Model-Based Testing of Post-WIMP Interactions Using Object Oriented Petri-nets

Alexandre Canny[1], David Navarre[1], José Creissac Campos[2] and Philippe Palanque[1]

[1] ICS-IRIT, Université Paul Sabatier – Toulouse III, Toulouse, France
[2] HASLab/INESC TEC & Department of Informatics/University of Minho, Portugal
{alexandre.canny,navarre,palanque}@irit.fr, jose.campos@di.uminho.pt

Abstract. Model-Based Testing (MBT) relies on models of a System Under Test (SUT) to derive test cases for said system. While Finite State Machine (FSM), workflow, etc. are widely used to derive test cases for WIMP applications (i.e. applications depending on 2D widgets such as menus and icons), these notations lack the expressive power to describe the interaction techniques and behaviors found in post-WIMP applications. In this paper, we aim at demonstrating that thanks to ICO, a formal notation for describing interactive systems, it is possible to generate test cases that go beyond the state of the art by addressing the MBT of advanced interaction techniques in post-WIMP applications.

Keywords: Post-WIMP Interactive Systems, Software Testing, Model-Based Testing.

1 Introduction

Model-Based Testing (MBT) of software relies on explicit behavior models of a system to derive test cases [30]. The complexity of deriving comprehensive test cases increases with the inner complexity of the System Under Test (SUT) that requires description techniques with an important expressive power. The modelling of post-WIMP (Windows, Icons, Menus and Pointers) interactive applications (i.e. applications with an interface not dependent on classical 2D widgets such as menus and icons [31]) proves to be a challenging activity as pointed out by [13]. For instance, when using a touch screen, each finger down/up is a virtual input device being added or removed from the systems at runtime and behaves in parallel with the other fingers or input devices. A modelling technique able to describe such interactive systems must support the description of dynamicity.

Beyond the problem of describing the SUT behavior, testing Graphical-User Interface, whether it is WIMP or post-WIMP, is known to be a complex activity [10], especially because of the unpredictability of the human behavior as well as the virtually infinite number of possible interaction sequences. To face such difficulty, model-based testing techniques have been developed to try to generate relevant test sequences without relying on manual scripting or capture and replay of tester's interactions.

The massive adoption of touch screens means advanced touch interactions (e.g. swipe, pinch-to-zoom, etc.) gained in popularity, while most of the existing MBT techniques for interactive applications are designed to deal with events performed on the standard GUI widgets (e.g. button, combo box, etc.) [1][10][16][28]. Lelli et al. [16] identified the need for new MBT techniques for post-WIMP applications by highlighting the need for supporting ad-hoc widgets (i.e. non-standard widgets developed specifically for the application) and advanced interaction techniques.

In this paper, we propose to build upon the work of Hamon et al. [13], which used the ICO [21] formal modelling technique to describe post-WIMP interactive systems, as a support to the generation of test cases for interaction techniques of post-WIMP applications, and to demonstrate that testing can be conducted following the standard process for Model-Based Testing proposed in [30]. As interaction techniques have to cope with the high dynamicity of Input/Ouput, as well as temporal aspects, they prove to be one of the most difficult components of interactive systems to be described. Thus, they are the prime focus of this paper, even though we will highlight that our proposed approach applies to other components of the interactive systems' architecture as well.

This paper is structured as follows: Section 2 presents related work on the MBT of interactive applications; Section 3 introduces the interaction technique on which we propose to apply the approach and its modelling in ICO; Section 4 discusses the generation of the test cases from the ICO specification and Section 5 provides some comments on test execution; Section 6 discusses the generalizability of the proposed approach to other components of the SUT; Section 7 concludes the paper by discussing future work.

2 Related Work

The classical approaches to interactive applications testing consider that the user's interaction takes place at the GUI widget level (e.g. buttons, icons, label, etc.). While it is the case in the WIMP paradigm, this assertion cannot be used in the post-WIMP paradigm where "at least one interaction technique is not dependent on classical 2D widgets such as menus and icons" [31]. Consider a gesture-based (post-WIMP) drawing tool. One may want to define (and test) whether moving two fingers on the drawing area means zooming (pinch-to-zoom), rotating or drawing. As this may be determined by how the user effectively moves his/her fingers (speed, angle, pressure level, delay between finger down events, etc.), it goes beyond available standard testing techniques for widget level interactions.

In this section, we first introduce the process of MBT and discuss the existing Model-Based Testing techniques for WIMP applications. We then discuss the testing of post-WIMP applications in order to highlight challenges to overcome.

2.1 The Process of Model-Based Testing

In their Taxonomy of Model-Based-Testing Approaches, Utting et al. [30] present the model-based testing process illustrated by **Fig. 1**. In this process, a model of the SUT is built from informal requirements or existing specification documents (**Fig. 1**.(1)) and test selection criteria (**Fig. 1**.(2)) are chosen to guide the automatic test generation to produce a test suite that fulfils the test policy defined for the SUT. These criteria are then transformed (**Fig. 1**.(3)) into a test cases specification (i.e. a high-level description of a desired test case). Utting et al. [30] use the example of test case specification using state coverage of a finite state machine (FSM). In such case, a set of test case specification {reach s0, reach s1, reach s2…} where s0, s1, s2 are all the states of the FSM is the test case specification.

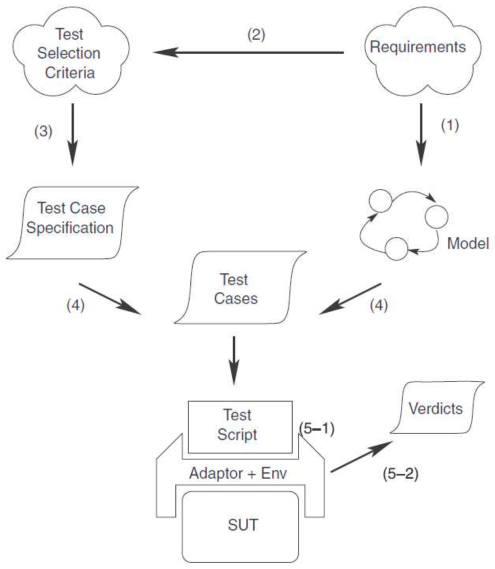

Fig. 1. The process of Model-Based-Testing (from [30])

Once the model and the test case specifications are defined, a set of test cases is generated with the aim of satisfying all the test case specifications (**Fig. 1**.(4)). With the test suite generated, the test cases are executed (either manually -i.e. by a physical person- or automatically thanks to a test execution environment). This requires concretizing the test inputs (**Fig. 1**.(5-1)) and comparing the results against expected ones to produces a verdict (**Fig. 1**.(5-2)).

2.2 Model-Based Testing of WIMP Application

In software engineering, the nearly three-decades-old field [1] that addresses concerns regarding the testing of user interfaces is called "GUI testing". In [1] GUI testing is defined as performing sequences of events (e.g., "click on button", "enter text", "open menu") on GUI widgets (e.g., "button", "text-field", "pull-down menu"). For each sequence, the test oracle checks the correctness of the state of the GUI either after each event or at the end of the sequence. Since the domain is three-decade-old, it naturally focused on WIMP UIs as they were the only available at the time. This focusing is still quite present today.

Some of the research works presented in the following paragraphs do not follow the process of MBT presented by Utting [30], but they propose relevant and inspiring approach for WIMP application testing.

Memon et al. [18] propose a detailed taxonomy of the Model-Based techniques employed to generate test cases in GUI testing. These techniques rely on various kinds of models (state machine, workflow, etc.) that target mono-event-based systems (i.e. systems on which UI events are produced directly as a result of a single action on a widget: key typed, mouse clicked, etc.). They describe the possible test cases by checking reachability of a node. It is important to mention that most of the techniques listed in [18] rely on models built by reverse engineering of the SUT [25].

Another approach based on reverse engineering is the one of Morgado et al. [20] in the iMPAcT tool. This tool uses patterns of common behavior on Android applications to automatically test them. The tool explores the SUT checking for UI patterns using a reverse engineering process. Each UI pattern has a corresponding testing strategy (a Test Pattern) that the tool applies.

Bowen et al. [8] adopt the test-first development approach in which abstract tests are built from formal specification of the system functionality (given using Z [29]) and from a presentation model describing the interactive components (widgets) of the user interface. These abstract test cases are used to produce JUnit and UISpec4J[1] test cases.

Finally, Campos et al. [9] propose an example of approach that matches the outlines of the MBT-process by using task models to perform scenario-based testing of user-interfaces coded in Java using the Synergistic IDE Toucan [17]. The conformance between the application code and the task models is checked at runtime thanks to annotations in the Java code that allow the association of methods calls to the Interactive Input and Output Tasks. The scenarios produced from the task model are then played automatically on the Java application.

2.3 Model-Based Testing of post-WIMP Application

Testing post-WIMP applications requires going beyond GUI testing as mentioned by Lelli et al. [16]. This requires considering ad-hoc widgets and complex interaction techniques that cannot be performed simply as sequences of events on GUI widgets. For instance, interactions such as gesture-based or voice command activations are not tied to a specific GUI widget.

One of the main references in post-WIMP application testing is Malai [16] that has been proposed as a framework to describe advanced GUI Testing. It allows the description of interaction using Finite State Machine (FSM) with two types of end state: terminal state and aborting state. These states are dedicated to identifying whether the user completed the interaction or aborted it. The output actions associated with completing the interaction (i.e. reaching its terminal state) are described in a specific reification of tools called instruments.

However, the use of FSM limits the description of interaction techniques and should be enhanced to support:

[1] https://github.com/UISpec4J/UISpec4J

- **The description of dynamic instantiation of physical and virtual input/output devices**: on systems with a touchscreen, the display is a physical output device and the touch layer the physical input device. When dealing with multi-touch interaction, a finger is a virtual device that is added/removed whenever it touches the screen or is removed from it;
- **The description of timing aspects** to represent quantitative temporal evolution of the interaction technique (available in timed-automata);
- **The description of concurrent aspect** to represent concurrent usage of input devices by the user; events from these devices might be fused to produce higher-level multimodal event [15];
- **The description of dynamic user interface behavior driven by temporal events** such as animations during transition between states of the system [19];
- **The description of system configurations** as, for instance, using resolution scaling on displays with high pixels densities affects the size, location and translation of the GUI elements on screen. Beyond, this also applies to mobile and web-based UI in which having a responsive-design behaving properly is a concern.

While advances have been made in the description of such aspect, especially in work such as [13], there are not, to the best of our knowledge, techniques taking advantages of them to generate tests cases for interactive applications. In the following of the paper, we introduce and use the ICO formalism to demonstrate the need for advanced modelling techniques for effective testing of interactive applications.

3 Modelling of a Post-WIMP Case Study Using ICO

In this section, we present an architecture for post-WIMP applications and highlight where the interaction techniques take place. We then present the informal requirements for the “finger clustering” interaction technique used as a case study in the remaining of this paper. Thereafter, we introduce the formal description technique we use, ICO [21], and present the models associated to the “finger clustering” interaction technique.

3.1 Architecture of a Post-WIMP Application

Effectively testing an interactive application requires a good understanding of its architecture and of the role of its components to select appropriate test criteria [10]. While a detailed architecture such as MIODMIT [11] is able to describe in detail the hardware and software components of interactive systems, we use in this paper a simpler software architecture (inspired by ARCH [4]) for touch applications, presented in **Fig. 2**, to detail the role of the component we focus on. The work presented in the remaining of this paper is still applicable to a more complex architecture.

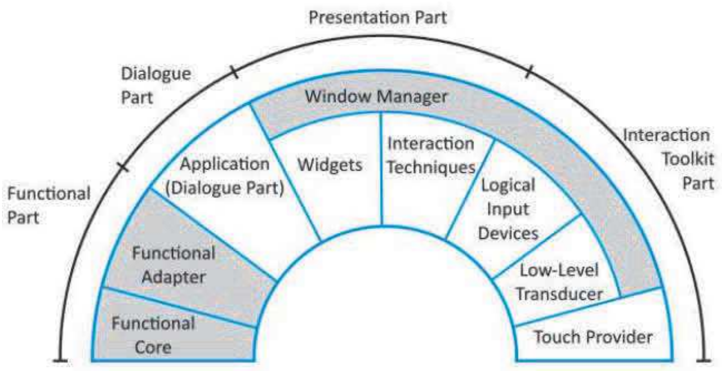

Fig. 2. Example of architecture of a post-WIMP application adapted from [13].

As this paper discusses specific aspects of post-WIMP application, we do not detail the “back-end”, or Functional Part, of the application (leftmost part of **Fig. 2**). The Dialogue Part of the application shares a common role in WIMP and post-WIMP applications, i.e. translating high-level events resulting of the user interaction into invocations on the Functional Part. The main difference between WIMP and post-WIMP applications then resides in the Window Manager that contains, from right to left, the widgets (that share similar roles to widgets of WIMP interfaces), the Interaction Techniques, the Logical Input Device and the Low-Level Transducer.

The Low-Level Transducer is connected to the Touch Provider (rightmost part of **Fig. 2**), i.e. the driver of the touch screen. The Touch Provider produces the lowest-level events in the input chain as they are directly derived from the touch screen behavior. The role of the Low-Level Transducer is to handle these low-level events and to translate them to make sense for the Window Manager logic. On touch applications, the Low-Level Transducer creates Logical Input Devices (i.e. Fingers) with unique IDs and additional information (coordinates, pressure level, etc.). The Logical Input Devices are added to the Window Manager Interaction Technique(s) that will notify widgets and other subscribers (such as a drawing panel) using high-level events when either simple (e.g. tap) or complex (e.g. pinch) interactions are performed.

While this paper focus on the testing of the Interaction Technique, i.e. on verifying that for a set of Logical Input Device actions, the correct high-level events are produced, we highlight the applicability of our methods to the other components of the architecture and on integration testing of these components.

3.2 Presentation of the “Finger Clustering” Interaction Technique

The case study we use in this paper is a multi-touch interaction technique that produces events when fingers are clustered (i.e. within a given range of each other) and de-clustered according to the requirements presented below. These requirements are the inputs for the MBT Process (top-right of **Fig. 1**):

- Clusters may either contain two or three fingers;

- Clusters of three fingers are always created in priority over clusters of two fingers (i.e. if 4 fingers are on the screen in a range suitable for creating a cluster of 3 fingers, a three finger cluster will be created with a finger left alone; in no occasion such circumstance may lead to the creation of two clusters of two fingers);
- The distance between two fingers must be under 100 pixels to create a 2 finger clusters;
- Clusters of three fingers are created when three fingers on the screen form a triangle with each of its edges measuring less than 100 pixels. If it happens that two fingers of an existing cluster of 2 fingers can be part of a three fingers cluster, then the three fingers cluster is created, removing the 2 fingers cluster.
- Clusters of 2 fingers are de-clustered whenever the distance between the 2 fingers it contains goes over 150 pixels;
- Clusters of 3 fingers are never de-clustered because of the length of the edges of the triangle;
- Clusters of 3 fingers are automatically de-clustered after 5 seconds;
- All the clusters cease to exist, producing the corresponding de-clustering event, whenever a finger contained in this cluster is removed from the screen.

The events produced by this interaction technique are the following ones: *twoFingersClustered*, *twoFingersDeclustered*, *threeFingersClustered*, *threeFingersDeclustered*.

3.3 ICO: A Formal Description Technique Dedicated to the Specification of Interactive Systems

The ICO formalism is a formal description technique dedicated to the specification of interactive systems [21]. It uses concepts borrowed from the object-oriented approach (dynamic instantiation, classification, encapsulation, inheritance and client/server relationship) to describe the structural or static aspects of systems and uses high-level Petri nets to describe their dynamic or behavioral aspects.

ICOs are dedicated to the modeling and the implementation of event-driven interfaces, using several communicating objects to model the system, where both the behavior of objects and the communication protocol between objects are described by the Petri net dialect called Cooperative Objects (CO). In the ICO formalism, an object is an entity featuring four components: a cooperative object which describes the behavior of the object, a presentation part (i.e. the graphical interface), and two functions (the activation function and the rendering function) which make the link between the cooperative object and the presentation part.

An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e. how user actions affects the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e. when and how the application displays information relevant to the user).

This formal specification technique has already been applied in the field of Air Traffic Control interactive applications [21], space command and control ground systems [22], interactive military [6] or civil cockpits [3].

The ICO notation is fully supported by a CASE tool called PetShop [5][23]. All the models presented in the following of this paper have been edited using it. Beyond, the presented test generation techniques are part of an effort to support MBT in PetShop.

3.4 Modeling of the Interaction Technique Using ICO

Based on the requirements provided in section 3.1, we can build a model of the interaction technique (step 1 of the MBT process) using ICO. **Fig. 4** presents this model, which is made of places (oval shapes), transitions (rectangular shapes) and arcs. Two communication means are proposed by ICO: a unicast and synchronous communication, represented by method calls, and a multicast asynchronous communication, represented by event handling:

- When an ICO proposes method calls, they are each mapped into a set of three places representing three communication ports (the service input, output and exception ports). For instance, on the top part of **Fig. 4**, the places called *SIP_addFinger*, *SOP_addFinger* and *SEP_addFinger* are the input, output and exception ports of the method addFinger. When this method is called (for instance, in the *addFingerToInteraction* transition of **Fig. 3**), a token is created, holding the parameters of the invocation and is put in place *SIP_addFinger*. The transitions that invoke such methods have got a 'I' on the right part of their header.
- When an ICO is able to handle events, it uses special transitions called event handlers such as transition *updateFingerX* in the middle-right of **Fig. 4**. Such transitions are described using a set of information holding the event source, the event name, extra event parameters and a condition that concerns the event parameters. In the example of transition *updateFingerX*, the event source is fx, a value held by place *FINGERS_MERGED_BY_TWO*, the event name is touchevent_up, the event parameters contain an object called info and there is no condition on the parameter. These event handlers may handle events from outer sources or from other models. When the event source is another model, this model contains transitions that raise events. Events are raised using the keyword raiseEvent in the code part of the transition and an "E->" is put in the right part of the header of the transition (see transition *merge2Fingers* of **Fig. 4**).

The model illustrated by **Fig. 4** represents the behavior of the "Finger Clustering" Interaction Technique described in section 3.2. This behavior may be divided into two different parts according to their role:

- **Managing fingers life cycle:** Each finger is added or removed from the interaction technique model. In between, their coordinates may be updated (i.e. the finger has moved):
  - Adding finger to the interaction technique is done using the method addFinger, implemented using the *SIP_addFinger* place, *addFinger* transition and

SOP_addFinger place (see **Fig. 4**). This method is called by a transition of the Low-Level transducer model (see **Fig. 3**). This invocation is made each time a Finger is created to add it to the interaction technique. When the finger enters the interaction technique, it is placed in the *SINGLE_FINGERS* place. This mechanism allows for dynamic appearance of fingers in the interaction technique. To ease the rest of the discussion, we limited the number of fingers instantiated in the interaction technique to 4 using the place *FINGER_LIMIT*. Removing this place would remove this restriction.

  - Removing or updating fingers coordinates is performed by handling events that comes from the Low-Level transducer model (see **Fig. 3**). When a touchEvent_up is received, the corresponding finger is removed from the interaction technique model (this is the case for instance with transition remove1 on the left part of **Fig. 4**). When a touchEvent_update is received, the corresponding point (associated with a finger) is updated (this is the case for instance with transition *updating1Finger* on the top right part of **Fig. 4**).

- **Detecting clusters of fingers:** Each time a finger is added or removed from the interaction technique model, or each time the coordinates of one finger is updated, the clustering or de-clustering of fingers is computed:
  - For two or three fingers, the principle is the same, supported thanks to the preconditions of the *mergeXFingersX* and *unMergeXFingers* transitions, that compute the proximity of the fingers.
  - The 5 seconds timeout for de-clustering three fingers is handled thanks to a "timed transition" (note the [5000] - expressed in ms - line at the bottom of the *unMerge3Fingers* transition) that removes the fingers held by place *FINGERS_MERGED_BY_THREE*.

While we are able to describe the interaction technique, the approach can be applied to other components of the architecture. For instance, **Fig. 3** presents the ICO model of the Low-Level Transducer component of the architecture presented earlier. Note that the *addFingerToInteraction* transition contains an invocation on the interaction technique. This invocation is the one associated with the SIP/SOP places in the Interaction Technique Model. To prevent inconsistent input such as two fingers at the same location (which is physically impossible), a test arc allows to check whether a touch down is associated with a touch point of a finger already on the screen.

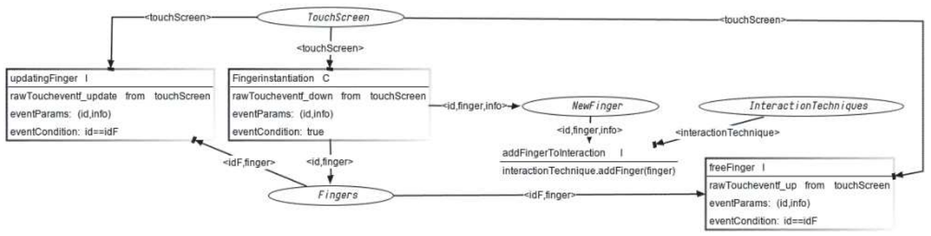

Fig. 3. ICO Model for the Low-Level Transducer

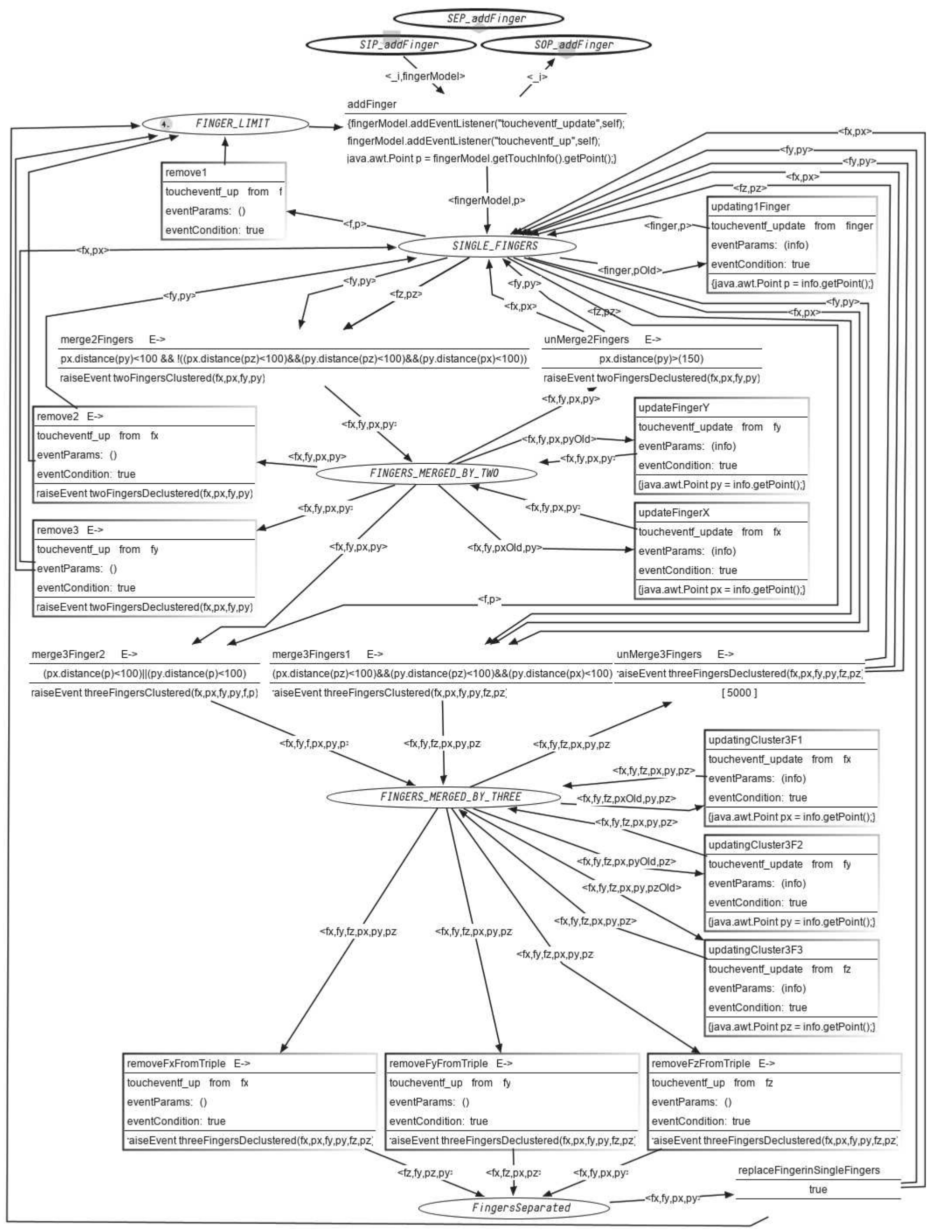

Fig. 4. ICO Model for the finger clustering interaction technique.

4 Generating Test Cases from ICO Specifications

In this section, we focus on steps 2, 3 and 4 of the MBT process (see **Fig. 1**) applied to our case study. We first present our test selection criteria and specification and then present our test generation approach.

4.1 Test Selection Criteria and Test Case Specification

Testing an interaction technique consists in verifying that, for a set of low-level input events, the corresponding high-level event is produced so that components subscribed to it (e.g. application dialogs or widgets) are notified with a well-formed event. This differs from testing the application as done in the work presented in section 2.2. Indeed, in these, the events considered in the test cases are already high-level ones and the verification that is made is that the effect on the UI is the correct one. To perform testing on the interaction techniques requires to i) describe the sequences of actions triggering the events raised by the interaction techniques and to ii) describe the associated events to observe on the interaction technique.

Regarding the finger clustering interaction techniques, this means that we want to be able to identify all the possible sequences of low-level events leading to the raising of the “twoFingersClustered”, “threeFingersClustered”, “twoFingersDeclustered and “threeFingersDeclustered” events in the interaction technique transitions. For illustration purpose, we focus on the raising of the “threeFingersClustered” event.

4.2 Generating Test Cases for the Interaction Technique

To identify the relevant test cases for the raising of the “threeFingersClustered” event, we use the reachability graph of the Petri-net. A reachability graph of a Petri-net is a directed graph G=(V,E), where v∈V represents a class of reachable markings; e∈E represents a directed arc from a class of markings to another class of markings [32]. **Fig. 5** presents the reachability graph of the interaction technique introduced previously. In this graph, each state contains four digits symbolizing the number of tokens contained in the places “FINGER LIMIT”, “SINGLE FINGER”, “FINGERS MERGED BY TWO” and “FINGERS MERGED BY THREE”. For instance, the state “4,0,0,0” at the top means that the “FINGER LIMIT” place contains 4 tokens and that the other places are empty. We take advantage of the APT (Analysis of Petri nets and labelled transition systems) project[2] [7] to generate this graph.

[2] https://github.com/CvO-Theory/apt

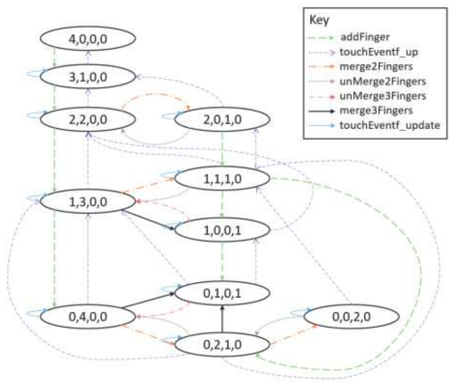

Fig. 5. Reachability graph derived from the ICO model of the interaction technique

As observable in **Fig. 5**, the reachability graph is actually a Finite State Machine with no accepting state. Considering that the event we focus on is raised in the "merge3FingersX" transition, we know that the event must be raised whenever a state of the FSM having a "merge3Fingers" incoming edge is reached. Marking these states (i.e. "1,0,0,1" and "0,1,0,1") as accepting ones allows us to describe the actual grammar of the test cases for the "threeFingersClustered" event. This grammar[3] only misses concrete values for fingers coordinates. The following is an example of test case matching this grammar expressed into Backus-Naur Form (BNF):

```
<testCase> ::= <addFinger> <touchEventf_update> <addFinger> <addFin-
ger> <touchEventf_update> <merge3Fingers>
```

The reachability graph we present in this case study contains values for each place as we intentionally limited to 4 the number of fingers in the interaction technique. However, some touch screens support more than 4 fingers and therefore one may want to use multiple clusters of three fingers. It would be possible to remove this restriction while still being able to apply our process by performing our analysis on a symbolic reachability graph. Symbolic reachability graphs use variables instead of concrete values in the states for the analysis of Petri-nets with such infinite marking, making it possible to express infinite number of states.

To prepare the instantiation of the test scripts, we must focus on how the required values are produced, partly supported by the model of the application. This model describes the conditions under which the transitions are fired. In our case, it describes the constraints on the distance between the points, defining the values domain. When instantiating the test scripts, the integration of these constraints relies on a semi-automated support, where the values are checked at editing time. For instance, in the instantiation of the grammar example proposed above, whatever the coordinates of the three added fingers are, the distance between them must fit the precondition of the transitions "merge3Fingers1" and "merge3Fingers2".

[3] For which the regular expression can be obtained from the FSM using tools such as FSM2Regex (http://ivanzuzak.info/noam/webapps/fsm2regex/)

5 Test Cases Execution

In this section, we discuss the execution of the test of the interaction technique, i.e. steps 5.1 and 5.2 of the MBT process. While the advances we propose are mostly related to test cases generation, we find it important to emphasis the relevance of selecting the test adapter appropriately and to discuss the possible ways to use our test cases.

5.1 Test Adapter Selection

Testing the interaction technique consists in verifying that for a set of input events the corresponding high-level event is produced. Key in executing such test properly is being able to produce an input event that is actually the event expected by the interaction technique as an input, i.e. an event from the low-level transducer. Assuming that we are testing our interaction technique as part of a JavaFX application, this means producing JavaFX Touch Events[4]. However, testing the interaction technique alone may prove to be insufficient to ensure that the interaction technique will behave properly for the end-user. Indeed, while evaluating our approach, we encountered a known issue that no touch events are forwarded to JavaFX by most popular distributions of Linux using a GTK-based desktop environment[5]. In other words, the Touch Provider of these distributions is not producing relevant events for the Low-Level Transducer that cannot, in turn, produce events for the interaction technique. This means that the JavaFX finger clustering cannot be used on a Linux platform even though tests based on JavaFX Touch Event would have indicated that the interaction technique behaves properly. Therefore, when testing touch applications, one may want to produce Operating System-level events and to perform integration testing of the Low-Level Transducer/Interaction Technique couple. Such tests can be executed on the Windows platform by using the Touch Injection technology of the Windows API[6] to produce OS-level touch events as inputs. Regarding Linux, it is worth mentioning that ARM versions of GTK are not prone to the issue presented earlier.

5.2 Test Execution for the Interaction Technique

The execution of the tests on the SUT is an activity that is highly dependent of the way the SUT is implemented. Overall, testing the interaction technique alone requires i) being able to forward the event sequence of the test script to the interaction technique and ii) being able to subscribe to the events the interaction technique produces. The easiest way to test the interaction technique of the SUT is to do it using white-box or grey-box testing. Indeed, in such cases, it is easy to either instrument the class of the SUT responsible for the interaction technique or to encapsulate it in a test adapter with which the test execution environment can interact. Then, the test execution environment can perform the event sequence described by the test script. The role of the oracle is

[4] https://openjfx.io/javadoc/11/javafx.graphics/javafx/scene/input/class-use/TouchEvent.html
[5] https://bugs.openjdk.java.net/browse/JDK-8090954
[6] https://docs.microsoft.com/en-us/windows/desktop/api/_input_touchinjection/

then to determine whether the test passed based on whether or not it received the expected event from the interaction technique in a timely manner.

6 Generalizability of the Approach

While this paper focused on the interaction technique component of the architecture presented in section 3.1, the ICO notation, alongside with its CASE tool Petshop, support the modelling and the test generation for other components of the architecture, as well as GUI Testing as defined by Banerjee et al. [1]. This section highlights the generalizability of the modelling philosophy and of the test case generation approach. Due to space constraint and to the highly SUT-dependent nature of the tests execution, we will however not develop further on test execution.

6.1 Generalizability of the Modelling Philosophy

In addition to interaction techniques, we pointed out in section 3.4 that ICO can be used to model the low-level transducer of a post-WIMP application (**Fig. 3**). Modelling of Logical Input Devices (e.g. fingers) and their dynamic instantiation is covered in [13]. Moreover, [21] demonstrates that ICO allows the description of the Application (dialog part) components, including those with dynamic instantiation of widgets, on examples such as an Air Traffic Control (ATC) plane manager. To validate that our work is compatible with GUI Testing of WIMP application, we modelled the application specified in Memon et al.'s [18] review of advances in MBT for applications with a GUI front-end. We had no trouble describing the behavior of this WIMP application using ICO in Petshop. Combining this with the modelling of post-WIMP interaction techniques demonstrated herein, shows that we are able to model post-WIMP applications.

6.2 Generalizability of the Test Case Generation Approach

Thanks to Memon et al.’s review of advances in MBT [18], we were able to verify that our test generation approach worked for WIMP applications. Indeed, [18] presented various models for the application it specifies, including one being a Finite State Machine. This allowed us to verify that the reachability graph of the Petri-net was the same (name of states aside) as the FSM in [18]. Beyond that, on applications that involve dynamicity such as the ATC plane manager dialog, the approach fits well as each aircraft is added to the dialog model using invocation in the same way as fingers are added to the interaction technique presented in this paper. Yet, as the number of aircraft on the radar visualization is virtually infinite, the use of a symbolic reachability graph is made mandatory, while standard reachability graph can be kept for interaction techniques (as the maximum number of touch points supported by the screen is known).

7 Conclusion and Future Work

Testing interactive applications is known to be a challenging activity, whether we consider WIMP or post-WIMP applications. In this paper, we have shown that while the testing of WIMP applications retained most of the attention of researchers and practitioners in the field of MBT, post-WIMP applications raise new challenges for the community. Indeed, properly testing post-WIMP following the standard Model-Based Testing process requires modelling techniques that are expressive enough to describe the dynamic instantiation of virtual and physical devices, timing aspects, system configuration, etc. Only such models allow the generation of exhaustive enough test cases.

Building on previous work on the Petri-net-based notation ICO (and its associated CASE tool, PetShop), we showed that we are able to propose a toolchain that addresses the need for expressive modelling techniques in order to support the generation of test cases for post-WIMP application following the MBT process. We showed that thanks to the mechanism supported by ICO we are able to support the high dynamicity of post-WIMP applications for all the software components of the architecture. This expressiveness allows for the generation of abstract test case using a grammar derived from the reachability graph of Petri-nets.

As we focused on a specific component of the architecture, i.e. the interaction technique, we found that post-WIMP applications are more sensitive than WIMP applications to the execution platform, as touch event are not always well forwarded to libraries by operating-systems, highlighting the need for integration testing. A future extension to our work would be to implement the generation of integration test cases into PetShop by relying on the different artifacts allowing the communication between models.

Finally, we are currently investigating using such approach for the testing of interactive applications to be deployed in large civil aircraft interactive cockpits. Indeed, following guidance from supplement DO-333 [27] on formal methods to the DO-178C certification process [26], one may use formal specifications during the development of such application. If a formal model of the interactive application is built for supporting reliability arguments (e.g. "low-level requirements are accurate and consistent [26]") we propose to exploit that model to generate test cases from that formal specification (as proposed by Gaudel [12]). Such process could result in more cost-effective test case generation leveraging on available formal models. Beyond, thanks to the expressive power of ICO, such approach could support the adoption of application offering richer interaction techniques (e.g. animations [19] or multitouch [13]) even in safety-critical context (e.g. brace touch [24]).

References

1. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science, 126:183–235, 1994.
2. Banerjee, I., Nguyen, B., Garousi, V., Memon, A.M.: Graphical user interface (GUI) testing: Systematic mapping and repository. Information and Software Technology. 55, 1679–1694 (2013).

3. Barboni E., Conversy S., Navarre D. & Palanque P. Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification. 13th conf. on Design Specification and Verification of Interactive Systems (DSVIS 2006), LNCS Springer Verlag. p25-38
4. Bass, L., Little, R., Pellegrino, R., Reed, S., Seacord, R., Sheppard, S. and Szezur, M.R. The arch model: Seeheim revisited. In User Interface Developpers' Workshop (1991).
5. Bastide, R., Navarre, D., Palanque, P.: A Model-based Tool for Interactive Prototyping of Highly Interactive Applications. In: CHI '02 Extended Abstracts on Human Factors in Computing Systems. pp. 516–517. ACM, New York, NY, USA (2002).
6. Bastide R., Navarre D., Palanque P., Schyn A. & Dragicevic P. A Model-Based Approach for Real-Time Embedded Multimodal Systems in Military Aircrafts. Sixth International Conference on Multimodal Interfaces (ICMI'04) October 14-15, 2004, USA, ACM Press.
7. Best, E., Schlachter, U.: Analysis of Petri Nets and Transition Systems. Electron. Proc. Theor. Comput. Sci. 189, 53–67 (2015).
8. Bowen, J., Reeves, S.: Generating Obligations, Assertions and Tests from UI Models. Proc. ACM Hum.-Comput. Interact. 1, 5:1–5:18 (2017).
9. Campos, J.C., Fayollas, C., Martinie, C., Navarre, D., Palanque, P., Pinto, M.: Systematic Automation of Scenario-based Testing of User Interfaces. In: Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 138–148. ACM, New York, NY, USA (2016).
10. Canny, A., Bouzekri, E., Martinie, C., Palanque, P.: Rationalizing the Need of Architecture-Driven Testing of Interactive Systems. In: Human-Centered and Error-Resilient Systems Development. Springer, Cham (2018).
11. Cronel, M., Dumas, B., Palanque, P., Canny, A.: MIODMIT: A Generic Architecture for Dynamic Multimodal Interactive Systems. In: Bogdan, C., Kuusinen, K., Lárusdóttir, M.K., Palanque, P., and Winckler, M. (eds.) Human-Centered Software Engineering. pp. 109–129. Springer International Publishing (2019).
12. Gaudel, M.-C.: Testing can be formal, too. In: TAPSOFT '95: Theory and Practice of Software Development. pp. 82–96. Springer, Berlin, Heidelberg (1995).
13. Hamon, A., Palanque, P., Silva, J.L., Deleris, Y., Barboni, E.: Formal Description of Multi-touch Interactions. In: Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 207–216. ACM, New York, NY, USA (2013).
14. Hamon, A., Palanque, P., Cronel, M., André, R., Barboni, E., Navarre, D.: Formal Modelling of Dynamic Instantiation of Input Devices and Interaction Techniques: Application to Multi-touch Interactions. In: Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 173–178. ACM, New York, NY, USA (2014).
15. Ladry, J.-F., Navarre, D., Palanque, philippe: Formal Description Techniques to Support the Design, Construction and Evaluation of Fusion Engines for Sure (Safe, Usable, Reliable and Evolvable) Multimodal Interfaces. In: Proceedings of the 2009 International Conference on Multimodal Interfaces. pp. 185–192. ACM, New York, NY, USA (2009). https://doi.org/10.1145/1647314.1647347.
16. Lelli, V., Blouin, A., Baudry, B., Coulon, F.: On model-based testing advanced GUIs. In: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). pp. 1–10 (2015).
17. Martinie, C., Navarre, D., Palanque, P., Barboni, E., Canny, A.: TOUCAN: An IDE Supporting the Development of Effective Interactive Java Applications. In: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 4:1–4:7. ACM, New York, NY, USA (2018).

18. Memon, A.M., Nguyen, B.N.: Advances in Automated Model-Based System Testing of Software Applications with a GUI Front-End. In: Zelkowitz, M.V. (ed.) Advances in Computers. pp. 121–162. Elsevier (2010).
19. Mirlacher, T., Palanque, P., Bernhaupt, R.: Engineering Animations in User Interfaces. In: Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 111–120. ACM, New York, NY, USA (2012).
20. Morgado, I.C., Paiva, A.C.R.: The iMPAcT Tool for Android Testing. Proc. ACM Hum.-Comput. Interact. 3, 4:1–4:23 (2019).
21. Navarre, D., Palanque, P., Ladry, J.-F., Barboni, E.: ICOs: A Model-based User Interface Description Technique Dedicated to Interactive Systems Addressing Usability, Reliability and Scalability. ACM Trans. Comput.-Hum. Interact. 16, 18:1–18:56 (2009).
22. Palanque P., Bernhaupt R., Navarre D., Ould M. & Winckler M. Supporting Usability Evaluation of Multimodal Man-Machine Interfaces for Space Ground Segment Applications Using Petri net Based Formal Specification. Ninth Int. Conference on Space Operations, Italy, June 18-22, 2006.
23. Palanque P., Ladry J-F, Navarre D. & Barboni E. High-Fidelity Prototyping of Interactive Systems can be Formal too 13th Int. Conf. on Human-Computer Interaction (HCI International 2009) LNCS, Springer.
24. Palanque P., Cockburn A., Gutwin C., Deleris Y. & Desert-Legendre L. Brace Touch: A Dependable, Turbulence-Tolerant, Multi-Touch Interaction Technique for Interactive Cockpits. In: International Conference on Computer Safety, Reliability, and Security 2019 (SAFECOMP). Springer, Verlag (2019).
25. Pezzè, M., Rondena, P., Zuddas, D.: Automatic GUI Testing of Desktop Applications: An Empirical Assessment of the State of the Art. In: Companion Proceedings for the ISSTA/ECOOP 2018 Workshops. pp. 54–62. ACM, New York, NY, USA (2018).
26. RTCA. DO-178C Software Considerations in Airborne Systems and Equipment Certification. 2011.
27. RTCA. DO-333 Formal Methods Supplement to DO-178C and DO-278A. 2011.
28. Shneiderman, B.: Direct Manipulation: A Step Beyond Programming Languages. Computer. 16, 57–69 (1983). https://doi.org/10.1109/MC.1983.1654471.
29. Spivey, J.M., Abrial, J.: The Z notation. Prentice Hall Hemel Hempstead (1992).
30. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Softw. Test. Verif. Reliab. 22, 297–312 (2012).
31. Van Dam, A.: Post-WIMP user interfaces. Communications of the ACM. 40.2, 63-67 (1997).
32. Ye, X., Zhou, J., Song, X.: On reachability graphs of Petri nets. Computers & Electrical Engineering. 29, 263–272 (2003). https://doi.org/10.1016/S0045-7906(01)00034-9.