SELF-OPTIMIZATION PROPERTY IN AUTONOMIC SPECIFICATION OF DISTRIBUTED MARF WITH ASSL

Emil Vassev[1] and Serguei A. Mokhov[2]
[1]*Lero - The Irish Software Engineering Research Center, University College Dublin, Ireland*
[2]*Computer Science and Software Engineering, Concordia University, Montreal, Quebec, Canada*

Keywords: Self-optimization, Algorithms, Software engineering, DMARF, ASSL.

Abstract: In this work, we venture out to develop self-optimization features in the Distributed Modular Audio Recognition Framework (DMARF). Here, we use the Autonomic System Specification Language (ASSL) to specify a self-optimization policy and generate the code for the same. This completes the first iteration of the autonomic specification layer for DMARF and enables re-engineered autonomic DMARF system, which also includes self-healing and self-protection, both developed earlier.

1 INTRODUCTION

We use the Autonomic System Specification Language (ASSL) (Vassev, 2008) to integrate a self-optimizing autonomic property into the Distributed Modular Audio Recognition Framework (DMARF) – an intrinsically complex system composed of multi-level operational layers. This work complements our related work on the self-protecting and self-healing properties for the system.

Problem Statement. Distributed MARF (DMARF) cannot be used in autonomous, partly or fully unattended environments due to the lack of design provision for such a use by applications that necessitate autonomic self-adapting requirements, such as self-optimization. Extending DMARF to support those requirements sustains a major development effort for an open-source project.

Proposed Solution. We provide an initial proof-of-concept ASSL specification of one of the three autonomic requirements for DMARF – self-optimization. Note that the other two, termed self-healing and self-protection, were defined in the course of this project (Mokhov and Vassev, 2009). Having the ASSL specification completed would allow for the automatic Java code generation of a special wrapper application providing an autonomic layer to DMARF to fulfill the stated autonomic requirements.

2 BACKGROUND

The vision and metaphor of autonomic computing (AC) (Murch, 2004) is to apply the principles of self-regulation and complexity hiding to software and hardware. The AC paradigm emphasizes the reduction of the workload needed to maintain complex systems by transforming them into self-managing autonomic systems. Today, a great deal of research effort is devoted to developing AC development tools. Such a tool is the ASSL framework, which helps AC researchers with problem formation, specification, system design, analysis and evaluation, and eventual implementation.

2.1 Distributed MARF

The classic Modular Audio Recognition Framework (MARF) (Mokhov, 2008) is an open-source research platform and a collection of various algorithm implementations for pattern recognition, signal processing, natural language processing (NLP), etc. written in Java. It is purposefully arranged into a modular and extensible framework facilitating addition or replacement of algorithms for variou scientific and biometric experiments and testing. A MARF-implementing system can run distributively, stand-alone, or may just act as a library in applications. The backbone of MARF consists of pipelined stages that communicate with each other in order to get the data they need for processing in a chained manner. In general, the pipeline consists of four basic stages: sample loading, prepro-

Vassev E. and A. Mokhov S. (2009).
SELF-OPTIMIZATION PROPERTY IN AUTONOMIC SPECIFICATION OF DISTRIBUTED MARF WITH ASSL.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 331-335
DOI: 10.5220/0002257303310335

cessing, feature extraction, and training/classification.

The classical MARF was extended (Mokhov, 2006) to allow the stages of the pipeline to run as distributed nodes as approximately illustrated in Figure 1. The basic stages and the front-end were implemented without backup recovery or hot-swappable capabilities at this point; just communication over Java RMI (Wollrath and Waldo, 2005), CORBA (Sun Microsystems, 2004), and XML-RPC WebServices (Sun Microsystems, 2006).

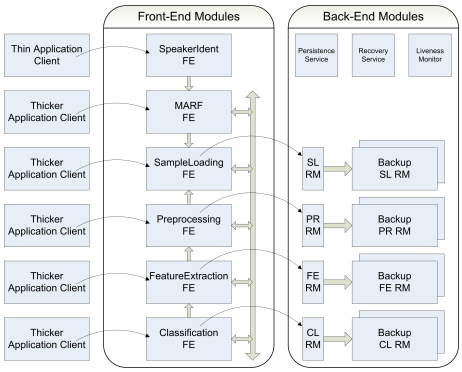

Figure 1: The Distributed MARF Pipeline.

There are a number of applications that test MARF's functionality and serve as examples of how to use MARF's modules. One of the most prominent applications is `SpeakerIdentApp` – Text-Independent Speaker Identification (who is the speaker, their gender, accent, spoken language, etc.). Its distributed extension is designed to support high-volume processing of recorded audio, textual, or imagery data among possible pattern-recognition and biometric applications of DMARF. Most of the emphasis in MARF was in audio, such as conference recordings (Mokhov, 2007) with purpose of attribution of uttered material to speakers' identities. Similarly, a bulk of recorded phone conversations can be processed in collaborating police departments for forensic analysis and biometric subject identification. Here through runs of MARF's pipeline instances on a remote machine an investigator has the ability of uploading from, e.g., a laptop, PDA, or cellphone collected voice samples to the servers constituting a DMARF-implementing network.

DMARF Self-optimization Requirements

DMARF's capture as an autonomic system primarily covers the autonomic functioning of the distributed pattern-recognition pipeline and its optimization, specifically its most computationally and I/O intensive Classification stage. The two major functional requirements applicable to large DMARF installations related to self-optimization are discussed further:

- *Training set classification data replication.* A DMARF-based system may do a lot of multimedia data processing and number crunching throughout the pipeline. The bulk of I/O-bound data processing falls on the sample loading stage and the classification stage. The preprocessing, feature extraction, and classification stages also do a lot of CPU-bound number crunching, matrix operations, and other potentially heavy computations. The stand-alone local MARF instance employs dynamic programming to cache intermediate results, usually in the form of feature vectors, inverse co-variance matrices, and other array-like data. A lot of these data are absorbed by the classification stage. In the case of the DMARF, such data may end up being stored on different hosts that run the classification service potentially causing recomputation of the already computed data on another classification host that did a similar evaluation already. Thus, the classification stage nodes need to communicate to exchange the data they have lazily acquired among all the classification members. Such data mirroring/replication would optimize a lot of computational effort on the end nodes.
- *Dynamic communication protocol selection.* Another aspect of self-optimization is automatic selection of the available most efficient communication protocol in the current run-time environment. E.g. if DMARF initially uses WebServices XML-RPC and later discovers all of its nodes can also communicate using say Java RMI, they can switch to that as their default protocol in order to avoid marshaling and demarshaling heavy SOAP XML messages that are always a subject of a big overhead even in the compressed form.

2.2 ASSL

The Autonomic System Specification Language (ASSL) (Vassev, 2008) approaches the problem of formal specification and code generation of autonomic systems (ASs) within a framework. The core of this framework is a special formal notation and a toolset including tools that allow ASSL specifications to be edited and validated. In general, ASSL considers ASs as composed of autonomic elements (AEs) communicating over interaction protocols. To specify those, ASSL is defined through the formalization of tiers. The ASSL tiers (cf. Figure 2) are abstractions of different aspects of any given AS. There are three major tiers (three major abstraction perspectives), each

composed of sub-tiers:

- *AS tier* – forms a general and global AS perspective, where we define the general system rules in terms of *service-level objectives (SLO)* and *self-management policies*, *architecture topology*, and *global actions*, *events*, and *metrics* applied in these rules. Note that ASSL expresses policies with *fluents* (special states) (Vassev, 2008).
- *AS Interaction Protocol (ASIP) tier* – forms a communication protocol perspective, where we define the means of communication between AEs. The ASIP tier is composed of *channels*, *communication functions*, and *messages*.
- *AE tier* – forms a unit-level perspective, where we define interacting sets of individual AEs with their own behavior.

For more details on the ASSL multi-tier specification model and the ASSL framework toolset, please refer to (Vassev, 2008). Note that as part of the framework validation and in the course of a new currently ongoing research project at Lero, ASSL has been used to specify autonomic properties and generate prototyping models for a few prospective autonomic systems such as the NASA ANTS concept mission (Vassev et al., 2008) and others.

3 ASSL SELF-OPTIMIZATION MODEL FOR DMARF

In our approach, we make DMARF autonomic (ADMARF) by adding an autonomic manager layer to the system architecture. This layer strives to imply an autonomic behavior over the entire system by imposing self-management policies. Here, the DMARF Classification stage is augmented with a self-optimizing autonomic policy. We use ASSL to specify this policy and generate implementation for the same.

Appendix A presents a partial specification of the ASSL self-optimization model for ADMARF. As specified, the autonomic behavior is encoded in a special ASSL construct denoted as the `SELF_OPTIMIZING` policy. The latter is specified at two levels – the global AS-tier level and the level of a single AE (the AE-tier). The algorithm behind is described by the following elements:

- Any time when ADMARF enters in the Classification stage, a self-optimization behavior takes place.
- The Classification stage itself forces the stage nodes synchronize their latest cached results.

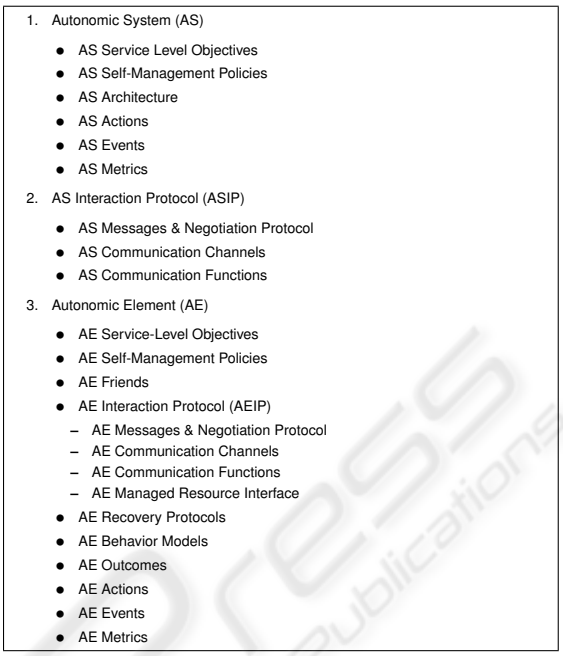

Figure 2: ASSL Multi-Tier Model (Vassev, 2008).

  Here each node is asked to get the results of the other nodes.
- Before starting with the real computation, each stage node strives to adapt to the most efficient currently available communication protocol.

What follows describe the ASSL specification of the simple self-optimization algorithm revealed here.

3.1 AS Tier Specification

At this tier we specify a global system-level `SELF_OPTIMIZING` policy and the actions and events supporting that policy. ASSL supports policy specifications with special constructs called *fluents* and *mappings* (Vassev, 2008). While the former are special states with conditional duration, the latter simply map actions to be executed when the system enters in such a state.

Figure 3 depicts the AS-tier specification of the `SELF_OPTIMIZING` policy. As we see the policy is triggered when the special fluent `inClassificationStage` is initiated. Here, when ADMARF enters the Classification stage at the AS-level the `enteringClassificationStage` event is prompted to initiate the `inClassificationStage` fluent.

Further, this fluent is mapped to an AS-level `runGlobalOptimization` action (cf. Appendix A). This action iterates over all the Classification stage nodes specified as distinct

AEs (cf. Section 3.2) and calls for each node a special AE-level `synchronizeResults` action (cf. Appendix A). In case of exception, the `optimizationNotSucceeded` event is prompted; otherwise, the `optimizationSucceeded` event is prompted. Both events terminate the `inClassificationStage` fluent, and consecutively ADMARF exits the `SELF_OPTIMIZING` policy.

```
SELF_OPTIMIZING {
  // DMARF enters in the Classification Stage
  FLUENT inClassificationStage  {
     INITIATED_BY { EVENTS.enteringClassificationStage }
     TERMINATED_BY {EVENTS.optimizationSucceeded,
                    EVENTS.optimizationNotSucceeded }
  }
  MAPPING {
     CONDITIONS { inClassificationStage }
     DO_ACTIONS { ACTIONS.runGlobalOptimization }
  }
}
```

Figure 3: AS Tier SELF_OPTIMIZING Policy.

To distinguish the AEs from the other AEs in ADMARF, we specified the architecture topology of the system. For this we used the `ASARCHITECTURE` ASSL construct (Vassev, 2008). Appendix A presents the specification of the ADMARF architecture topology. Note that this is a partial specification depicting only two AEs. The full `ASARCHITECTURE` specification includes all the AEs of ADMARF. As depicted, we specified a special group of AEs called `CLASSF_STAGE` with members all the AEs representing the Classification stage nodes. This group allows the `runGlobalOptimization` action iterates over the stage nodes.

3.2 AE Tier Specification

At this tier we specified the `SELF_OPTIMIZING` policy for the Classification stage nodes. Here we specified for every node a distinct AE. Our specification has the partial specification of two AEs, each representing a single node of the Specification stage. At this level, self-optimization concentrates on adapting the single nodes to the most efficient communication protocol. Similar to the AS-level policy specification (cf. Section 3.1), an `inCPAdaptation` fluent is specified to trigger such adaptation when ADMARF enters in the Classification stage. This fluent is initiated by the AS-level `enteringClassificationStage` event (cf. Appendix A). The same fluent is mapped to an `adaptCP` action to perform the needed adaptation. This action is specified as `IMPL`, i.e., requiring further implementation (Vassev, 2008). In ASSL, we specify `IMPL` actions to hide complexity via abstraction. Here, the `adaptCP` action is a complex structure, which explanation is beyond the scope of this paper. Therefore, we abstracted the specification of this action (through `IMPL`) and provided only the prerequisite *guard* conditions and prompted events.

4 CONCLUSIONS

We constructed a self-optimizing specification model for ADMARF. To do so we devised an algorithm with ASSL for the Classification stage of the DMARF's pattern recognition pipeline. When fully-implemented, the ADMARF system will be able to fully function in autonomous environments, be those on the Internet, huge multimedia processing farms, law enforcement, or simply even patter-recognition research groups that can rely more on the availability of their systems that run for multiple days, unattended.

Future Work. Some work on both projects, DMARF and ASSL is still on-going, that, when complete, will allow a more complete realization of ADMARF. Some items of the future work are as follows:

- We plan on integration of the ASSL aspects such as self-protection and self-healing with the work to build a complete ADMARF.
- We plan on releasing the Autonomic Specification of DMARF, ADMARF as open-source.

ACKNOWLEDGEMENTS

This work was supported in part by an IRCSET postdoctoral fellowship grant (now termed as EMPOWER) at University College Dublin, Ireland, by the Science Foundation Ireland grant 03/CE2/I303_1 to Lero – the Irish Software Engineering Research Centre, and by the Faculty of Engineering and Computer Science of Concordia University, Montreal, Canada.

REFERENCES

Mokhov, S. (2006). On design and implementation of distributed modular audio recognition framework: Requirements and specification design document. [online], http://arxiv.org/abs/0905.2459. Project report.

Mokhov, S. A. (2007). Introducing MARF: a modular audio recognition framework and its applications for scientific and software engineering research. In *Advances in Computer and Information Sciences and Engineering*, pages 473–478. Springer Netherlands.

Mokhov, S. A. (2008). Study of best algorithm combinations for speech processing tasks in machine learning

using median vs. mean clusters in MARF. In Desai, B. C., editor, *Proceedings of C3S2E'08*, pages 29–43. ACM.

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

Murch, R. (2004). *Autonomic Computing: On Demand Series*. IBM Press, Prentice Hall.

Sun Microsystems (2004). Java IDL. Sun Microsystems, Inc. http://java.sun.com/j2se/1.5.0/docs/guide/idl/index.html.

Sun Microsystems (2006). The java web services tutorial (for Java Web Services Developer's Pack, v2.0). Sun Microsystems, Inc. http://java.sun.com/webservices/docs/2.0/tutorial/doc/index.html.

Vassev, E., Hinchey, M. G., and Paquet, J. (2008). 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.

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

Wollrath, A. and Waldo, J. (1995–2005). Java RMI tutorial. Sun Microsystems, Inc. http://java.sun.com/docs/books/tutorial/rmi/index.html.

A ASSL SPECIFICATION

```
// ASSL self-optimization specification model for DMARF

AS DMARF {
  ASSELF_MANAGEMENT {
    // DMARF strives to optimize by synchronizing cached
    // results before starting with the Classification Stage
    SELF_OPTIMIZING {
      // DMARF enters in the Classification Stage
      FLUENT inClassificationStage  {
        INITIATED_BY { EVENTS.enteringClassificationStage }
        TERMINATED_BY {  EVENTS.optimizationSucceeded,
            EVENTS.optimizationNotSucceeded }
      }
      MAPPING {
        CONDITIONS { inClassificationStage }
        DO_ACTIONS { ACTIONS.runGlobalOptimization }
      }
    }
  } // ASSELF_MANAGEMENT

  ASARCHITECTURE {
    AELIST {AES.CLASSF_STAGE_NODE_1, AES.CLASSF_STAGE_NODE_2}
    DIRECT_DEPENDENCIES {
      DEPENDENCY AES.CLASSF_STAGE_NODE_1 { AES.CLASSF_STAGE_NODE_2 }
      DEPENDENCY AES.CLASSF_STAGE_NODE_2 { AES.CLASSF_STAGE_NODE_1 }
    }
    GROUPS {
      GROUP CLASSF_STAGE {
        MEMBERS { AES.CLASSF_STAGE_NODE_1, AES.CLASSF_STAGE_NODE_2 }
      }
    }
  }

  ACTIONS {
    ACTION runGlobalOptimization {
      GUARDS { ASSELF_MANAGEMENT.SELF_OPTIMIZING.inClassificationStage }
      DOES {
        FOREACH member IN ASARCHITECTURE.GROUPS.CLASSF_STAGE.MEMBERS {
          call IMPL member.ACTIONS.synchronizeResults
        }
      }
      TRIGGERS {
        EVENTS.optimizationSucceeded
      }
      ONERR_TRIGGERS {
        // if error then report unsuccessful optimization
        EVENTS.optimizationNotSucceeded
      }
    }
  } // ACTIONS

  EVENTS { // these events are used in the fluents specification
    EVENT enteringClassificationStage { }
    EVENT optimizationSucceeded { }
    EVENT optimizationNotSucceeded { }
  } // EVENTS
} // AS DMARF

AES {
  AE CLASSF_STAGE_NODE_1 {
      AESELF_MANAGEMENT {
      SELF_OPTIMIZING {
        FLUENT inCPAdaptation {
          INITIATED_BY { AS.EVENTS.enteringClassificationStage }
          TERMINATED_BY {  EVENTS.cpAdaptationSucceeded,
              EVENTS.cpAdaptationNotSucceeded }
        }
        MAPPING {
          CONDITIONS { inCPAdaptation }
          DO_ACTIONS { ACTIONS.adaptCP }
        }
      }
    }

    ACTIONS {
      ACTION IMPL synchronizeResults {
        GUARDS { AS.ASSELF_MANAGEMENT.SELF_OPTIMIZING.
            inClassificationStage
        }
      }
      ACTION IMPL adaptCP {
        GUARDS { AESELF_MANAGEMENT.SELF_OPTIMIZING.inCPAdaptation }
        TRIGGERS { EVENTS.cpAdaptationSucceeded }
        ONERR_TRIGGERS { EVENTS.cpAdaptationNotSucceeded }
      }
    } // ACTIONS

    EVENTS { // these events are used in the fluents specification
      EVENT cpAdaptationSucceeded { }
      EVENT cpAdaptationNotSucceeded { }
    } // EVENTS
  }
  AE CLASSF_STAGE_NODE_2 {
    AESELF_MANAGEMENT {
      SELF_OPTIMIZING {
        FLUENT inCPAdaptation {
          INITIATED_BY { AS.EVENTS.enteringClassificationStage }
          TERMINATED_BY {  EVENTS.cpAdaptationSucceeded,
              EVENTS.cpAdaptationNotSucceeded }
        }
        MAPPING {
          CONDITIONS { inCPAdaptation }
          DO_ACTIONS { ACTIONS.adaptCP }
        }
      }
    }

    ACTIONS {
      ACTION IMPL synchronizeResults {
        GUARDS { AS.ASSELF_MANAGEMENT.SELF_OPTIMIZING.
            inClassificationStage
        }
      }
      ACTION IMPL adaptCP {
        GUARDS { AESELF_MANAGEMENT.SELF_OPTIMIZING.inCPAdaptation }
        TRIGGERS { EVENTS.cpAdaptationSucceeded }
        ONERR_TRIGGERS { EVENTS.cpAdaptationNotSucceeded }
      }
    } // ACTIONS

    EVENTS { // these events are used in the fluents specification
      EVENT cpAdaptationSucceeded { }
      EVENT cpAdaptationNotSucceeded { }
    } // EVENTS}
  }
}
```