ASSL SPECIFICATION OF RELIABILITY SELF-ASSESSMENT IN THE AS-TRM

Emil Vassev, Olga Ormandjieva and Joey Paquet
Department of Computer Science and Software Engineering
Concordia University, Montreal, Quebec, H3G 1M8, Canada

Keywords: Specification language, autonomic system, reactive system, reliability self-assessment.

Abstract: This article is an introduction to our research towards a formal framework for tackling reliability in reactive autonomic systems with self-monitoring functionality. The Autonomic System Specification Language (ASSL) is a framework for formally specifying and generating autonomic systems. With ASSL, we can specify high-level behavior policies, which shows that it is very appropriate language for specifying reliability models as part of overall system behavior. In this paper, we show how ASSL can be used to specify reliability self-assessment i n
reliability self-assessment is performed at two levels: autonomic element (local) and system (global). It depends on the configuration of the system and is concerned with the uncertainty analysis of the AS-TRM as it evolves. An appropriate architecture for supporting reliability self-assessment, along with a communication mechanism to implement the reactive and autonomic behavior, are specified with ASSL.

1 INTRODUCTION

In order to overcome the increasing complexity of IT infrastructure and the associated workload required to maintain such a complex infrastructure, we need self-adaptive and autonomic computing systems. Autonomic systems (AS) are not only inherently complex, but also evolve during execution. Thus, it is important to monitor the behavior of such systems and to ensure a high level of system reliability at runtime.

Research Problem. The computing industry thrives on the assumption in the marketplace that software is reliable and correct, but countless examples from experience over the decades cast doubt on the validity of this assumption. There is no automated, general purpose method for building reliable systems that fully meets system reliability requirements. This represents a major gap that has yet to be fully addressed by the software engineering community. The runtime reliability verification method proposed in this paper attempts to bridge this gap through self-monitoring of system reliability and its assessment measured against the system policies of interest at runtime. Reliability self-assessment has to be regulated by policies stating the required minimum level of reliability for the system and the constraints on system reliability level fluctuations at runtime.

Approach. Our paper reports on the ongoing work on built-in reliability self-assessment, which would allow for the capture of reliability policies, the modeling of reliability self-assessment and the implementation of a runtime reliability verification method in an evolving Autonomic System Timed Reactive Model (AS-TRM). The vision of the AS-TRM is to be able to create autonomic distributed real-time reactive systems on a framework that leverages their modeling, development, integration and maintenance. The reliability self-monitoring of an evolving AS-TRM is rooted in the theory of Markov chains. The reliability level is an indicator of the amount of certainty (excess entropy) in the environment-system's behavior (Ormandjieva et al., 2006; Ormandjieva, 2002). This paper extends our previous work on reliability to complex composite reliability structures.

One of the main contributions of this paper is the formalization of the AS-TRM approach with the Autonomic System Specification Language (ASSL) (Vassev, Paquet, 2007) – which is a framework for formally specifying and generating ASs, such systems being specified as formal executable models with an interaction protocol and autonomic

Vassev E., Ormandjieva O. and Paquet J. (2007).
ASSL SPECIFICATION OF RELIABILITY SELF-ASSESSMENT IN THE AS-TRM.
In *Proceedings of the Second International Conference on Software and Data Technologies - SE*, pages 198-206
DOI: 10.5220/0001328601980206

elements. In our understanding, ASSL facilitates the achievement of the goal of specifying the reliability self-assessment in such a framework. Moreover, ASSL focuses on the service-level objectives and the AS's self-management policies, thus making ASSL suitable for specifying the reliability self-assessment. The advantages of using ASSL for modeling AS-TRM systems are that it enables software assurance and it provides proof of the correctness of the behavior of such systems. Moreover, formally derived models can be used as the basis for code generation.

The rest of this paper is organized as follows: Section 2 surveys related work. The AS-TRM and the reliability self-assessment mechanism at both autonomic component and system levels are described in Section 3. Section 4 introduces ASSL. Section 5 presents the formal specifications of reliability self-assessment in the AS-TRM with ASSL. Our conclusions and future work directions are outlined in Section 6.

2 RELATED WORK

IBM Research has developed a framework called Policy Management for Autonomic Computing (PMAC) (IBM Tivoli, 2005), which provides a standard model for the definition of policies and an environment for the development of software objects that hold and evaluate policies. PMAC is used for the development and management of intelligent autonomic software agents. With PMAC, these agents have the ability to dynamically change their behavior, an ability provided through a formal specification of policies encompassing the scope under which these policies are applicable. Moreover, policy specification includes the conditions under which a policy is in conformity (or has been violated), a set of resulting actions, goals or decisions that need to be taken and the ability to determine the relative value (priority) of multiple applicable actions, goals or decisions.

In (Goseva-Popstojanova, Kamavaram, 2004), a methodology was proposed for the uncertainty analysis of architecture-based software reliability models suitable for large, complex, component-based applications which is applicable throughout the software life cycle. Within this methodology, two methods for uncertainty analysis have been developed: the method of moments and Monte Carlo simulation. The method of moments is used to quantify the uncertainty in software reliability due to uncertainty in component reliabilities. The expressions derived in (Goseva-Popstojanova, Kamavaram, 2004) are valid for random variables and do not allow the uncertainty in software reliability to be studied due to uncertainty in the operational profile.

In (Dai, 2005), a new model-driven scheme for autonomic management is presented, based on a comprehensive survey of reliability models. This scheme can better allocate resources by using the reliability models to predict and direct the distribution of monitoring efforts. If certain services or components are predicted to have a high degree of reliability at a particular time, then there is no need for intensive monitoring during that period. However, those with low reliability require more intensive monitoring.

The reliability evaluation method discussed in this paper differs from previous reliability evaluation methods in the following ways:

- The most common stochastic queuing model for the arrival time of the external events, namely a Poisson distribution, is assumed.
- It is based on the architecture model of an AS and the extended state diagrams.

The work presented in this paper builds on the research results on the reliability self-assessment of autonomic components in the AS-TRM (Vassev et al., 2006; Ormandjieva et al., 2006).

3 AS-TRM AND RELIABILITY ASSESSMENT

The AS-TRM (Vassev et al., 2006; Ormandjieva et al., 2006) differs considerably from related work in the area of autonomic computing in that it targets the modeling of both reactiveness and self-managing in distributed systems. This section provides a comprehensive conceptual view of the AS-TRM architecture (see Figure 1), which is intended to capture and convey the significant decisions that will serve as a foundation for further design and implementation. The architectural concepts for ASs are mainly based on the IBM's blueprints and the on-going research into autonomic computing being conducted at IBM laboratories (IBM, 2006).

3.1 AS-TRM Architecture

The AS-TRM is a three-tier layered model, in which each upper tier communicates only with the tier immediately below it (Vassev et al., 2006). The three-tier structure describes the AS configuration, autonomic peer groups and grouped reactive components.

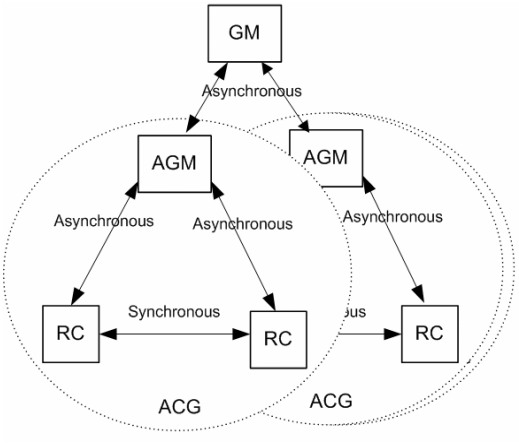

Figure 1: AS-TRM Architecture.

Reactive Component (RC). This tier encapsulates reactive objects in an AS-TRM reactive component. A reactive object is modeled as a labeled transition system. The timing requirements are modeled as constraints on the transitions, and are specified with the aid of local clocks initialized as actions associated with transitions. The synchronous interaction between the reactive objects allows for realization of the reactive task. Synchronous communication axioms govern the interaction between the reactive objects. Communication between an RC and its upper tier – the ACG – is realized through an interface, and is asynchronous.

Autonomic Group of RCs (ACG). The AS-TRM Component Group is a set of synchronously communicating RCs cooperating in fulfillment of a group task. Each ACG can independently accomplish a complete real-time reactive task. The self-monitoring behavior at the ACG tier level and the asynchronous interaction in an ACG are realized by the ACG's Autonomic Group Manager (AGM).

Autonomic System (AS). The AS consists of a set of asynchronously communicating ACGs. The self-monitoring behavior and the asynchronous interaction between the AS and the ACGs are realized by the Global Manager (GM). The responsibilities of the GM include the continuous monitoring of the reliability level of the AS required to the reliability policies of the AS.

Anatomy of the AGM and the GM. The GM is responsible for the self-configuring, self-healing and self-optimizing, and self-protecting of the autonomic group. The responsibilities of the AGM include the continuous monitoring of the reliability level of the ACG required by the evolving nature of the group for self-configuration purposes. Every ACG communicates with the GM via its AGM, its status and its measurements. According to the input received from the ACGs, the GM makes decisions based on the policies, facts and rules and communicates instructions to the AGMs.

The reactive behavior is modeled at the RC level. We model the environmental objects communicating with the system as reactive objects, and incorporate them into the RCs fulfilling the corresponding reactive task. Autonomic functionalities like reliability self-assessment can be implemented at group level, using locally maintained policies and specific characteristics such as timing constraints and synchronous communication axioms, and at system level using the knowledge on the global policies, system characteristics, etc.

3.2 ACG Reliability

In our approach, the reliability level of an ACG is an indicator of the amount of certainty (excess entropy) in the environment-system's behavior (Ormandjieva et al., 2006; Ormandjieva, 2002). The reliability self-monitoring of the evolving AS-TRM is based on the theory of Markov chains.

Traditionally, the use of Markov chains has required monitoring the states of all RCs. However, this approach does not scale (there is too much to monitor) and usually does not work (for most systems, the probabilities cannot be calculated). In our approach, reliability is assessed from the specifications of the reactive objects modeled as labeled transition systems, reactive components consisting of synchronously interacting reactive objects and the AS-TRM architecture, and so those states do not require monitoring. The specification information is available in the form of text files and serves as input to the reliability assessment, which has to be performed before the evolving system's change is actually implemented. Moreover, the reliability assessment model allows for calculation of the transition probabilities in the Markov chain from the extended state machines of the individual reactive objects and of groups of synchronously communicating reactive objects. What this means is that there is no need for statistical data on the system's usage. The details of reliability self-monitoring in autonomic components are given in (Ormandjieva et al., 2006; Ormandjieva, 2002). One of the contributions of this paper to reliability assessment is the definition of reliability for AS, which is given in Section 3.3.

3.3 AS Reliability

The reliability levels of the ACGs, reported to the GM, are used to determine the reliability of the whole AS based on the configuration of its *n* ACGs. There are two interesting limit cases of such a configuration, namely, parallel structures and serial structures (Zhang, Nakamura, 2005). We have a parallel configuration when there is at least one component necessary to ensure that the entire AS functions. Assuming the independence of failures of the corresponding components, reliability is calculated:

$$R = 1 - \Pi_{k=1,n} (1 - R_k)$$

where R_k is the reliability of ACG_k. In a serial reliability structure, the functioning of the system is ensured while all the components are functioning properly. In this case, reliability is given by the following formula:

$$R = \Pi_{k=1,n} R_k$$

In the complex composite reliability structure, the above two simple reliability structures can be adopted to form a composite reliability structure. The basic composite reliability structures include the parallel-serial composite reliability structure and the serial-parallel composite reliability structure (Dai, 2005). The reliability of the system with the parallel-serial composite reliability structure is defined as

$$R = \Pi_{m=1,C} (1 - \Pi_{k=1,Cm} (1 - R^m{}_k))$$

where C is the number of serial composites and C_m is the number of parallel components in the serial composite m. The reliability of the system with the serial-parallel composite reliability structure is

$$R = 1 - \Pi_{m=1,C} (1 - \Pi_{k=1,Cm} R^m{}_k)$$

where C is the number of parallel composites and C_m is the number of serial components in the parallel composite m.

3.4 Reliability Self-Assessment Method

The reliability assessment at the AS level would allow a request for reconfiguration of the system to be deployed by the GM if and only if the policies for reliability level fluctuations hold. The **reliability policy** assumed in this paper is the following: "*The reconfiguration shall not lead to a reliability level below the required minimum.*" The reliability self-assessment tasks during runtime are modeled as a control loop (IBM, 2006) comprising the following steps: (i) **Monitor**: continuously track requests for evolving changes within the AS-TRM, either from the environment or from within the system, such as configuration changes, time constraint changes and synchronization axiom changes; (ii) **Analyze**: based on the requested change received in step (i), predict the new system reliability from the new reliability of the ACG and of the new system configuration (see 3.2 and 3.3), and assess the request for change against the AS reliability policies; (iii) **Execute**: based on the results of the analysis performed in step (ii), accept or deny the request for change.

Advantages. The advantages of the reliability self-assessment in the AS-TRM include:

- Assessment of reliability from the specifications of the reactive objects, the RCs and the architecture of groups, with the result that complex mechanisms for monitoring the states of the reactive components are not required.
- The probabilities are calculated from the extended state machines that specify the behavior of the reactive objects/components, and thus do not rely on the statistical data collected on the system behavior at run time.
- The reliability self-assessment is performed before a request for change is implemented, thus ensuring compliance with the policies during the self-management tasks.

Assumptions. In our system reliability analysis, all the ACGs are considered to function independently, and therefore it is assumed that there is independence of failures of the corresponding ACGs. This view is also supported by the ASSL approach to specifying an AS.

4 ASSL

The Autonomic System Specification Language (ASSL) is a framework that implies a multi-tier structure for specifying ASs. By its virtue, ASSL is generic and expressive enough to describe a variety of ASs (Vassev, Paquet, 2007). The ASSL framework is defined through formalization tiers. Over these tiers, ASSL provides a layered structure for specifying ASs as formal executable models. ASSL defines an AS with its interaction protocol and autonomic elements (AE). The ASSL's tiers and their sub-tiers describe different aspects of the AS, like policies, communication interfaces, execution semantics, actions, etc. All of them ensure that the system is well-defined and consistent, providing a "bottom-up" style where the upper tiers are expressed using the elements described in the lower ones. The following elements represent the major tiers and sub tiers in ASSL.

I. Autonomic System (AS)
   - AS Service-Level Objectives
   - AS Self-Management Policies
   - Metrics

- Architecture

II. AS Interaction Protocol (ASIP)
- Public AS Messages & Negotiation Protocol
- Public Communication Channels
- Public Communication Functions

III. Autonomic Element (AE)
- AE Service-Level Objectives
- AE Self-Management Policies
- Friends
- AE Interaction Protocol (AEIP)
  - Private AE Messages & Negotiation Protocol
  - Private Communication Channels
  - Private Communication Functions
  - Managed Resource Interface
- Recovery Protocol
- Behavior
- Outcomes
- Actions
- Events
- Metrics

4.1 Autonomic System Tier

The Autonomic System tier specifies an autonomic system in terms of service-level objectives, self-management policies, metrics and architecture (Vassev, Paquet, 2007).

AS Service-Level Objectives. Service-level objectives (SLO) are a higher-level form of behavioral specification that establishes objectives – for example, performance. The ASSL concept (Vassev, Paquet, 2007) assumes that the AS service-level objectives (AS SLO) constitute a global task, the realization of which is to be decomposed into AE service-level objectives (AE SLO).

AS Self-Management Policies. At this tier, the ASSL formal model specifies the four self-management policies of an AS as stated in (IBM, 2006): self-configuring, self-healing, self-optimizing and self-protecting. In addition, ASSL leaves available the option of specifying other AS-level policies that cannot be classified as any of these four policies.

AS Metrics. ASSL generally classifies metrics as AS-level metrics and AE-level metrics, together constituting a set of global metrics – parameters and observables – that the AEs can control.

AS Architecture. ASSL addresses ASs as multi-agent systems, where the individual agents are AEs controlling resources and delivering services (Tesauro et al., 2004). Each AS agent is an AE. At this tier, the ASSL framework helps specify the topology of the AS. The architecture is specified as a correlation between the AEs or groups of AEs.

4.2 AS Interaction Protocol

To achieve effective interoperation among AEs, we need the individual AEs to adopt standard communication interfaces. At this tier, the ASSL framework specifies the AS-level interaction protocol (ASIP). The ASSL framework specifies the AEs as entities communicating over ASIP. ASIP is a public communication interface, expressed as public messages exchanged among AEs using public communication channels and public communication functions (Vassev, Paquet, 2007).

4.3 Autonomic Element Tier

The ASSL formal model considers AEs to be analogous to software agents able to manage their own behavior and their relationships with other autonomic elements, through which they provide or consume computational services. At this level of the framework, ASSL describes the correlation between low-level system measurements, events, and actions, and higher-level AE service-level objectives.

5 RELIABILITY SELF-ASSESSMENT WITH ASSL

As a formal language, ASSL defines a language-independent representation for ASs, where they are described as a set of interacting AEs. ASSL provides a rich set of structures and elements, including self-management policy structures (Vassev, Paquet, 2007). In this research, we focus on the specification of reliability self-assessment, which we consider to be a self-management policy. The assumptions underlying the modeling of the tasks required to fulfill the reliability self-assessment are:

- A reconfiguration plan has been received by the GM and propagated to all the AGMs.
- The AGMs have completed their reconfiguration analysis.

The following scenario describes the reliability self-assessment policy specified with ASSL:

1. Each AGM sends a “reconfiguration analysis done” message to the GM (see the “Monitor” step in Section 3.4).

2. The GM requests, from all the AGMs, their reliability levels that correspond to the new configuration.
3. Each AGM sends their reliability level.
4. The GM computes the new reliability level of the AS and analyzes it.
   (*Note: For steps 2, 3 and 4 see the "Analyze" step in Section 3.4*).
5. The GM accepts or denies the request for reconfiguration (see the "Execute" step in Section 3.4)

5.1 AS Tier

At this tier, we specify a global quality metric, called *SystemReliability*, which, as its name implies, is an indicator of the system reliability. We specify a range for system reliability between *min* and *max* (see Listing 1). This metric is used by the GM reliability self-assessment behavior (see 5.3). Moreover, at this tier, we specify the AS-TRM architecture model. In our example, we consider an AS-TRM as consisting of a GM and three AGMs. As specified (see Listing 1), the AS-TRM has a centralized architecture. The GM and all the AGMs are grouped together, in *groupGM*, where the GM is the group council node (Vassev, Paquet, 2007), and AGM1, AGM2 and AGM3 are the member nodes. The group council is an AE coordinating the work of the group members. Moreover, we specify a list of dependencies, to show that the AGMs do not depend on each other, but on the GM. Listing 1 represents a partial specification of the AS-TRM's AS tier.

```
AS ASTRM {
   METRICS {
      METRIC SystemReliability {
      type = QUALITY;
      description = "Measures the reliability of the system.";
      THRESHOLD_CLASS Reliability: double [min, max];
   }
    ASARCHITECTURE { // centralized system with 3 AGMs
        AE_LIST: GM, AGM1, AGM2, AGM3;
        DIRECT_DEPENDENCIES {
            DEPENDENCY GM: AGM1, AGM2, AGM3;
            DEPENDENCY AGM1: GM;
            DEPENDENCY AGM2: GM;
            DEPENDENCY AGM3: GM; };
        TRANSITIVE_DEPENDENCIES {};
        GROUPS {
            GROUP groupGM {
                MEMBERS: AGM1, AGM2, AGM3;
                FINAL COUNCIL: GM;
            };
        };
    } // ASARCHITECTURE
} // ASTRM
```

Listing 1: AS AS-TRM Partial Specification.

In the AS architecture specification, ASSL considers only AEs. Therefore, we do not specify the RCs (see Section 3.1), since they are not AEs, but rather managed resources which are controlled by the associated AGM (Vassev, Paquet, 2007).

5.2 ASIP Tier

At this tier, we specify the communication protocol needed by the GM and AGMs to communicate and transfer data for the needs of the reliability self-assessment policy. First, we specify the messages needed for the reliability self-assessment specification (see Listing 2):

- *analysisDone* – used by the AGMs to report to the GM that they have completed the reconfiguration analysis.
- *requestReliability* – used by the GM to request the reliability level from the AGMs. Moreover, the same message is used by the AGMs to return the computed reliability level to the GM.

The reconfiguration itself is not specified here. In general, it should be triggered by the GM, i.e. the GM should send a message to all the AGMs requesting reconfiguration, together with a reconfiguration plan.

```
ASIP {
   MESSAGES {
      FINAL MESSAGE analysisDone  {
          SENDER: {AGM1, AGM2, AGM3};
          RECEIVER: GM;
          TYPE: TEXT;
          ID: none;
          BODY: "analysisDone ";}
      FINAL MESSAGE requestReliability {
          SENDER: {GM, AGM1, AGM2, AGM3};
          RECEIVER: {GM, AGM1, AGM2, AGM3};
          TYPE: TEXT;
          ID: none;
          BODY: "systemReliability = ?";}
   }
   CHANNELS {
      CHANNEL configChannel {
          ACCEPT: { requestReliability, analysisDone };
          ACCESS: SEQUENTIAL;
          DIRECTION: BIDIRECTIONAL;   }
   }
   FUNCTIONS {
      FUNCTION sendRequestReliability {
          requestReliability >> configChannel; }
      FUNCTION receiveRequestReliability {
          requestReliability << configChannel; }
      FUNCTION sendAnalysisDone {
          analysisDone >> configChannel; }
      FUNCTION receiveAnalysisDone {
          analysisDone << configChannel; }
   }
} // ASIP
```

Listing 2: ASIP Partial Specification.

In addition, at this tier, we specify a single communication channel (see *configChannel* in Listing 2) and the functions operating the messages over that channel, i.e. functions for receiving and sending the messages *requestReliability* and *analysisDone* from and to the *configChannel*. It is important to mention that the ASIP specification is not complete. In Listing 2, we present only those messages, channels and functions needed by the system reliability self-assessment specification.

5.3 AE Tier - GM Specification

At the AE tier, we specify the AS-TRM's AEs, i.e. we specify the GM, AGM1, AGM2 and AGM3. In Listing 3, we present a part of the GM specification, which describes the GM's reliability self-assessment policy specification. To specify the reliability self-assessment policy, we use four major elements:

- *SystemReliability* – an AS metric expressing the current system reliability level (see Section 5.1);
- *RELIABILITY_SELF_ASSESSMENT* – a self-management policy describing in ASSL terms the reliability self-assessment policy of the GM. Here, we use a set of *fluents* and *mappings* to specify the policy (Vassev, Paquet, 2007). With the fluents, we express specific situations in which the reliability self-assessment policy is interested, and with the mappings we map those situations to actions (see Listing 3). A fluent has a timed duration, for example a state like "reliability is changing". When the system gets into that specific condition, the fluent is considered to be valid;
- *actions* – a set of actions that could be undertaken by the GM in response to certain conditions, and according to the reliability self-assessment policy;
- *events* – a set of events that are triggered by, the actions, according to the reliability self-assessment policy.

```
AE GM {  // GM (Global Manager) ASSL specification
    AESELF_MANAGEMENT {
        OTHER_POLICIES {
            RELIABILITY_SELF_ASSESSMENT {
                FLUENT inRequestReliability {
                    INITIATES: isAnalysisDone ;
                    TERMINATES: isRequestReliabilitySent;     }
                FLUENT inReceiveReliability {
                    INITIATES: isRequestReliabilitySent;
                    TERMINATES: isRequestReliabilityReceived; }
                FLUENT inChangeReliability {
                    INITIATES: isRequestReliabilityReceived;
                    TERMINATES: isReliabilityChanged; }
                MAPPING { // request reliability from all the AGMs
                    CONDITION: inRequestReliability;
                    ACTION:  ACTIONS.requestReliability;}
                MAPPING { // receive reliability from all the AGMs
                    CONDITION: inReceiveReliability;
                    ACTION:  ACTIONS.receiveReliability;}
                MAPPING { // receive reliability from all the AGMs
                    CONDITION: inChangeReliability;
                    ACTION:  ACTIONS.computeReliability;}
                MAPPING { // denies the new configuration
                    CONDITION: EVENTS.isConfigurationDenied;
                    ACTION:  ACTIONS.configurationDenied;}
            } // RELIABILITY_SELF_ASSESSMENT
        }
    } // AESELF_MANAGEMENT
    ACTIONS {
        ACTION requestReliability {
            GUARDS: isAnalysisDone ;
            ENSURES: isRequestReliabilitySent;
            DOES {
                foreach member in AS.GROUPS.groupGM.MEMBERS {
                    call: ASIP.FUNCTIONS.sendRequestReliability; }
            }
        }
        ACTION receiveReliability {
            GUARDS: isRequestReliabilitySent;
            ENSURES: isRequestReliabilityReceived;
            DOES {
                foreach member in AS.GROUPS.groupGM.MEMBERS {
                    call: ASIP.FUNCTIONS.receiveRequestReliability; }
            }
        }
        ACTION computeReliability {
            GUARDS: isRequestReliabilityReceived;
            ENSURES: isReliabilityChanged;
            DOES {
                double product = 1;
                foreach member in AS.GROUPS.groupGM.MEMBERS {
    product = product * (1 -  member. METRICS.AGMReliability);}
                set: AS.METRICS.SystemReliability =  1 – product;
            }
            TRIGGERS: isConfigurationDenied;
        }
        ACTION configurationDenied {
            GUARDS: isConfigurationDenied;
            DOES {
                foreach member in AS.GROUPS.groupGM.MEMBERS {
                    call: ASIP.FUNCTIONS.receiveRequestReliability; }
                call:  IMPL DenyConfiguration;
            }
        }
    } // ACTIONS
    EVENTS { // these events are used in the fluents' specification
        EVENT isRequestReliabilityReceived:
            foreach member in AS.GROUPS.groupGM.MEMBERS {
                received ASIP.MESSAGES.requestReliability and
                ASIP.MESSAGES.requestReliability.ID = msgID and
                ASIP.MESSAGES.requestReliability.SENDER = member; }
        EVENT isRequestReliabilitySent:
            foreach member in AS.GROUPS.groupGM.MEMBERS {
                sent ASIP.MESSAGES.requestReliability and
            ASIP.MESSAGES.requestReliability.RECEIVER = member; }
        EVENT isAnalysisDone :
            foreach member in AS.GROUPS.groupGM.MEMBERS {
                received ASIP.MESSAGES.analysisDone  and
                ASIP.MESSAGES.analysisDone.ID = msgID and
                ASIP.MESSAGES.analysisDone .SENDER =member; }
EVENT isReliabilityChanged: changed AS.METRICS.SystemReliability;
        EVENT isConfigurationDenied: ;
    } // EVENTS
} // AE GM
```

Listing 3: AE GM Partial Specification.

The following elements describe the specification listed in Listing 3.

inRequestReliability. This fluent takes place when the GM requests the new reliability levels from the AGMs. The fluent is initiated by the *isAnalysisDone* event, which happens when the GM has received the *analisysDone* message from all the AGMs. Moreover, this fluent terminates when the *isRequestReliabilityEvent* happens, i.e. when the GM has sent the *requestReliability* message to all the AGMs. Further, this fluent is mapped to the *requestReliability* action, which uses the specified at the ASIP tier (see 5.2) *sendRequestReliability* function to send the message to all the AGMs.

inReceiveReliability. This fluent is triggered when the *requestReliability* message has been sent to all the AGMs (see the *isRequestReliabilitySent* specification), and terminates when all the new reliability levels have been received (see *isRequestReliabilityReceived* in Listing 3). This fluent is mapped to the *receiveReliability* action.

inChangeReliability. This fluent is triggered when the *requestReliability* message has been received from all the AGMs (see the

isRequestReliabilityReceived specification), and terminates when the system reliability level has been changed (see the *isReliabilityChanged* specification). It is mapped to the *computeReliability* action, which then applies the formula stated in the algorithm described in Section 3.3 for computing system reliability. Further, the new reliability level is assigned to the AS *SystemReliability* metric. If the new reliability level does not conform to the metric's range (specified by its threshold class – see Listing 1), the action denies the new level and triggers the *isConfigurationDenied* event.

isConfigurationDenied. This event is mapped to the *configurationDenied* action. The latter uses an IMPL routine (see the ASSL clause IMPL in (Vassev, Paquet, 2007)) to deny the new configuration due to a lower reliability level. The IMPL clause states for "further implementation", which means that the ASSL framework will generate an empty routine and its content should be implemented manually.

5.4 AE Tier – AGM Specification

At this tier, we specify an AE AGM class (AECLASS in ASSL). This class specifies three AGM AEs in common: AGM1, AGM2 and AGM3, which extend the AGM class (see the end of Listing 4). Since the AGMs should monitor their own reliability level, we specify a quality metric, called *AGMReliability*, with a reliability range between *min* and *max* (see Listing 4). This metric is used by the AGM reliability self-assessment behavior. As in the GM specification, we specify the reliability self-assessment policy as a separate structure, called *RELIABILITY_SELF_ASSESSMENT*, which is specified in the *OTHER_POLICIES* section, subsection of *AESELF_MANAGEMENT*.

We specify two fluents with the appropriate mappings. The first fluent – *inChangeReliability* - merges the events *isAnalysisDoneSent* and *isReliabilityChanged*. The former is raised when the message announcing a successful reconfiguration is sent to the GM, and the latter is raised when the reliability level has been changed successfully. Therefore, we compute the new reliability level right after sending the reconfiguration analysis message. Moreover, this fluent is mapped to the *computeReliability* action, which uses an IMPL routine, called *ComputeAGMReliability*, to compute the AGM's reliability level. This routine is a further implementation (see the ASSL clause IMPL) of the algorithm described in Section 3.2. If the new reliability level does not conform to the *AGMReliability* metric's range (specified by its threshold class – see Listing 4), the action denies the new level and triggers the *isConfigurationDenied* event. The specification of this event is similar to the specification of its homolog in the GM specification.

The second fluent, called *inRequestReliability*, merges the *isRequestReliabilityReceived* and *isRequestReliabilitySent* events. The former is raised when the message requesting reliability evaluation has been received from the GM, and the latter is raised when the reliability level has been sent to the GM. This fluent is mapped to the *sendReliability* action (see Listing 4), which does the following:

- sets the receiver of the *requestReliability* message to the GM;
- sets the body of the message to the computed reliability level;
- sends the message to the GM, by using the specified at the ASIP tier *sendRequestReliability* function (see 5.2).

```
AECLASS AGM { // AE class
   AESELF_MANAGEMENT {
      OTHER_POLICIES {
         RELIABILITY_SELF_ASSESSMENT {
            FLUENT inChangeReliability {
               INITIATES isAnalysisDoneSent;
               TERMINATES: isReliabilityChanged;}
            FLUENT inRequestReliability {
               INITIATES: isRequestReliabilityReceived;
               TERMINATES: isRequestReliabilitySent;}
            MAPPING {
               CONDITION: inChangeReliability;
               ACTION:  ACTIONS.computeReliability;}
            MAPPING {
               CONDITION: inRequestReliability;
               ACTION:  ACTIONS.sendReliability;}
            MAPPING { // denies the new configuration
               CONDITION: EVENTS.isConfigurationDenied;
               ACTION:  ACTIONS.configurationDenied; }
         } // RELIABILITY_SELF_ASSESSMENT
      }
   } // AESELF_MANAGEMENT
   ACTIONS {
      ACTION computeReliability {
         GUARDS: isAnalysisDoneSent;
         ENSURES: isReliabilityChanged;
         DOES {
            set: AGMReliability = IMPL ComputeAGMReliability;
         }
          TRIGGERS: isConfigurationDenied;
      }
      ACTION sendReliability {
         GUARDS: isRequestReliabilityReceived;
         ENSURES: isRequestReliabilitySent;
         DOES {
            set: ASIP.MESSAGES.requestReliability.RECEIVER = GM;
          set: ASIP.MESSAGES.requestReliability.BODY = AGMReliability;
            call: ASIP.FUNCTIONS.sendRequestReliability;
         }
      }
      ACTION configurationDenied {
         GUARDS: isConfigurationDenied;
         DOES {
            call:  IMPL DenyAGMConfiguration;
         }
      }
   } // ACTIONS
   EVENTS {
      EVENT isAnalysisDoneSent:
         sent ASIP.MESSAGES.analysisDone ;
      EVENT isRequestReliabilityReceived:
         received ASIP.MESSAGES.requestReliability and
         ASIP.MESSAGES.requestReliability.ID = msgID and
         ASIP.MESSAGES.requestReliability.SENDER = GM;
      EVENT isRequestReliabilitySent:
         sent ASIP.MESSAGES.requestReliability;
      EVENT isReliabilityChanged: changed METRICS.AGMReliability;
      EVENT isConfigurationDenied;
   } // EVENTS
   METRICS {
      METRIC AGMReliability {
```

```
            type = QUALITY;
            description = "Measures the reliability of the AGM.";
            THRESHOLD_CLASS Reliability: double [min, max];
        }
    }
} // AECLASS AGM

// specify three similar AGMs by extending the AGM class
AE AGM1 extends AGM {};
AE AGM2 extends AGM {};
AE AGM3 extends AGM {};
```

Listing 4: AGM Partial Specification.

At the end of AGM class specification, we specify our three AGMs. They extend the AGM AE class.

6 CONCLUSION AND FUTURE WORK

One of the most important aspects of ASs is self-monitoring – a feature requiring a formal mechanism for self-diagnosis of AS status. In this paper, a system reliability self-assessment method is described for the diagnosis of potential reliability flaws, and consequently safety problems in evolving reactive ASs. A new formal specification language, ASSL, for specifying ASs has been applied to specify the AS-TRM and the reliability self-assessment. ASSL constitutes a hierarchical approach to specifying ASs where the low-level tiers express high-level detail structures of AEs, and the high-level tiers express a general architectural view of an AS. This exercise has demonstrated that ASSL is sufficiently generic and adaptable to accommodate most of an AS's aspects, thus allowing their specification not only at design time, but also during runtime (Vassev, Paquet, 2007).

Future research is concerned with modeling the effect of failure types and their assumed probabilities on the reliability computation. We will also explore rules for monitoring other non-functional requirements on system behavior, such as security, performance and trustability. There is a need to develop and analyze algorithms and negotiation protocols for conflicting non-functional requirements, and to determine what bidding or negotiation algorithms are the most effective. These are some of the issues that are expected to be tackled in the future.

REFERENCES

IBM Corporation, 2006. *An architectural blueprint for autonomic computing*, White Paper, 4th Edition.

Vassev, E., Paquet, J., 2007. ASSL - Autonomic System Specification Language, In *Proceedings of the 31st Annual IEEE/NASA Software Engineering Workshop (SEW-31)*, Baltimore, MD, USA.

Vassev, E., Paquet, J., 2007. Towards an Autonomic Element Architecture for ASSL, In *Proceedings of the 29th International Conference on Software Engineering / Software Engineering for Adaptive and Self-managing Systems (ICSE 2007 SEAMS)*, Minneapolis, MN, USA.

Goseva-Popstojanova, K., Kamavaram, S., 2004. Software Reliability Estimation under Uncertainty: Generalization of the Method of Moments, In *Proceedings of the 8th IEEE International Symposium on High Assurance Systems Engineering (HASE'04)*, pp. 209-218, Tampa, FL, USA.

Dai, Y., 2005. Autonomic Computing and Reliability Improvement, In *Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pp. 204-206, Seattle, WA, USA.

Zhang, T., Nakamura, M., 2005. Reliability-based Optimal Maintenance Scheduling by Considering Maintenance Effect to Reduce Cost, *Quality and Reliability Engineering*, International, 21:203–220.

Vassev, E., Kuang, H., Ormandjieva, O., Paquet, J., 2006. Reactive, Distributed and Autonomic Computing Aspects of AS-TRM, In *Proceedings of the 1st International Conference on Software and Data Technologies - ICSOFT'06*, pp. 196-202, Setubal, Portugal.

Ormandjieva, O., Kuang, H., Vassev, E., 2006, Reliability Self-Assessment in Reactive Autonomic Systems: Autonomic System-Time Reactive Model Approach, *International Transactions on Systems Science and Applications*, Volume 2 (1), pp.99-104.

IBM Tivoli, 2005. *Policy Management for Autonomic Computing – Version 1.2*, Tutorial, IBM Corp.

Tesauro, G., Chess, D., Walsh, W., Das, R., Whalley, I., Kephart, J., White, S., 2004. A multi-agent systems approach to autonomic computing, In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS'04)*, New-York, NY, USA.

Ormandjieva, O., 2002. *Deriving New Measurements for Real-Time Reactive Systems*. Ph.D. Thesis. Computer Science Department, Concordia University, Montreal, Canada.