonella, P. & Ricca, F. (2008) State-based testing of Ajax web applications. In: *Proceedings of the 1st International Conference on Software Testing, Verification, and Validation, ICST 2008, 9-11 April 2008, Lillehammer, Norway*. Washington, DC, United States, IEEE Computer Society. pp. 121-130.

Miao, Y. & Yang, X. (2010) An FSM based GUI test automation model. In: *Proceedings of the 11th International Conference on Control Automation Robotics & Vision, ICARCV 2010, 7-10 December 2010, Singapore*. Washington, DC, United States, IEEE Computer Society. pp. 120-126. doi: 10.1109/ICARCV.2010.5707766.

Naing, M. M., Lim, E. P. & Goh, D. H. L. (2003) A survey of ontology-based web annotation. In: *Proceedings of the 1st International Conference on Computer Applications, ICCA 2003, 15-16 January 2003, Yangon, Myanmar*. Singapore Management University. pp. 113-123.

Nguyen, B. N., Robbins, B., Banerjee, I. & Memon, A. (2014) GUITAR: an innovative tool for automated testing of GUI-driven software. *Automated Software Engineering*. 21, 65-105. doi: 10.1007/s10515-013-0128-9.

Pan, Z., Chen, J., Yao, L. & Chen, Z. (2020) Research on functional test of mobile app based on robot. In: *Proceedings of the IEEE 5th International Conference on Signal and Image Processing, ICSIP 2020, 23-*

25 October 2020, Nanjing, China. Washington, DC, United States, IEEE Computer Society. pp. 960-964.

Qian, J., Shang, Z., Yan, S., Wang, Y. & Chen, L. (2020) RoScript: a visual script driven truly non-intrusive robotic testing system for touch screen applications. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE 2020, 05-11 October 2020, Seoul, South Korea*. New York, NY, United States, Association for Computing Machinery. pp. 297-308.

Reis, J. & Mota, A. (2018) Aiding exploratory testing with pruned GUI models. *Information Processing Letters*. 133, 49-55. doi: 10.1016/j.ipl.2018.01.008.

Salihu, I. A., Ibrahim, R., Ahmed, B. S., Zamli, K. Z., & Usman, A. (2019) AMOGA: A static-dynamic model generation strategy for mobile apps testing. *IEEE Access*. 7, 17158-17173. doi: 10.1109/ACCESS.2019.2895504.

Schulze, C., Lindvall, M., Bjorgvinsson, S. & Wiegand, R. (2015) Model generation to support model-based testing applied on the NASA DAT Web-application - An experience report. In: *Proceedings of the IEEE 26th International Symposium on Software Reliability Engineering, ISSRE 2015, 2-5 November 2015, Gaithersbury, MD, USA*. Washington, DC, United States, IEEE Computer Society. pp. 77-87.

Shi, H., Li, J., Mao J. & Hwang K. (2023) Lateral Transfer Learning for Multiagent Reinforcement Learning. *IEEE Transactions on Cybernetics*. 53(3), 1699-1711. doi: 10.1109/TCYB.2021.3108237.

Shi, H., Wu, H., Xu, C., Zhu, J., Hwang, M. & Hwang, K. S. (2020). Adaptive image-based visual servoing using reinforcement learning with fuzzy state coding. *IEEE Transactions on Fuzzy Systems*. 28(12), 3244-3255. doi: 10.1109/TFUZZ.2020.2991147

Song, Y. (2021) Construction of Event Knowledge Graph based on Semantic Analysis. *Tehnički Vjesnik [Technical Gazette]*. 28(5), 1640-1646. doi: 10.17559/TV-20210427063132.

Tan, Z. Y. J., Hasa, M. M., Wong, M. Y. & Ramasamy, R. K. (2022) Implementation approach of unit and integration testing method based on recent advancements in functional software testing. *Journal of System and Management Sciences*. 12(4), 85-100. doi: 10.33168/JSMS.2022.0406.

Xu, H., Li, P., Cong, Z., Zhang, F., Pan, Y., Ren, X., & Xing, Y. (2021) Test case prioritization based on artificial immune algorithm. *Tehnički Vjesnik [Technical Gazette]*. 28(6), 1871-1876. doi: 10.17559/TV-20210311060442.

Xue, F., Wu, J. & Zhang, T. (2022) Visual Identification of Mobile App GUI Elements for Automated Robotic Testing. *Computational Intelligence and Neuroscience*. 2022, 4471455. doi: 10.1155/2022/4471455.

Zhang, T., Liu, Y., Gao, J., Gao, L. P. & Cheng, J. (2020) Deep learning-based mobile application isomorphic GUI identification for automated robotic testing. *IEEE Software*. 37(4), 67-74. doi: 10.1109/MS.2020.2987044.