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 the 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*; I.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-

```
I. Autonomic System (AS)
 * AS Service-level Objectives
 * AS Self-managing Policies
 * AS Architecture
 * AS Actions
 * AS Events
 * AS Metrics
II. AS Interaction Protocol (ASIP)
 * AS Messages
 * AS Communication Channels
 * AS Communication Functions
III. Autonomic Element (AE)
 * AE Service-level Objectives
 * AE Self-managing Policies
 * AE Friends
 * AE Interaction Protocol (AEIP)
   - AE Messages
   - AE Communication Channels
   - AE Communication Functions
   - AE Managed Elements
 * AE Recovery Protocol
 * AE Behavior Models
 * AE Outcomes
 * AE Actions
 * AE Events
 * AE Metrics
```

Figure 1: ASSL Multi-Tier Model

tier system architecture (AS) including formal language constructs to specify service-level objectives (SLOs), core self-CHOP (i.e. self-configuration, self-healing, self-optimization, and self-protection) autonomic properties, corresponding architecture, allowed actions, events, and metrics to aid the self-management aspect of the system. It also specifies the interaction protocols between the AS' managed autonomic elements, including specification of of messages exchanged and how they are communicated. Finally, it provides for specification of the autonomic element (AE) architecture, like for the whole system, each element is a subject to the SLOs, self-CHOP policies, behavior, actions, metrics, and interaction protocols, the summary of all of which is enumerated in Figure 1.

ASSL formal modeling, specification, and model checking [36, 35] has been applied to a number open-source, academic, and research software system specifications, e.g. such as Voyager imagery processing [34], the Distributed Modular Audio Recognition Framework (DMARF) [41, 20, 40], and the General Intensional Programming System (GIPSY) [43] reliability of self-assessment, distributed, and other autonomic aspects of the autonomic system-time reactive model (AS-TRM) [38, 42], self-adapting properties of NASA swarm missions [31, 5, 37] and others [33].

2.2 Self-Forensics Concept

The study of self-forensics [12, 21, 13], is an additional property one of the authors is investigating throughout his ongoing PhD thesis work with the contextual forensic logging with Forensic Lucid and case specification [14, 16, 10, 18, 17]. Forensic Lucid is an intensional context-oriented forensic case specification, modeling, and evaluation language. Forensic Lucid was initially proposed for specification and automatic deduction and event reconstruction in the cybercrime domain of digital forensics [23]. It has been proposed to extend its use onto other domains such as investigation of incidents in various vehicle crash investigations, and autonomous software and hardware systems. Its primary feature inherited from the Lucid family of languages is to be able to specify and work with context [45, 25, 30] as a first-class value, and the context represents the evidence and stories told by witnesses.

Forensic Lucid's primary experimental platform for compilation (Forensic Lucid compiler is a member of the General Intensional Programming Compiler (GIPC) framework) and evaluation is the General Intensional Programming System (GIPSY) [24, 4, 15]. GIPSY's run-time system, the General Eduction Engine (GEE), is designed to be flexible to allow various modes of execution, including the planned used of the evaluation by the PRISM- [29] and AspectJ-based [2] backends as illustrated in Figure 2 [19] and Figure 3.

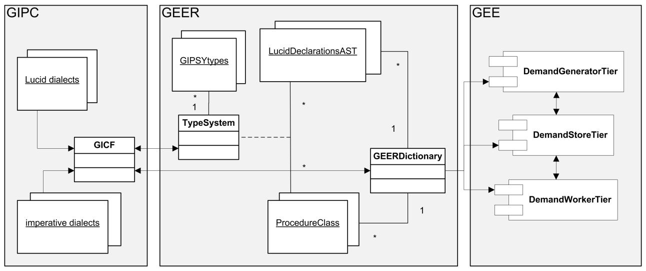

Figure 2: GIPSY High Level Overview

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 intensional 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 [47, 46] 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-

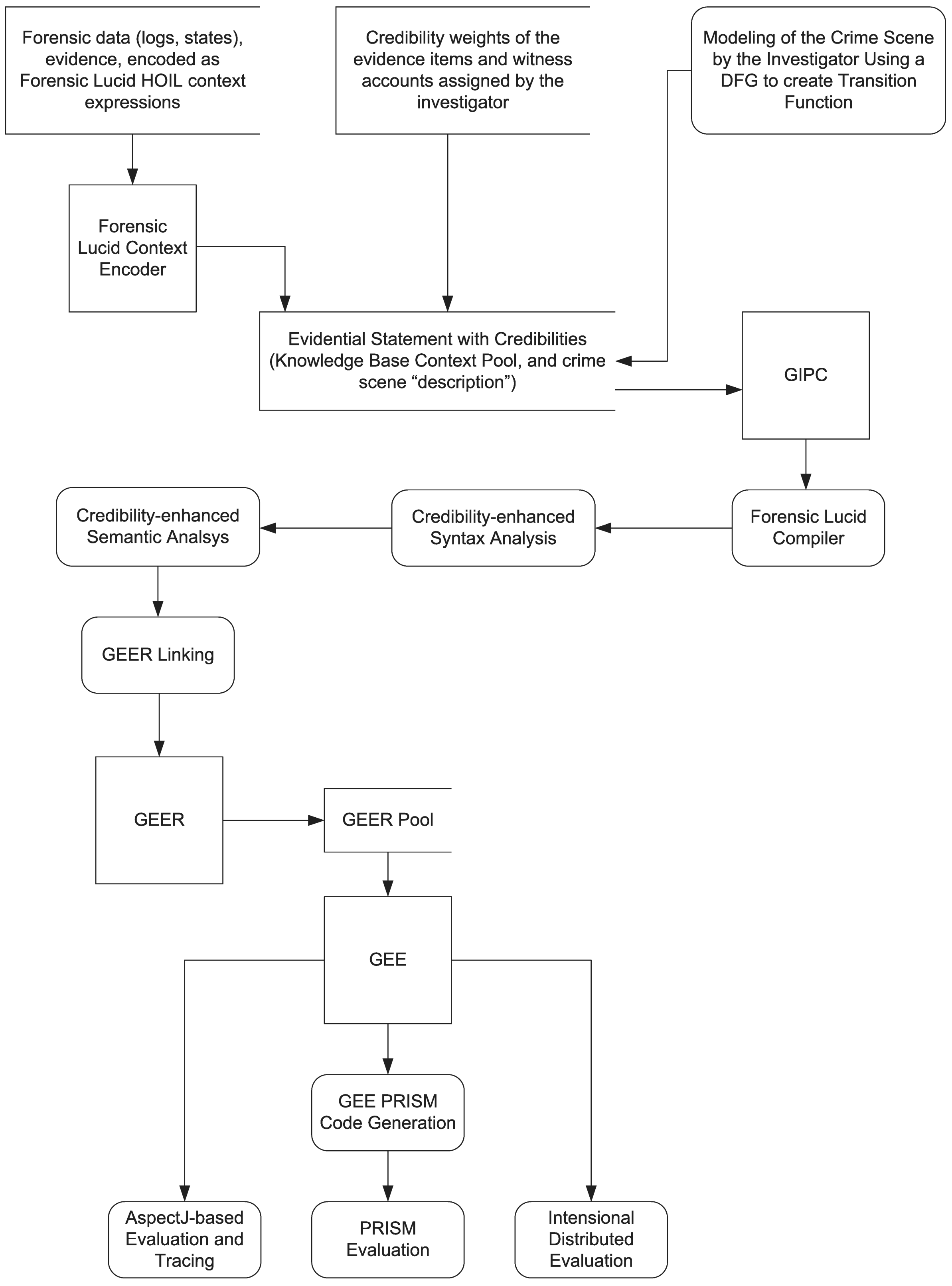
Forensic data (logs, states), evidence, encoded as Forensic Lucid HOIL context expressions
Credibility weights of the evidence items and witness accounts assigned by the investigator
Modeling of the Crime Scene by the Investigator Using a DFG to create Transition Function
Forensic Lucid Context Encoder
Evidential Statement with Credibilities (Knowledge Base Context Pool, and crime scene "description")
GIPC
Forensic Lucid Compiler
Credibility-enhanced Syntax Analysis
Credibility-enhanced Semantic Analsys
GEER Linking
GEER
GEER Pool
GEE
GEE PRISM Code Generation
AspectJ-based Evaluation and Tracing
PRISM Evaluation
Intensional Distributed Evaluation

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 a executable code inside the GEE engine resources (GEER), which then would have three choices of evaluation of it – the traditional eduction model of GEE, AspectJ-based eduction 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

[1] D. Agrawal et al. Autonomic computing expressing language. Technical report, IBM Corporation, 2005.

[2] AspectJ Contributors. *AspectJ: Crosscutting Objects for Better Modularity.* eclipse.org, 2007. `http://www.eclipse.org/aspectj/`.

[3] R. Haenni, J. Kohlas, and N. Lehmann. Probabilistic argumentation systems. Technical report, Institute of Informatics, University of Fribourg, Fribourg, Switzerland, Oct. 1999.

[4] B. Han, S. A. Mokhov, and J. Paquet. Advances in the design and implementation of a multi-tier architecture in the GIPSY environment with Java. In *Proceedings of SERA 2010.* IEEE Computer Society, 2010. To appear; online at `http://arxiv.org/abs/0906.4837`.

[5] M. G. Hinchey, J. L. Rash, W. Truszkowski, C. Rouff, and R. Sterritt. Autonomous and autonomic swarms. In *Software Engineering Research and Practice*, pages 36–44. CSREA Press, 2005.

[6] P. Horn. Autonomic computing: IBM's perspective on the state of information technology. Technical report, IBM T. J. Watson Laboratory, Oct. 2001.

[7] IBM Corporation. An architectural blueprint for autonomic computing. Technical report, IBM Corporation, 2006.

[8] IBM Tivoli. Autonomic computing policy language. Technical report, IBM Corporation, 2005.

[9] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[10] S. A. Mokhov. Encoding forensic multimedia evidence from MARF applications as Forensic Lucid expressions. In T. Sobh, K. Elleithy, and A. Mahmood, editors, *Novel Algorithms and Techniques in Telecommunications and Networking, proceedings of CISSE'08*, pages 413–416, University of Bridgeport, CT, USA, Dec. 2008. Springer. Printed in January 2010.

[11] S. A. Mokhov. Enhancing the formal cyberforensic approach with observation modeling with credibility factors and mathematical theory of evidence. [online], also in *;login: vol. 34, no. 6, p. 101*, Dec. 2009. Presented at WIPS at USENIX Security'09, `http://www.usenix.org/events/sec09/wips.html`.

[12] S. A. Mokhov. The role of self-forensics modeling for vehicle crash investigations and event reconstruction simulation. In *Proceedings of HSC'09.* SCS, Oct. 2009. To appear, online at `http://arxiv.org/abs/0905.2449`.

[13] S. A. Mokhov. Towards improving validation, verification, crash investigations, and event reconstruction of flight-critical systems with self-forensics. [online], June 2009. A white paper submitted in response to NASA's RFI NNH09ZEA001L, `http://arxiv.org/abs/0906.1845`.

[14] S. A. Mokhov and J. Paquet. Formally specifying and proving operational aspects of Forensic Lucid in Isabelle. Technical Report 2008-1-Ait Mohamed, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, Aug. 2008. In Theorem Proving in Higher Order Logics (TPHOLs2008): Emerging Trends Proceedings.

[15] S. A. Mokhov and J. Paquet. Using the General Intensional Programming System (GIPSY) for evaluation of higher-order intensional logic (HOIL) expressions. In *Proceedings of SERA 2010.* IEEE Computer Society, 2010. To appear; online at `http://arxiv.org/abs/0906.3911`.

[16] S. A. Mokhov, J. Paquet, and M. Debbabi. Formally specifying operational semantics and language constructs of Forensic Lucid. In O. Göbel, S. Frings, D. Günther, J. Nedon, and D. Schadt, editors, *Proceedings of the IT Incident Management and IT Forensics (IMF'08)*, pages 197–216, Mannheim, Germany, Sept. 2008. GI. LNI140.

[17] S. A. Mokhov, J. Paquet, and M. Debbabi. Reasoning about a simulated printer case investigation with Forensic Lucid. In *Proceedings of the Huntsville Simulation Conference (HSC'09).* SCS, Oct. 2009. To appear, online at `http://arxiv.org/abs/0906.5181`.

[18] S. A. Mokhov, J. Paquet, and M. Debbabi. Towards automated deduction in blackmail case analysis with Forensic Lucid. In *Proceedings of the Huntsville Simulation Conference (HSC'09).* SCS, Oct. 2009. To

```
AS ADMARF {

    TYPES { MonitoredElement }

    ASSELF_MANAGEMENT {
        SELF_FORENSICS {
            FLUENT inIntensiveForensicLogging {
                INITIATED_BY { EVENTS.anomalyDetected }
                TERMINATED_BY {
                    EVENTS.anomalyResolved,
                    EVENTS.anomalyFailedToResolve
                }
            }

            MAPPING {
                CONDITIONS { inIntensiveForensicLogging }
                DO_ACTIONS { ACTIONS.startForensicLogging }
            }
        }
    }

    ACTIONS {
        ACTION startForensicLogging {
            GUARDS { ASSELF_MANAGEMENT.SELF_FORENSICS.inIntensiveForensicLogging }
              VARS { Boolean ... }
              DOES {
                  ...
                  FOREACH member in AES {
                    ...
                  };
              }
              ONERR_DOES {
                  // if error then log it too
                  ...
              }
        }
    } // ACTIONS

    EVENTS { // these events are used in the fluents specification
        EVENT anomalyDetected {
            ACTIVATION { SENT { ASIP.MESSAGES.... } }
        }
        ...
    } // EVENTS

    METRICS {
        METRIC thereIsInsecurePublicMessage {
            METRIC_TYPE { CREDIBILITY }
            DESCRIPTION { "sets event's trustworthiness/credibility AE" }
            VALUE { ... }
            ...
        }
    }
} // AS ADMARF

// ...

MANAGED_ELEMENTS
{
   MANAGED_ELEMENT STAGE_ME
   {
       INTERFACE_FUNCTION logForensicEvent
       {
           PARAMETERS { ForensicLucidEvent poEvent }
           RETURNS { Boolean }
       }
   }
}
```

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

appear, online at `http://arxiv.org/abs/0906.0049`.

[19] S. A. Mokhov, J. Paquet, and X. Tong. A type system for hybrid intensional-imperative programming support in GIPSY. In *Proceedings of C3S2E'09*, pages 101–107, New York, NY, USA, May 2009. ACM.

[20] S. A. Mokhov and E. Vassev. Autonomic specification of self-protection for Distributed MARF with ASSL. In *Proceedings of C3S2E'09*, pages 175–183, New York, NY, USA, May 2009. ACM.

[21] S. A. Mokhov and E. Vassev. Self-forensics through case studies of small to medium software systems. In *Proceedings of IMF'09*, pages 128–141. IEEE Computer Society, Sept. 2009.

[22] R. Murch. *Autonomic Computing: On Demand Series.* IBM Press, Prentice Hall, 2004.

[23] G. Palmer (Editor). A road map for digital forensic research, report from first digital forensic research workshop (DFRWS). Technical report, DFRWS, 2001.

[24] J. Paquet. Distributed eductive execution of hybrid intensional programs. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09)*, pages 218–224, Seattle, Washington, USA, July 2009. IEEE Computer Society.

[25] J. Paquet, S. A. Mokhov, and X. Tong. Design and implementation of context calculus in the GIPSY environment. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, pages 1278–1283, Turku, Finland, July 2008. IEEE Computer Society.

[26] M. Parashar and S. Hariri, editors. *Autonomic Computing: Concepts, Infrastructure and Applications*. CRC Press, Dec. 2006.

[27] G. Shafer. *The Mathematical Theory of Evidence.* Princeton University Press, 1976.

[28] B. Shishkov, J. Cordeiro, and A. Ranchordas, editors. *ICSOFT 2009 - Proceedings of the 4th International Conference on Software and Data Technologies*, volume 1. INSTICC Press, July 2009.

[29] The PRISM Team. PRISM: a probabilistic model checker. [online], 2004–2010. `http://www.prismmodelchecker.org/`, last viewed June 2009.

[30] X. Tong. Design and implementation of context calculus in the GIPSY. Master's thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, Apr. 2008.

[31] W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff. NASA's swarm missions: The challenge of building autonomous software. *IT Professional*, 6(5):47–52, 2004.

[32] E. Vassev. *ASSL: Autonomic System Specification Language – A Framework for Specification and Code Generation of Autonomic Systems.* LAP Lambert Academic Publishing, Nov. 2009. ISBN: 3-838-31383-6.

[33] E. Vassev and M. Hinchey. Assl: A software engineering approach to autonomic computing. *IEEE Computer*, 42(6):90–93, 2009.

[34] E. Vassev and M. Hinchey. ASSL specification model for the image-processing behavior in the NASA Voyager mission. Technical report, Lero - The Irish Software Engineering Research Center, 2009.

[35] E. Vassev, M. Hinchey, and A. J. Quigley. A self-adaptive architecture for autonomic systems developed with ASSL. In Shishkov et al. [28], pages 163–168.

[36] E. Vassev, M. Hinchey, and A. J. Quigley. Towards model checking with Java PathFinder for autonomic systems specified and generated with ASSL. In Shishkov et al. [28], pages 251–256.

[37] E. Vassev, M. G. Hinchey, and J. Paquet. Towards an ASSL specification model for NASA swarm-based exploration missions. In *Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC 2008) - AC Track*, pages 1652–1657. ACM, 2008.

[38] E. Vassev, H. Kuang, O. Ormandjieva, and J. Paquet. Reactive, distributed and autonomic computing aspects of AS-TRM. In J. Filipe, B. Shishkov, and M. Helfert, editors, *ICSOFT (1)*, pages 196–202. INSTICC Press, Sept. 2006.

[39] E. Vassev and S. A. Mokhov. An ASSL-generated architecture for autonomic systems. In *Proceedings of C3S2E'09*, pages 121–126, New York, NY, USA, May 2009. ACM.

[40] E. Vassev and S. A. Mokhov. Self-optimization property in autonomic specification of Distributed MARF with ASSL. In B. Shishkov, J. Cordeiro, and A. Ranchordas, editors, *Proceedings of ICSOFT'09*, volume 1, pages 331–335, Sofia, Bulgaria, July 2009. INSTICC Press.

[41] E. Vassev and S. A. Mokhov. Towards autonomic specification of Distributed MARF with ASSL: Self-healing. In *Proceedings of SERA 2010*. IEEE Computer Society, 2010. To appear.

[42] E. Vassev, O. Ormandjieva, and J. Paquet. ASSL specification of reliability self-assessment in the AS-TRM. In J. Filipe, B. Shishkov, and M. Helfert, editors, *ICSOFT (SE)*, volume SE, pages 198–206. INSTICC Press, July 2007.

[43] E. Vassev and J. Paquet. Towards autonomic GIPSY. In *Proceedings of the Fifth IEEE Workshop on Engineering of Autonomic and Autonomous Systems (EASE 2008)*, pages 25–34. IEEE Computer Society, 2008.

[44] E. I. Vassev. *Towards a Framework for Specification and Code Generation of Autonomic Systems.* PhD thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2008.

[45] K. Wan. *Lucx: Lucid Enriched with Context.* PhD thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2006.

[46] A. Wu, J. Paquet, and S. A. Mokhov. Object-oriented intensional programming: Intensional Java/Lucid classes. In *Proceedings of SERA 2010*. IEEE Computer Society, 2010. To appear; online at: `http://arxiv.org/abs/0909.0764`.

[47] A. H. Wu. *OO-IP Hybrid Language Design and a Framework Approach to the GIPC.* PhD thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, 2009.