Developing Testing Techniques for Event-driven Pervasive Computing Applications

Atif M. Memon
Department of Computer Science
University of Maryland
College Park, MD 20742
atif@cs.umd.edu

1 Introduction

As we move closer to the world of pervasive computing, embedded devices, sensors, and software agents find increasingly more applications in today's inter-networked world. Consequently, several classes of event-driven software (EDS) are becoming ubiquitous, including:

1. graphical-user interfaces (GUIs) [4, 18] that are now seen in cars, phones, dishwashers, washing machines, etc. A user interacts with complex underlying software by performing actions such as menu-clicks and selections on the GUI. The software responds by changing its state and/or producing an output, and waits for the next input.
2. web applications [42] that are used via computers, televisions, PDAs, cell phone, and device-specific browsers. Users perform actions from a multitude of platforms to interact with these applications that change their state, produce outputs, and wait for the next user action.
3. network protocols [38] that form the basis for all network traffic. Protocol Data Units (PDUs) are sent from one networked device (node) to another. The recipient node processes the PDU, responds by changing its state and/or sending a PDU, and waits for the next PDU.
4. embedded software [36] that controls modern buildings, cars, elevators, fire-alarms, etc. Sensors send signals to the software that changes its state, sends output signals to control devices, and continues to wait for signals.
5. software components (and objects) [40] that form the building-blocks of most of today's large software systems. Messages are sent from one component to another. Components react by changing their internal state, respond with messages, and/or wait for the next message.
6. device drivers [9], database applications [11], heterogeneous active agents [6], robot software [7], simulation software [3], and visualization software [12] that also use a similar model of execution, *i.e.*, they receive interrupts, database-operations, events, commands, simulation-signals, and user inputs, respectively, to change their internal states, produce outputs, and wait for subsequent inputs.

As these EDS are used in critical applications (*e.g.*, air-traffic controllers [1] and space missions), [2] their quality becomes important. There are several useful techniques that may be employed during the software development and deployment process to help improve software quality. An important technique is *software testing*, in which test cases are created and executed on the software. Currently, there are a large number of testing techniques available, and new areas of testing research are being identified. Current research in testing continues to produce new techniques to help improve software quality and create new research opportunities.

One of the reasons that so many testing techniques exist is that different types of software require different types of testing. Moreover, testing has multiple goals such as performance, correctness, and usability; satisfying each testing goal may require the application of a different technique or the development of a new one. Several researchers, including us, have shown that existing testing techniques do not apply directly to certain classes of EDS, *e.g.*, database applications [11], GUIs [18, 19, 34], device drivers [9], web applications [35, 37], and network protocol implementations [41]; testing them requires the development of new techniques [1].

Our Position: We posit that fundamental changes have to be made to existing testing techniques, their underlying models, and algorithms to test EDS.

[1] http://catless.ncl.ac.uk/Risks/23.41.html
[2] http://www.space.com/scienceastronomy/solarsystem/mpl_latest_update.html

1.1 Challenges of Testing EDS

EDS is gaining popularity because of the flexibility that it offers to both developers and users: (1) it allows a software developer to implement the EDS by coding (reusable) *event-handlers* (program code that handles or responds to an event) that can be developed and maintained fairly independently, (2) it gives many degrees of freedom to the software user, *e.g.*, in a GUI, the user is not restricted to a fixed ordering of inputs, (3) it enables complex systems to be built using a collection of loosely coupled pieces of code, and (4) in interconnected/distributed systems, event handlers may be distributed, migrated, and updated independently. However, this flexibility creates problems during execution primarily because of the large number of permutations of events that need to be handled by the EDS. Since, in principle, event handlers may be executed in any order, we have shown that unanticipated interactions between them lead to software failures [32,33,41]. The large number of interactions also create new challenges for quality assurance (QA), as described next.

1. The **space of possible interactions** with an EDS is enormous. Each event sequence can result in a different state, and the software may, in principle, need to be tested in all of these states. This large space of interactions results in a large number of input permutations, requiring a large number of test cases.
2. Events performed on EDS drive it into different states. Not all events are allowed in each state. Explicit or implicit protocols specify the allowed (and sometimes disallowed) event sequences. EDS must not only be checked for event sequences allowed/disallowed by the protocols but also for those left unspecified. Hence, testing EDS requires the development and execution of a large number of **off-nominal test cases**, *i.e.*, those that test the software for invalid input event sequences.
3. It is difficult to determine the **test coverage** of a set of test cases. For conventional software, coverage is often evaluated using the amount and type of underlying code exercised. These coverage criteria alone do not work well for EDS, because in addition to satisfying conventional measures of coverage, it is important to cover the different interactions between event handlers.
4. It is difficult to design robust **test oracles** (mechanisms to determine whether software has executed correctly for a test case) for EDS. In conventional software testing, the verification is done after the end of test case execution. The entire test case is executed by the software and the final output is compared with the expected output. In contrast, test case execution of EDS requires that the verification and test case execution be interleaved because an incorrect state may prevent further execution of the test case, as events in the test case may not be allowed in the incorrect state. Thus, execution of the test case must be terminated as soon as an error is detected. Also, if verification is not done after each step of test case execution, errors may be overlooked and pinpointing the actual cause of the error becomes difficult.
5. The **run-time environment** in which an EDS executes may change the software's behavior. The space of possible run-time environment configurations is determined by the platform (hardware and software) in which the EDS executes. Since many EDS systems (*e.g.*, web applications) are expected to execute in a large number of client configurations, the space of all possible configurations becomes very large, thereby requiring extensive testing.
6. Finally, it is difficult to create a **representation** of EDS. It has been shown that finite state machine (FSM) models as well as representations for conventional software fail to scale for large EDS [10, 18, 39].

An additional challenge that hinders research and development of techniques for testing EDS is that since very few specialized techniques exist to test EDS, unlike conventional software, there is little tool and artifact support for experimentation. For example, there are no *fault-seeding models* [8] for EDS, which are essential for running controlled testing experiments. Hence it is difficult to compare new testing techniques with existing ones. Some of the challenges mentioned above also apply to conventional software. For example, many conventional software applications also have large input spaces. Numerous researchers have addressed (and continue to address) issues pertinent to conventional software testing. We focus specifically on EDS testing because of its growing need, especially for a world of interconnected devices.

1.2 Our New Approach – An Event-flow Model to Test EDS

Many researchers have used state-machine models to test specific classes of EDS [2, 4, 15, 16, 39]). Recognizing the problems of scalability with these state-based models [10, 18, 39], we have developed a new representation of EDS based on event interactions. The key to the success of this representation is that it does not contain explicit state information. At a theoretical level, we have:

1. developed new models of the EDS's "event space," *i.e.*, the software's *events and their interactions*. For example, a structure called an *event-flow graph* represents allowable event sequences in the software. Nodes in an

event-flow graph represent events (not states); an edge from node x to node y would mean that event y can be executed after event x. The model contains other structures such as *integration trees* and *pre/postconditions* to generate state information on-demand, *e.g.*, for test oracle creation [26].

2. used abstractions such as detection of mutually exclusive events [31] to make the space manageable for testing.
3. defined an EDS test case as a sequence of events and developed search techniques to explore the event space for test case selection/generation, test coverage evaluation, and test oracle creation.

Much like how existing control-flow and data-flow models capture the control and data interactions in a program, the event-flow model captures event interactions in a GUI. Our event-based model has resulted in seminal work on testing certain classes of EDS and a strong foundation for extending the work. Using the event-flow model has enabled us to successfully develop and implement, as tools, several techniques to traverse the event space. They include a new *test case generation* technique that uses AI planning [25, 27–29], test *oracles* that check the state of the EDS during test case execution [20, 24, 26, 33], test *coverage criteria* based on events [31], a *regression tester* that repairs test cases that can no longer be executed on the software [21, 22, 30], a *smoke tester* for GUI-based applications [23, 32], a tool for automated protocol implementation robustness evaluation [41], and techniques for searching large client configuration spaces of web [5] and middleware software [13, 14, 17].

We posit that our work can be extended to develop a new framework for testing EDS. The core of this framework will be based on an *event-flow model* that we have developed for GUIs. The framework will combine the event-flow model with new techniques for automated test case generation, execution, coverage evaluation, and verification. At the workshop, we will outline the key components of such a framework.

References

[1] BAKSAAS, B. Computer-aided software testing. Dr. Dobb's Journal of Software Tools 19, 2 (Feb. 1994), 36, 38, 78.

[2] BERSTEL, J., REGHIZZI, S. C., ROUSSEL, G., AND SAN PIETRO, P. A scalable formal method for design and automatic checking of user interfaces. In Proceedings of the 23rd international conference on Software engineering (2001), IEEE Computer Society, pp. 453–462.

[3] CARLOGANU, A., AND RAGUIDEAU, J. Claire: An event-driven simulation tool for test and validation of software programs. In Proceedings of the 2002 International Conference on Dependable Systems and Networks (2002), IEEE Computer Society, p. 538.

[4] DWYER, M. B., CARR, V., AND HINES, L. Model checking graphical user interfaces using abstractions. In ESEC/FSE '97 (1997), M. Jazayeri and H. Schauer, Eds., vol. 1301 of Lecture Notes in Computer Science, Springer / ACM Press, pp. 244–261.

[5] EATON, C., AND MEMON, A. Improving browsing environment compliance evaluations for websites. In Proceedings of The International Workshop on Web Quality (WQ'04) (Munich, Germany, July 2004).

[6] EITER, T., SUBRAHMANIAN, V. S., AND PICK, G. Heterogeneous active agents, i: semantics. Artif. Intell. 108, 1-2 (1999), 179–255.

[7] GAFNI, V. Robots: a real-time systems architectural style. In Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering (1999), Springer-Verlag, pp. 57–74.

[8] HARROLD, M. J., OFFUT, A. J., AND TEWARY, K. An approach to fault modelling and fault seeding using the program dependence graph. Journal of Systems and Software 36, 3 (Mar. 1997), 273–296.

[9] HOLZMANN, G. J., AND SMITH, M. H. A practical method for verifying event-driven software. In Proceedings of the 21st international conference on Software engineering (1999), IEEE Computer Society Press, pp. 597–607.

[10] JACKSON, D. Exploiting symmetry in the model checking of relational specifications. Technical Report CS-94-219, Carnegie Mellon University, School of Computer Science, Dec. 1994.

[11] KAPFHAMMER, G., AND SOFFA, M. L. A family of test adequacy criteria for database-driven applications. In Proceedings of the 9th European Software Engineering Conference (ESEC) and 11th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-11) (Sept. 2003).

[12] KRANZLMULLER, D., GRABNER, S., AND VOLKERT, J. Event graph visualization for debugging large applications. In Proceedings of the SIGMETRICS symposium on Parallel and distributed tools (1996), ACM Press, pp. 108–117.

[13] KRISHNA, A. S., SCHMIDT, D. C., PORTER, A., MEMON, A., AND SEVILLA-RUIZ, D. Improving the Quality of Performance-intensive Software via Model-integrated Distributed Continuous Quality Assurance. In Proceedings of the 8th International Conference on Software Reuse (Madrid, Spain, July 2004), ACM/IEEE.

[14] KRISHNA, A. S., YILMAZ, C., MEMON, A., PORTER, A., AND SCHMIDT, D. C. A Distributed Continuous Quality Assurance Process to Manage Variability in Performance-intensive Software. Submitted to the International Journal of Software Process: Improvement and Practice special issue on Software Variability: Process and Management (2004).

[15] LEE, D., SABNANI, K. K., KRISTOL, D. K., PAUL, S., AND UYAR, M. U. Conformance testing of protocols specified as communicating FSMs. In Proceedings of the 12th Annual Joint Conference of the IEEE Computer and Communications Societies on Networking: Foundations for the Future. Volume 1 (Los Alamitos, CA, USA, Mar. 1993), M. G. Hluchyj, Ed., IEEE Computer Society Press, pp. 115–127.

[16] LEE, N. H., KIM, T. H., AND CHA, S. D. Construction of global finite state machine for testing task interactions written in message sequence charts. In Proceedings of the 14th international conference on Software engineering and knowledge engineering (2002), ACM Press, pp. 369–376.

[17] MEMON, A., PORTER, A., YILMAZ, C., NAGARAJAN, A., SCHMIDT, D. C., AND NATARAJAN, B. Skoll: Distributed Continuous Quality Assurance. In Proceedings of the 26th IEEE/ACM International Conference on Software Engineering (Edinburgh, Scotland, May 2004), IEEE/ACM.

[18] MEMON, A. M. A Comprehensive Framework for Testing Graphical User Interfaces. Ph.D. thesis, Department of Computer Science, University of Pittsburgh, July 2001.

[19] MEMON, A. M. GUI testing: Pitfalls and process. IEEE Computer 35, 8 (Aug. 2002), 90–91.

[20] MEMON, A. M. Advances in GUI testing. In Advances in Computers, ed. by Marvin V. Zelkowitz, vol. 57. Academic Press, 2003.

[21] MEMON, A. M. Automated GUI regression testing using AI planning. World Scientific Series in Machine Perception and Artificial Intelligence: Edited Volume on Artificial Intelligence Methods in Software Testing (2003).

[22] MEMON, A. M. Using tasks to automate regression testing of guis. In Proceedings of The IASTED International Conference on ARTIFICIAL INTELLIGENCE AND APPLICATIONS (AIA 2004) (Innsbruck, Austria, Feb. 2004).

[23] MEMON, A. M., BANERJEE, I., HASHMI, N., AND NAGARAJAN, A. DART: A framework for regression testing nightly/daily builds of GUI applications. In Proceedings of the International Conference on Software Maintenance 2003 (September 2003).

[24] MEMON, A. M., BANERJEE, I., AND NAGARAJAN, A. What test oracle should I use for effective GUI testing? In Proceedings of the IEEE International Conference on Automated Software Engineering (Oct.12–19 2003), IEEE Computer Society.

[25] MEMON, A. M., POLLACK, M. E., AND SOFFA, M. L. Using a goal-driven approach to generate test cases for GUIs. In Proceedings of the 21st International Conference on Software Engineering (May 1999), ACM Press, pp. 257–266.

[26] Memon, A. M., Pollack, M. E., and Soffa, M. L. Automated test oracles for GUIs. In Proceedings of the ACM SIGSOFT 8th International Symposium on the Foundations of Software Engineering (FSE-8) (NY, Nov. 8–10 2000), pp. 30–39.

[27] Memon, A. M., Pollack, M. E., and Soffa, M. L. Plan generation for GUI testing. In Proceedings of The Fifth International Conference on Artificial Intelligence Planning and Scheduling (Apr. 2000), AAAI Press, pp. 226–235.

[28] Memon, A. M., Pollack, M. E., and Soffa, M. L. A planning-based approach to GUI testing. In Proceedings of The 13th International Software/Internet Quality Week (May 2000).

[29] Memon, A. M., Pollack, M. E., and Soffa, M. L. Hierarchical GUI test case generation using automated planning. IEEE Transactions on Software Engineering 27, 2 (Feb. 2001), 144–155.

[30] Memon, A. M., and Soffa, M. L. Regression testing of GUIs. In Proceedings of the 9th European Software Engineering Conference (ESEC) and 11th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-11) (Sept. 2003).

[31] Memon, A. M., Soffa, M. L., and Pollack, M. E. Coverage criteria for GUI testing. In Proceedings of the 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9) (Sept. 2001), pp. 256–267.

[32] Memon, A. M., and Xie, Q. Empirical evaluation of the fault-detection effectiveness of smoke regression test cases for gui-based software. In Proceedings of The International Conference on Software Maintenance 2004 (ICSM'04) (Chicago Illinois, USA, Sept. 2004).

[33] Memon, A. M., and Xie, Q. 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) (Linz, Austria, Sept. 2004).

[34] Ostrand, T., Anodide, A., Foster, H., and Goradia, T. A visual test development environment for GUI systems. In Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA-98) (New York, Mar.2–5 1998), ACM Press, pp. 82–92.

[35] Ricca, and Tonella. Building a tool for the analysis and testing of web applications: Problems and solutions. In TACAS: International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, LNCS (2001).

[36] Schmidt, D. C., and Buschmann, F. Patterns, frameworks, and middleware: their synergistic relationships. In Proceedings of the 25th international conference on Software engineering (2003), IEEE Computer Society, pp. 694–704.

[37] Selvakumar, M. Automated testing for Web applications. Dr. Dobb's Journal of Software Tools 24, 5 (May 1999), 88, 90, 92, 95–96.

[38] Shankar, A. U. Verified data transfer protocols with variable flow control. ACM Trans. Comput. Syst. 7, 3 (1989), 281–316.

[39] Shehady, R. K., and Siewiorek, D. P. A method to automate user interface testing using variable finite state machines. In Proceedings of The Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing (FTCS'97) (Washington - Brussels - Tokyo, June 1997), IEEE Press, pp. 80–88.

[40] Sliwa, C. Event-driven architecture poised for wide adoption. COMPUTERWORLD (May 2003).

[41] Vasan, A., and Memon, A. M. Aspire: Automated systematic protocol implementation robustness evaluation. In Proceedings of The Australian Software Engineering Conference (ASWEC 2004) (Melbourne, Australia, Apr. 2004).

[42] Welsh, M., Culler, D., and Brewer, E. Seda: an architecture for well-conditioned, scalable internet services. In Proceedings of the eighteenth ACM symposium on Operating systems principles (2001), ACM Press, pp. 230–243.