A Model-Driven Engineering Transition-Based GUI Testing Technique

Eman M. Saleh
Software Engineering Department
Applied Science University
Amman, Jordan
e_saleh@asu.edu.jo

Omar Al Sheik Salem
Software Engineering Department
Applied Science University
Amman, Jordan
O_alsheiksalem@asu.edu.jo

***Abstract*—Model Driven Engineering (MDE) have arisen as a new software development paradigm which is based on creating a set of models that represent the GUI; afterwards to generate the GUI based on these models using a series of transformations to convert the models between the different levels of abstractions, which enables the automation of the development process. This inspires us to think of a model-based testing technique that is able to test the GUIs that are designed using Model-Driven engineering by finding the proper model that can serve as a testing model.**

This paper proposes model-based testing technique that is derived from the design models used to develop the GUI in the Model-Driven Engineering paradigm.

Keywords-Concur Task Trees; Model-based testing; Model-Driven Engineering; Task models.

I. INTRODUCTION

Model-Driven Engineering (MDE) and Multi-Platform User Interface Development (MPUID) is a well known paradigm that is targeted toward creating the user interface out of a set of predefined models [1]. The excessive work that had been done in MDE aimed at reducing the time needed to re-implement the GUI of an application for every target platform or context of use (i.e. IOS, Windows …). The idea is based on creating a number of models, starting from abstract models that are gradually customized with platform dependent information until a final interface that is targeted to a specific platform is reached [2]. Different contributors have used different models in order to define the user interface aspects. The literature review showed that most of the contributors and successful work such as TERESA [3] and UsiXML [4]. It is clear that most of the work in MDE was based on task models. Recently a general framework, namely the CAMELEON reference framework [5], was set and followed to standardize the work in the area of model-driven engineering. The framework is based on defining four levels of abstraction as shown in Fig. 1 [6].

Each level is then refined by adding a more specific platform information until the final user interface which is the platform specific model is reached. Model-based testing methods aim to the automation of test case generation based on a model of the system under test (SUT) [7][8]. This needs a deep investigation on the properties of the chosen model [9] and poses a new difficulty of the need to implement a new model for testing purposes. To avoid creating a new model, we have chosen the task model as a base of the testing criterion, as well as, to conform to the CAMELEON reference framework.

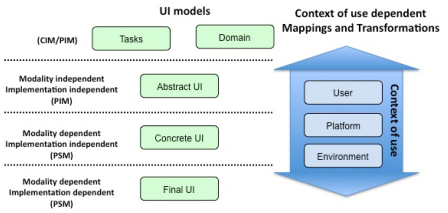


Figure 1. CAMELEON reference framework [6]

In [10] we proposed to use the DSM [11] in order to derive some test cases. The DSM is based on State Charts, were each state represents a presentation unit (Window or screen) the initial dialog states model is generated automatically from the task model; The model was a step in the MDE approach for designing multi-platform UIs and to suit devices with small screen sizes [2], this leads to a conclusion that the DSM is not much suitable as a test oracle because of the large number of states and transitions that may lead to a test case explosion problem.

In this paper we have used an extended version of the DSM to minimize the number of test cases and introduce a model-based GUI testing technique that suits GUIs which are developed using an MDE approach. The testing technique in this paper takes into consideration all main requirements of the Model-Based testing: The test oracle, the test coverage criteria and the derived test sets.

II. RELATED WORK

The testing technique of this paper is based on the Concur Task Tree formalism (CTT)[12] and the navigation model which works as the test oracle. The following subsections briefly explain these models.

A. CTT Task Model

CTT notation is a hierarchical task model that provides a graphical syntax, a hierarchical structure and

978-1-4673-9795-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCI.2015.109

a notation to specify the temporal relation between tasks [12], an example of CTT task model is shown in Fig. 2. With this notation, tasks can be classified into four categories: abstract tasks , interaction tasks , user tasks and application tasks . Tasks at the same level can be connected by temporal operators like choice ([]), independent concurrency (|||), concurrency with information exchange (|[]|), disabling ([>) , enabling (>>), enabling with information exchange ([]>>), suspend/resume (|>) and order independence (|=|).[12].

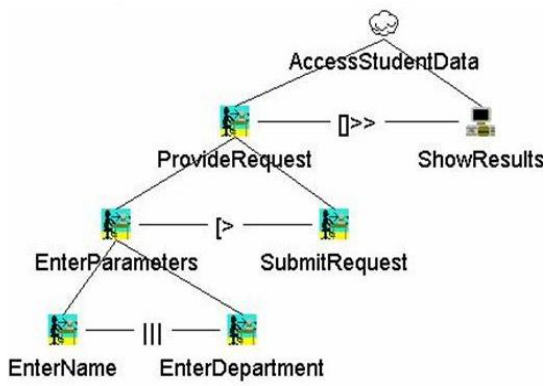


Figure 2. CTT

B. *The navigation model (EDSM)*

The CTT is not suitable as a test oracle or to derive test cases due to its high level of GUI abstraction, and it does not represent many aspects of the dynamic behavior of the GUI such as: inputs, events, and transitions to new states. For this purpose we use the EDSM as the test oracle.

The Navigation model (EDSM) is an extended version of the DSM [11]. The EDSM is a state chart model that is created in an algorithmic way based on the CTT model. For the purpose of using the model as a test oracle, the model is derived using the semantics of the temporal operations and the facets of the tasks in CTT model. Besides that, the algorithm has been extended to take into consideration the guards and conditions more precisely also the states of the EDSM are annotated with abstract dynamic events that are to be mapped to the abstract and concrete GUI properties. Compared to the DSM, the EDSM has less number of states and transitions. The EDSM that is extracted from the CTT in Fig. 1 is shown in Fig. 3.

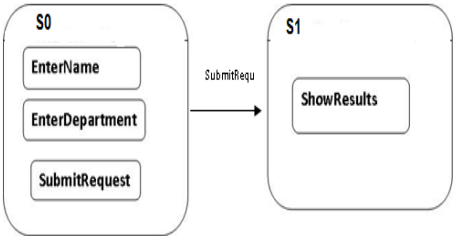


Figure 3. The EDSM for the CTT in Fig. 1

The details of the algorithm are out of the scope of this paper and are defined in [11]. The EDSM features that are most relevant to this paper are:

1. EDSM elements are annotated with mapping information to next reification model elements, i.e. the abstract UI model, and these are mapped accordingly to next models until reaching a final UI level of actual windows of the system. This governs test case execution.
2. Transitions between the states in the EDSM are defined using the semantics of the temporal operators and the facets of the tasks; these transitions are mapped to abstract events at the abstract UI level.

Fig. 4 shows the position of the test oracle EDSM in a multi-platform (MDE) technique [2]. The EDSM does not affect other transformations. We have added event transformations from abstract events to final events in order to conform to the CAMELEON reference framework and MDE transformations. The abstract events are used in the test oracle (EDSM) to derive state transitions in parallel to their equivalent Final events that govern the actual GUI navigation. This shows that our MDE testing criteria is a multi-platform testing technique.

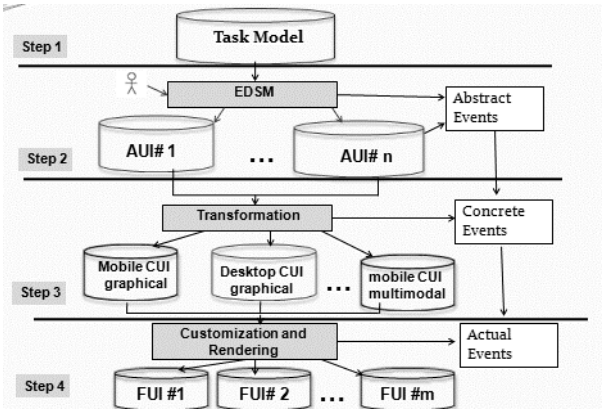


Figure 4. The test oracle EDSM within an MDE approach, modified from [2]

III. THE TESTING TECHNIQUE

In order to generate test cases automatically we need to define the three main requirements for the automated model-Based Criterion. Basically, (1) The test oracle (2) A coverage Criteria and (3) a test case generator.

In this paper the EDSM is the test oracle, the coverage criterion is transition coverage and the teat cases are generated based on the EDSM and represented in a transition table.

Following are the steps of the proposed testing technique:

1) According to MDE and the CAMELELEON reference framework, the designer starts by creating (drawing) the task and domain models. The CTT elements specify the type of the task (Abstract,

interaction, application) and the temporal relation of every task with its neighbour siblings. Then the designer builds the domain model Using UML class diagrams.

2) Annotating the CTT task model and Mapping to Domain Model: Leaf tasks of the CTT represent the objects or widgets that appear on the user interface, these tasks are either interaction tasks or application tasks. Annotation of leaf tasks is necessary to build the new model that serves as the test oracle. Each leaf task should be annotated with two main attributes: facets and task item.

- Facets identify the role of task in the GUI, as whether it is an input, output, navigation or a control task. These are similar to task properties (attributes) defined by IdealXML [13]. Currently we'll use the facets defined by IdealXML. We are using a similar annotation by allowing the designer to specify these task facets and task item attributes when building the CTT model in contrast to IdealXML [13] where these facets were specified at the Abstract user interface model. We added facets at an earlier stage of the design namely as an attribute of tasks in the task model in order to derive the navigation model and the transition between the states in the navigation model which represents our test oracle.

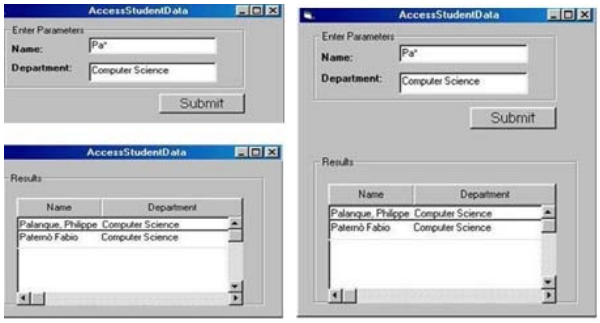
- The task item attribute, represents a mapping between the task model (CTT) and the domain model (class diagram). This attribute is needed to link the task with a corresponding affected element of the domain model and hence to the application code elements later in the design process. Part of the annotations for the task model in Fig. 2 is shown in table 1.

3) The EDSM is generated automatically from the CTT model, see Fig 3.

TABLE1. Annotations of CTT in Fig.3

Task	Type	Facet	Task item
EnterName	Interaction	Input	Student.name
EnterDepartment	Interaction	Input	Student.Department
SubmitRequest	Interaction	Navigation	ResultWindow
ShowResults	Application	Control	SearchAnd displayResults()

Every state of the EDSM is mapped to an abstract container in the next level of abstractrion; until reaching the final GUI constructs. For example states S0 and S1 in EDSM in Fig. 3 are mapped to windows: *Results* and *EnterParameters*, respectively, of the final UI (Fig. 5). In MDE a main goal of the modeling (using abstractions and refications) is to develop GUIs for many target platforms [1] without reimpleming the GUI for every target patform. Hence; following set of transformations, many representations may be extracted from the CTT and EDSM depending on the target platform. For example Fig. 4 shows two possible representations of the final UI. The final UI representation does not affect the testing process since we rely on a more abstract model namely the EDSM which is later annotated with mappings to actual windows. On the other hand; tasks with navigation and control facets (in the CTT) are hmapped using transformations to actual events of the



(a) Final UI for a mobile device (b) Final UI for a PC

Figure 5. Final UI

final UI. For our example the task *SubmitRequest* has a navigation facet accordingly, it was mapped to a transition the EDSM (Figure 3) further transformations will map this event to the actual GUI event for example a mouse click on the button *SubmitRequest* in the final UI.

4) Creating the State Transition table: This table contains the states of the EDST at the rows and the events at the columns. The entries of the table show the target state when the event is executed in the source state. This table is used to derive test cases.

5) Executing the test: In traditional software testing test oracles might be used after executing the system to compare the actual results with the expected results (defined in the test oracle). This does not work in testing GUIs due to the huge input space and nature of the GUI; where the same action may lead to different states of the system depending on the current executing state. Hence; the test oracle usage is interleaved with the execution of the system.

The test starts with executing the main screen of the system on parallel to executing state S0 of the EDSM. A test case is picked from the state transition table; applying the event on the table to current window and comparing the target window in the actual GUI with the target state in the oracle (the EDSM).

IV. COVERAGE CRITERIA AND TEST CASE EXTRACTION

Different state based testing criteria exists, one criterion requires that the tests visit every state or every transition. Another potentially higher coverage is the switch coverage criterion [7] requires that at least one test cover each transition sequence of N or less length. If you cover all transitions of length one, then “N-1 switch coverage” means “0 switch coverage.” Notice that this is

the same as the lowest level of coverage; visiting every transition.

The idea of this work requires the test oracle and the GUI under test to be executed simultaneously. The test oracle (EDSM) starts at state S0 which is already mapped to the main window of the GUI. The test starts by the first row of the transition table and executes events in order; when it picks the first event from the transition table, this event corresponds to a transition in the EDSM (Recall that events are extracted from tasks with navigation or control facet) and the EDSM will move to the corresponding next state and the GUI will show a new window according to the actual event. If the shown window corresponds to the new state of EDSM the test will continue by picking the next event until reaching the final state of the EDSM. In case of mismatching between the target state and its corresponding actual shown window a test failure is reported. It is easy detect the location of the error in terms of the window and the event that causes the error by using state mapping to actual window and transition to actual event mappings.

The coverage criterion in this paper is a transition-based testing criterion that is higher than N-switch criteria as we choose the test cases from the transitions table which is defined by tasks that derive transition. This is similar to transition table based testing and covering every row in the table. The transitions table for our example will show one row because we have only one task with a navigation face (SubmitRequest) which will exercise the navigation from the window *EnterParameters* to the window *ShowResults* causing visiting the target window. Picking events from the transition table, guarantees exercising all valid events, and hence visiting all states and all transitions.

V. CASE STUDY

In this section we present a portion of the case study "Rent a car" to illustrate the transition–based testing technique.

When following the MDE technique, the designer draws and annotates the CTT, see Fig. 6. The annotations of the tree in Fig. 6 are shown in table 2. Note that these annotation connect the tasks to the corresponding implementation aspects, for example the task *InputFirstname* will input data that is saved in the attribute *name* of class *Cust*, while the task *Submit* will lead to opening the window named *ShowConfirm* in class *Main*. Many other actions can be considered such as the task SubmitResults in CTT in Fig. 1 has a control facet (see table1) which means that execution of this task must lead to calling the function *SearchStudent* of class *Student*. This information has been embedded into the different mappings within the MDE design process and can be used on different testing criteria. We only consider tasks with navigation facets which are necessary for transition-based testing of this paper.

TABLE 2: Annotations of CTT in Fig. 6

Task	Type	Facet	Domain element
InputFirstName	Interaction	Input	Cust..name
InputlastName	Interaction	Input	Cust.Last
SelectCarClass	Interaction	Input	Car.Class
SelectTransmType	Interaction	Input	Car.Transm
SelectCardType	Interaction	Input	Card.Type
SelectYear	Interaction	Input	Car.Date.year
SelectMonth	Interaction	Input	Car.Date.year
Submit	Interaction	Navigation	Main.showConfirm
Cancel	Interaction	Navigation	Main.PersonalInfo
Next	Interaction	Navigation	Main.Carinfo
Next	Interactiom	Navigation	Main.PaymentInfo

After annotating the task model and deriving the annotation matrix the EDSM is automatically derived in an algorithmic way. For our case study the EDSM is shown in Fig. 7. Note the addition of the next task to both the annotation table and the states S0 and S1 in the EDSM, this is due to creating a new state when an enabling operator (<<) or an enabling with information exchange operator (<<[]) is encountered [11]; this new state needs a transition to the next state in the EDSM and hence a navigation to the next window in the final UI.

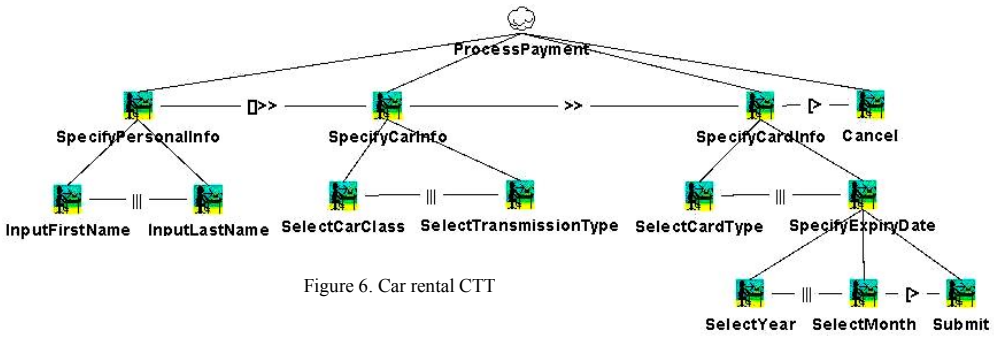


Figure 6. Car rental CTT

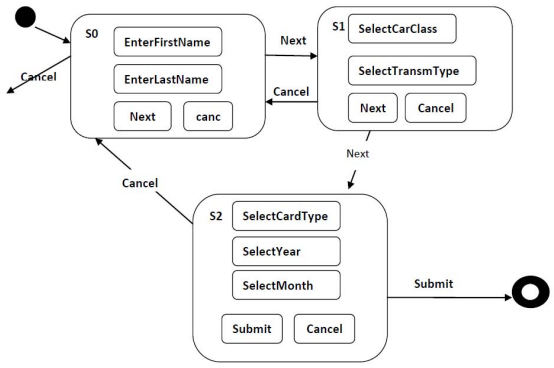


Figure 7. EDSM for CTT in Figure 6

The last step in the testing method is to create the state transition table. Table 3 shows the transition table for our study, each state is represented by a row in the table, where events are at the columns and entries show the target state on that event. The number of test cases equals the number on none empty table cells.

TABLE 3 State Transition table for EDSM in Fig 7

	Next	Submit	Cancel
S0	S1	-	-
S1	S2	-	-
S2	-	Sfinal	S0

The mapping information includes mapping states of the EDSM to windows in the final UI and mapping abstract events (transitions) to actual GUI events. Suppose that S0 is mapped to window *PersonalInfo,* S1 is mapped to window *CarInfo*, and S2 is mapped to window *PaymentInfo*. The tasks with navigation facets (Next and cancel) are mapped to actual widgets on the windows, currently let's assume command buttons *Ok* and *Cancel* appear on each final UI window. At this case the mapping info will map the abstract event *next* to the action performed on button Ok which let's say a left mouse click.

The EDSM starts at the initial state at the same time the GUI starts at its main window. The testing starts traversing the transition table in row wise fashion. Picking the first event (next), executing it on EDSM and executing the corresponding actual event on the window *PersonalInfo*. If the window *CarInfo* executes (appears on the screen) the testing goes back to S0 repeating the same steps with the next event. If the actual GUI window fails to open or another window has been appeared; a test failure is reported registering the window name and the event that causes the error.

VI. CONCLUSION

We presented a transition based testing criteria that is applicable in the MDE environment, the technique avoids putting more effort in creating the testing model by extracting this model from the basic CTT model that is already created in the development process.

Test oracles contribute significantly to test effectiveness and cost. The frequency of comparison is ignored except at two important points, O_{all} and O_{last}. O_{all} requires checking the equality of expected output and the actual output after every event, while O_{last} requires checking the equality of expected output and the actual output after the last event of the test case [14].

Given the fact that most errors in the GUI occur after opening a new window or terminating an existing window; our test oracle (EDSM) is comparable to O_{all} in effectiveness and O_{last} in cost. This is due to the fact that the EDSM includes only the transitions (events) that correspond to tasks with navigation or control facet, which actually enforces opening a new window after closing a previous one. This minimizes the number of test cases and focuses on most common errors locations.

Due to simultaneous execution of the test oracle and the GUI, the technique can easily detect the error location by finding the corresponding window that is mapped to the current state, and the actual event at the point of transaction.

Another important feature of the technique is its applicability in the MDE approach and more specifically in multi-platform user interface development, as the EDSM is an abstract model that can be annotated and transformed to any next level of reification depending on the target platform.

Future work will focus on more detailed mapping to different levels of abstractions in the CAMELEON reference framework and considering invalid input space into consideration.

ACKNOLEDGMENT

The authors are grateful to the Applied Science Private University, Amman, Jordan, for the full financial support granted to this research.

REFERENCES

[1] J. Vanderdonckt, "Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures", Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008 , Bucharest, pp. 1-10, 2008.

[2] Eman Saleh, Amr Kamel and Aly Fahmy, "An MDE Design Approach for Developing Multi-Platform User Interfaces", WSEAS Transactions On Computers Journal, Issue 5, Volume 9, ISSN: 1109-2750, ACM press, pp. 536-545, May, 2010.

[3] F. Paterno, and C. Santoro, "One model, many interfaces," In Christophe Kolski and Jean Vanderdonckt, editors, CADUI 2002, volume 3, pp. 143-154, 2002.

[4] Q. Limbourg, J. Vanderdonckt, B. Michotte, and V. López , "UsiXML: a Language Supporting Multi-Path Development of User Interfaces," Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, pp. 200-220, 2005.

[5] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J., A Unifying Reference Framework for Multi-Target User Interfaces, Interacting with Computers, Vol. 15, No. 3, pp. 289-308, June 2003.
[6] Introduction to Model-Based User Interfaces, W3C Working Group Note 07, available at: http://www.w3.org/TR/mbui-intro/, January 2014.
[7] A.P. Mathur, "Foundation of Software Testing," Pearson Education India, 81-317-0795-4, 1st Ed, 2010.
[8] Marlon Vieira, et. al. "Automation of GUI testing using a model-driven approach", Proceedings of the 2006 international workshop on Automation of software test, pp. 9-14, ACM, 2006.
[9] Catherine Dubois, Michalis Famelis, Martin Gogolla, Leonel Nobrega, Ileana Ober, et al., Research Questions for Validation and Verication in the Context of Model-Based Engineering. International Workshop on Model Driven Engineering, Verication and Validation – MoDeVVA 2013, Oct 2013, Miami, United States. 1069, pp. 67-76, 2014.
[10] Eman Saleh, "Towards Automatic GUI Testing Using Task and Dialog Models", Proceeding of SERP'14, Las Vegas, 2014.
[11] Eman Saleh, Aly Fahmy and Amr Kamel, "Dialog States a Multi-Platform Dialog Model", ECS journal, vol. 33, No. 2, Egypt, Sep. 2009, pp 1-9.
[12] F. Paternò, C. Mancini, and S. Meniconi, "ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models", in Proceedings of the Interact'97, 1997.
[13] F. Montero, V. Víctor López Jaquero, J. Vanderdonckt, P. Gonzalez, M. Lozano, and Q. Limbourg, "Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML", Lecture Notes in Computer Science, Vol. 3941, Springer-Verlag, Berlin, pp. 161-172, 2005.
[14] A. M. Memon and Q. Xie. Using transient/persistent errors to develop automated test oracles for event-driven software. In Proceedings of The International Conference on Automated Software Engineering 2004 (ASE'04), pages 186–195, 2004.