Mining Modal Scenarios from Execution Traces

(poster abstract)

David Lo
National University of Singapore
dlo@comp.nus.edu.sg

Shahar Maoz
The Weizmann Institute of Science, Israel
shahar.maoz@weizmann.ac.il

Siau-Cheng Khoo
National University of Singapore
khoosc@comp.nus.edu.sg

Abstract

Specification mining is a dynamic analysis process aimed at automatically inferring suggested specifications of a program from its execution traces. We describe a method, a framework, and a tool, for mining inter-object scenario-based specifications in the form of a UML2-compliant variant of Damm and Harel's Live Sequence Charts (LSC), which extends the classical partial order semantics of sequence diagrams with temporal liveness and symbolic class level lifelines, in order to generate compact and expressive specifications. Moreover, we use previous research work and tools developed for LSC to visualize, analyze, manipulate, test, and thus evaluate the scenario-based specifications we mine. Our mining framework is supported by statistically sound metrics. We demonstrate and evaluate our work using a case study.

Categories and Subject Descriptors: D.2.1[Software Engineering]: Requirements/Specifications–Tools; D.2.7[Soft–ware Engineering]: Distribution, Maintenance,and Enhance–ment–Restructuring, reverse engineering and reengineering
General Terms: Algorithms, Design, Experimentation
Keywords: Specification Mining, UML Sequence Diagrams, Live Sequence Charts

1. Introduction

Analyzing the behavior of software systems, in order to aid program comprehension, reduce their maintenance costs, and improve their quality, is a complex and challenging task. Having incorrect, incomplete, or outdated documented specifications, as a result of short time-to-market constraints, changing requirements, and poorly managed product evolution, reduces comprehension of the code base, increases maintenance costs, and adds challenges towards verification of their correctness. One approach to address this challenge is to automatically infer specifications of a system from its execution traces by a dynamic analysis process referred to as *specification mining* (see, e.g., [2, 9]).

In this work we focus on mining specifications of reactive systems, discrete event systems which maintain ongoing interaction with their environment, and on their behavioral specification using inter-object scenarios. Scenarios, depicted using variants of sequence diagrams, are a popular means to specify the inter-object behavior of systems (see, e.g., [5]), are included in the UML standard, and are supported by many modeling tools. In particular, we are interested in modal scenarios presented using a UML2 compliant variant of Damm and Harel's Live Sequence Charts (LSC) [3, 7], which extends the partial order semantics of sequence diagrams with universal and existential modalities and allows symbolic class level lifelines, resulting in compact and expressive specifications.

2. Modal Scenarios & Mining Framework

We use here a restricted subset of the LSC language. An LSC includes a set of instance lifelines, representing system's objects, and is divided into two parts, the *pre-chart* ('cold' fragment) and the *main-chart* ('hot' fragment), each specifying an ordered set of method calls between the objects represented by the instance lifelines. A universal LSC specifies a *universal liveness requirement*: for all runs of the system, and for every point during such a run, whenever the sequence of events defined by the pre-chart occurs (in the specified order), eventually the sequence of events defined by the main-chart must occur (in the specified order). Events not explicitly mentioned in the diagram are not restricted in any way to appear or not to appear during the run (including between the events that are mentioned in the diagram).

Syntactically, instance lifelines are drawn as vertical lines, pre-chart (main-chart) events are colored in blue (red) and drawn using a dashed (solid) line. LSCs can be edited and visualized within standard UML2 compliant modeling tools (e.g., IBM RSA) using the *modal* profile [7].

The input for the mining algorithm are finite traces consisting of events, where each event corresponds to a triplet: caller object identifier, callee object identifier, and method signature. To relate between LSCs and execution traces we introduce notions of positive and negative witnesses.

We consider traces to be finite words over a finite alphabet of events $\Sigma = \{a, b, c...\}$, where a unique letter corresponds to a unique triplet. We use the symbol $+\!\!+$ to represent the concatenation operator between finite words. For two words w, u we denote the projection of w onto the alphabet of events appearing in u by w_u. A *positive witness* of an LSC $M(pre, main)$ with respect to a trace T is defined as a *minimal* subword s of T such that $s_m = pre{+\!\!+}main$. Note that T may include many positive-witnesses of M. A *negative-witness* of an LSC $M(pre, main)$ with regard to a trace T, is a positive-witness of the word pre that cannot be extended to a positive-witness of M.

Given a trace, for an LSC $M(pre, main)$ we measure its statistical significance using two metrics (adopted from data mining [4]): (1) *support* – the number of positive witnesses of $pre{+\!\!+}main$ in the trace, and (2) *confidence* – the likelihood of the pre being followed by the $main$ in the trace (which can be found from the number of positive and negative witnesses of pre and $pre{+\!\!+}main$). Thus, given a trace and user-defined thresholds for minimum support and confidence, our algorithm finds *a sound and complete set* of statistically significant modal scenarios.

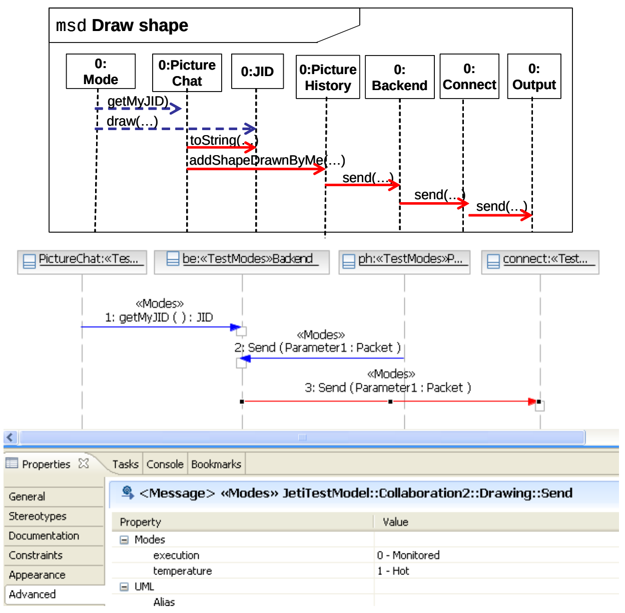
The mined set of LSCs is post-processed to identify class level LSCs (see LSC *symbolic instances* [11]). In addition, we provide an array of additional user-guided filters and abstractions, such as removing logically redundant LSCs, ensuring connectivity, and limiting the length of mined LSCs, to further refine the resulting set of mined scenarios.

3. Case Study, Conclusion & Future Work

To demonstrate our work we used AspectJ to instrument Jeti [1], a popular full featured open source instant messaging application, and created trace files of recorded interactions between several Jeti clients, each of which is approximately 1K events long. Many informative LSCs were mined. An example is highlighted below.

From traces involving the use of Jeti's group whiteboard, the miner has captured a scenario of drawing a line and sending it to the other chat users. In Jeti, different graphic elements (`LineMode`, `EllipseMode`, `RectangleMode`, etc.) are all sub-classes of the abstract class `Mode`. Interestingly, the results included additional very similar LSCs corresponding to drawing of ellipses and rectangles. Indeed, the only difference between these LSCs was the participating classes of the first leftmost lifelines. We thus performed a super-class aggregation resulting in the LSC shown in Fig. 1 (Top). Note the abstract class `Mode` referenced on the leftmost lifeline. This mined LSC takes advantage of the semantics of LSC symbolic instances in defining compact and expressive scenarios.

We have implemented a programmatic translation of the mined LSCs (in textual format) into UML2 Sequence Diagrams extended with the *modal* profile [7], using the Eclipse UML2 APIs. This allows the visualization and manipulation of LSCs inside IBM Rational Software Architect (RSA)



<Message> «Modes» JetiTestModel::Collaboration2::Drawing::Send

Property	Value
Modes	
execution	0 - Monitored
temperature	1 - Hot
UML	
Alias	

Figure 1. (Top) A mined LSC: Drawing a general shape (`Mode`); (Bottom) A mined LSC inside IBM RSA.

(Fig. 1 (Bottom)). Finally, we used the S2A compiler [6], developed at the Weizmann Institute of Science, to programmatically compile selected LSCs into (monitoring) *scenario aspects* [10]. These served as scenario-based tests for Jeti and allowed us to 'validate' selected mined LSCs during subsequent executions.

In this extended abstract we have proposed a novel method to mine a sound and complete set of statistically significant modal scenarios from program execution traces. The framework exploits the unique features of LSCs – universal liveness, class-level lifelines – and existing related tools – IBM RSA, the S2A compiler – to improve the usefulness of the mined specifications. The case study demonstrates the utility of our approach. Our current method is limited to mining of total order LSCs. In the future, we plan to mine for additional features of sequence diagrams in general, such as explicit partial order, various structural constructs (alternatives, loops, etc.), and functional state invariants.

Acknowledgement We thank David Harel for his valuable advice.

References

[1] Jeti. Version 0.7.6 (Oct. 2006). http://jeti.sourceforge.net/.
[2] G. Ammons, R. Bodik, and J. R. Larus. Mining specification. In *POPL*, 2002.
[3] W. Damm and D. Harel. LSCs: Breathing Life into Message Sequence Charts. *J. on Formal Methods in System Design*, 19(1):45–80, 2001.
[4] J. Han and M. Kamber. *Data Mining Concepts and Techniques, 2nd Ed.* Morgan Kaufmann, 2006.
[5] D. Harel. From play-in scenarios to code: An achievable dream. *IEEE Computer*, 34(1):53–60, 2001.
[6] D. Harel, A. Kleinbort, and S. Maoz. S2A: A compiler for multi-modal UML sequence diagrams. In *FASE*, 2007.
[7] D. Harel and S. Maoz. Assert and negate revisited: Modal semantics for UML sequence diagrams. *Software and System Modeling*, 2007.
[8] J. Klose, T. Toben, B. Westphal, and H. Wittke. Check it out: On the efficient formal verification of Live Sequence Charts. In *CAV*, 2006.
[9] D. Lo and S.-C. Khoo. SMArTIC: Towards building an accurate, robust and scalable specification miner. In *SIGSOFT FSE*, 2006.
[10] S. Maoz and D. Harel. From multi-modal scenarios to code: compiling LSCs into AspectJ. In *SIGSOFT FSE*, 2006.
[11] R. Marelly, D. Harel, and H. Kugler. Multiple Instances and Symbolic Variables in Executable Sequence Charts. In *OOPSLA*, 2002.