Interactive System Testing using Interaction Sequences

Jessica Turner
University of Waikato
Hamilton, New Zealand
jdt11@students.waikato.ac.nz

ABSTRACT

Interaction sequences (ISeqs) are an abstraction of interactive systems which allow us to inspect the interactive system behaviour. In this research, ISeqs are used to support interactive system testing. In interactive system testing the components of an interactive system, the functional and the interactive, are often tested separately. However, errors can still occur when these components overlap. Therefore, we must investigate ways to test this overlap as part of a more comprehensive testing approach. ISeqs provide a clear view of this overlap, which we use to inform a model-based testing approach. By testing not only the functional and interactive components of a system, but also this overlap, ISeqs provide us with a way to better cover the interactive system state space, improving system reliability and safety.

CCS CONCEPTS

• **Software and its engineering** → **Formal methods**; *Model-driven software engineering*;

KEYWORDS

Formal Methods; Model-Based Testing; Interaction Sequences; Interactive System Testing

EICS '18, June 19–22, 2018, Paris, France

ACM Reference Format:
Jessica Turner. 2018. Interactive System Testing using Interaction Sequences. In *EICS '18: EICS '18: ACM SIGCHI Symposium on Engineering Interactive Computing Systems, June 19–22, 2018, Paris, France.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3220134.3220148

INTRODUCTION

In this research, ISeqs are used as an abstraction of interactive systems to allow us to inspect the interactive system behaviour. Specifically, they provide us with a clear view of the overlap between the interactive and functional components of an interactive system. The interactive component of an interactive system is the user interface of that system while the functional component is the underlying functionality, set out as a set of instructions in a specified programming language. The overlap component stores the information required for these two components to "communicate", i.e. how they interact with each other.

Despite extensive testing processes, errors can still occur in interactive systems (see [1, 3]). As stated by Dijkstra, "Program testing can be used to show the presence of bugs, but never to show their absence" [2]. Therefore, while we cannot prove an interactive system to be free of error, we can improve reliability and safety of such systems by identifying and removing as much error as possible.

Several strategies exist for testing the interactive and functional components of interactive systems. However, there is less research in testing the overlap of these components, specifically with ISeqs. In existing research (see [5]), the overlap component is not tested comprehensively (comprehensive meaning inclusion of test oracles).

Therefore, we must investigate new ways to test the overlap component comprehensively to support interactive system testing. In this paper we will discuss the use of ISeqs as an abstraction of the interactive system and how we use this to inform a model-based testing strategy. This will provide better coverage of the interactive system and improve system reliability and safety.

USING INTERACTION SEQUENCES AS AN ABSTRACTION

The main contribution of this research is to define the theory of ISeqs and how they can be used for interactive system testing. Here we discuss creating ISeqs and the types of tests we can generate.

Assuming a good software engineering process has been followed we can create ISeqs using information from pre-existing design artefacts, such as specifications, requirements, prototypes, or from the interactive system itself etc. In this research, we have chosen to focus on task-widget based sequences, as these allow the length of sequences to be constrained easily, resulting in a more tractable state space. To build sequences of this type we require information about the widgets of the system and their associated interactions, in addition to task knowledge.

Formalised ISeqs are built from interaction steps using knowledge of the widgets and their interactions. The steps are of the form: "(Interaction) (Widget) (Number of Interactions)". The interaction is how an end user would interact with the specified widget, based on the user interface. This is followed by the number of interactions required for this particular step of the sequence. For example, the step "Press Off 1" would indicate that an end user may press the off button once. We use these steps to build sequences for each task of the interactive system.

We use task information to group sets of interactions together, which provides us with a predetermined "end-point" to the sequence. This end point encapsulates the "goal" that the end user was attempting to achieve by carrying out this task. Using observations of the functional component, we make start and end assumptions for a given task. These define the state the system should be in before the execution of the sequence and after the sequence has been executed successfully. Defining these assumptions ensures reproducibility of the sequence on simulation or execution.

For example, a smart light sensor system senses when an end user enters a room and switches the light on. After a pre-defined time out period when no movement is sensed the light is switched off. The level of light is determined by the time of day that the end user enters the room. In the early morning the light is dimmer and in the evening the light is brighter. An end user may determine the appropriate schedule and light settings as they require.

This system has the following observations: LightLevel, the level the light is currently set at based on the time of day; Movement, whether movement has been sensed or not; Time, the current time; and State, whether the light is switched on or off. The task specifies an end user switches on the light at 1am. The assumptions are as follows:

Start assumptions:
LightLevel: 60
Movement: No
Time: 0059
State: Off

End assumptions:
LightLevel: 60
Movement: Yes
Time: 0100
State: On

As there is only a single widget in this interactive system, the light itself, the ISeq for this task is very simple: "Trigger Light 1". Note that as the steps are defined based on the user interface we do not attempt to model the end user in these interactions. That is, the interaction "trigger" in this step could refer to several different physical actions made by the end user, but we abstract this to the interaction "Trigger" in consultation with the widget information provided. While this is a trivial example, typically we have far more complex systems to investigate, resulting in longer and more interesting ISeqs.

We use these formalised ISeqs to inform the modelling process. There are several models we could have used, however, we chose finite state automata (FSA) for their simplicity and well-defined theory (see [4]). The FSA are used to explore sequences of varying lengths for tasks and to explore properties of ISeqs.

ISeqs as an abstraction simplifies the process of investigating and testing the overlap component, as we can ignore the complexities of the actual implementation of the interactive system. However, applying this simple abstraction to complex large interactive systems can result in lengthy ISeqs with seemingly intractable state spaces. Therefore, investigation was required into finding ways we could control this state space. This led to identification of the "self-containment" property.

The self-containment property was observed in the example systems we investigated. It was evident that parts of ISeqs could be "grouped" together. Using this property we were able to abstract these self-contained groups into an abstract state to reduce and expand the state space as required. This led to the development of a formal definition of self-containment and proofs of the ability to abstract and expand models using this property.

The self-containment property not only provided greater control over the state space, but also allows us to specify only certain parts of an ISeq to model and test. This provides control over the set of tests created for the overlap component. For example, in a safety critical interactive system a test suite may focus specifically on safety-critical sequences, ignoring non-safety critical aspects of the system. However, if investigation into the non-safety critical aspects is required, we can simply expand the abstract state.

Using the FSA models, ISeq assumptions, and specifications of the interactive and functional components of the system, we can generate abstract tests for the overlap component. An abstract test is a test which adheres to no specific test language, it can then be used to create a concrete test which is defined using some testing language and often in a specific testing tool. The reason we define abstract tests is to allow for flexibility of the testing tool or language used to create the concrete tests, regardless of what language or tools are used to create the interactive system and corresponding test suite.

Abstract tests can be generated to ensure at any given step the following step is active and available. We can generate tests to ensure that our start and end assumptions are correct (as test oracle equations), and check that each step executes the correct corresponding behaviours as expected. It is important to note that these tests are designed to ensure that an ISeq behaves as expected, and consequently the overlap component behaves as expected.

PROGRESS

Previously, we introduced the concept of the self-containment property [6]. The self-containment property has since been formally defined and proven to demonstrate it is useful in controlling the

state space. This property is used with the ISeq models to simulate sequences for exploration. Work has begun into generating abstract tests from these models, however, further exploration is required into the types of tests we can generate.

FUTURE WORK

In this paper we have discussed the need to investigate comprehensive ways in which to test the overlap component of an interactive system. We have discussed ISeqs as one solution to this problem, and discussed how these formalised sequences can be used to inform a model-based testing approach.

In future work further investigation is required into the types of tests we generate from FSA. It would also be useful to find ways in which to automatically convert abstract tests to concrete tests in order to reduce time costs, however, this would force a specific testing tool and/or language to be used when following this approach. Furthermore, exploration is needed into the types of issues specifically identified by testing the overlap component in comparison to traditional interactive and functional component testing.

REFERENCES

[1] CNBC. 2017. Uber suspends self-driving car program after Arizona crash. Website. Retrieved March 13, 2018 from https://www.cnbc.com/2017/03/26/uber-self-driving-car-arizona-crash-suspended.html.

[2] Edsger Wybe Dijkstra. 1976. *A Discipline of Programming*. Vol. 1. Prentice-hall Englewood Cliffs.

[3] U.S. FDA. 2015. Infusion Pumps. Website. Retrieved December 2, 2015 from http://www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/GeneralHospitalDevicesandSupplies/InfusionPumps/.

[4] John E Hopcroft. 1979. *Introduction to Automata Theory, Languages, and Computation.* Pearson Education India.

[5] Bao N Nguyen, Bryan Robbins, Ishan Banerjee, and Atif Memon. 2014. GUITAR: an Innovative Tool for Automated Testing of GUI-driven Software. *Automated Software Engineering* 21, 1 (2014), 65–105.

[6] Jessica Turner, Judy Bowen, and Steve Reeves. 2017. Supporting Interactive System Testing with Interaction Sequences. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '17)*. ACM, New York, NY, USA, 129–132. https://doi.org/10.1145/3102113.3102149