Towards a Self-Forensics Property in the ASSL Toolset

Serguei A. Mokhov  
Computer Science and  
Software Engineering  
Concordia University  
Montreal, QC, Canada  
mokhov@cse.concordia.ca

Emil Vassev  
School of Computer Science  
and Informatics  
University College Dublin  
Dublin, Ireland  
emil.vassev@ucd.ie

Joey Paquet  
Computer Science and  
Software Engineering  
Concordia University  
Montreal, QC, Canada  
paquet@cse.concordia.ca

Mourad Debbabi  
Concordia Institute for  
Information Systems  
Engineering  
Concordia University  
Montreal, QC, Canada  
debbabi@ciise.concordia.ca

ABSTRACT

This preliminary conceptual work discusses a notion of self-forensics as an autonomic property to augment the Autonomic System Specification Language (ASSL) framework of formal specification tools for autonomic systems. The core of the proposed methodology leverages existing designs, theoretical results, and implementing systems to enable rapid completion of and validation of the experiments and their results initiated in this work. Specifically, we leverage the ASSL toolkit to add the self-forensics autonomic property (SFAP) to enable generation of the Java-based Object-Oriented Intensional Programming (JOOIP) language code laced with traces of Forensic Lucid to encode contextual forensic evidence and other expressions.

Categories and Subject Descriptors

D.3.2 [Programming Languages]: Language Classifications—Very high-level languages; Multiparadigm languages;  
D.3.4 [Programming Languages]: Processors—Compilers; Preprocessors; Run-time environments; 1.2.2 [Artificial Intelligence]: Automatic Programming—Program synthesis; Program transformation; Forensic computing;  
D.2.11 [Software Architectures]: Domain-specific architectures; Languages

General Terms

Languages, Theory, Design

Keywords

self-forensics, Forensic Lucid, JOOIP, ASSL, forensic computing, autonomic computing, GIPSY

1. INTRODUCTION

1.1 Problem and Proposed Solution

The novel concept of self-forensics and the idea of its implementation within ASSL and GIPSY is described through their founding core works. These preliminary findings and discussions are currently at the conceptual level, but the authors are confident to provide a concrete formal model, the complete requirements, design, and implementation of the concept described here by leveraging the resources provided by the previous research work. To the authors’ knowledge there is no preceding work other than the authors’ own that does attempt something similar to what is described here.

1.2 Organization

First, we give a glimpse overview of the founding background work on ASSL and self-forensics in Section 2.1 and Section 2.2. Then, we describe the core principles and ideas of the methodology of realization of the self-forensics autonomic property (SFAP) within the ASSL framework in Section 3. We provide a quick notion of the syntactical notation of SFAP and where it fits within the generating toolset of ASSL and the run-time environment of the General Intensional Programming System (GIPSY). We conclude in Section 4 for the merits and the future endeavors for the developments in this direction.

2. BACKGROUND

2.1 ASSL Formal Specification Toolset

The ASSL framework [44, 39, 32] takes as an input a specification of properties of autonomic systems [6, 7, 9, 8, 1, 26, 22], does formal syntax and semantics checks of the specifications, and if the checks pass, it generates a Java collection of classes and interfaces corresponding to the specification. Subsequently, a developer has to fill in some overridden interface methods corresponding to the desired autonomic policies in a proxy implementation within the generated Java skeleton application or map them to the existing legacy application [44, 39, 32].

The ASSL framework [44] includes the autonomic multi-
Figure 1: ASSL Multi-Tier Model

3. SELF-FORENSICS AUTONOMIC PROPERTY (SFAP)

First, we add a notion of a _Self_Forensics_ policy specification for AS tier and AE, just like it is done for the self-CHOP properties. The property introduction consists of two major parts: (1) adding the syntax and semantical support to the lexical analyzer, parser, and semantic checker of ASSL as well as (2) adding the appropriate code generator for JOOIP and Forensic Lucid to translate forensic events. The JOOIP code is mostly Java with embedded fragments of Forensic Lucid-encoded evidence [10, 21].

We use ASSL’s managed-element (ME) specification of AE to encode any module or subsystem of any software system under study to increase or reduce the amount of forensic evidence logged as Forensic Lucid events depending on the criticality of faults (that can be expressed as ASSL metrics).

A very high-level example of the generic self-forensic specification is in Figure 4. Many details are presently omitted due to the preliminary work on this novel concept and will be provided in our subsequent publication.

Wu and the GIPSY team came up with a hybrid intentional OO language, JOOIP [47, 46], to allow mixing Java and Lucid code by placing Lucid fragments nearly anywhere within Java classes (as data members or within methods. As a part of this conceptual research work, we propose that the ASSL toolset in this instance be augmented with a code-generation plug-in that generates JOOIP code laced with Forensic Lucid contextual expressions for forensic analysis. The evaluation of the JOOIP+Forensic Lucid code further is to be performed by the GIPSY’s general eduction engine (GEE), described in detail elsewhere [24, 4, 15].

Furthermore, in this proposed prototype the EVENTS members would be the basic building blocks of the contextual specification of the Forensic Lucid observation sequences. The INITIATED_BY and TERMINATED_BY clauses would corre-
Figure 3: Forensic Lucid Compilation and Evaluation Flow in GIPSY
spond to the beginning and end of data stream Lucid operators bod and eod. ASSL fluents would map to the Lucid streams of the observation sequences where each stream is a witness account of systems behavior. All fluents constitute an evidential statement. The mapping and actions correspond to the handling of the anomalous states within the JOOIP’s Java code.

Once JOOIP code with Forensic Lucid fragments is generated by the ASSL toolset, it is passed on to the hybrid compiler of GIPSY, the GIPC to properly compile the JOOIP and Forensic Lucid specifications, link them together in an executable code inside the GEE engine resources (GEER), which then would have three choices of evaluation of it—the traditional deduction model of GEE, AspectJ-based deduction model, and probabilistic model checking with the PRISM backend.

4. CONCLUSION

We laid out some preliminary groundwork of requirements to implement formally the self-forensics autonomic property within the ASSL toolset in order to allow any implementation of the self-forensics property added to the legacy small-to-medium open-source and academic software systems.

Our future work will be to complete the implementation of the said property and export it onto the target example software systems of ADMARF, AGIPSY [43], and others described conceptually in [21].

We will investigate the use of the open-source PRISM tool [29], for probabilistic model checking of the produced Forensic Lucid specifications as Forensic Lucid forensic case specification models include credibility and trustworthiness factors of the evidence and witnesses based on the Dempster-Shafer mathematical theory of evidence [11, 3, 27] into the ASSL specifications.

5. REFERENCES

Figure 4: The Prototype Syntactical Specification of the SELF_FORENSICS in ASSL for ADMARF


