PROTOTYPING SELF-HEALING BEHAVIOR FOR NASA SWARM-BASED SYSTEMS WITH ASSL

EMIL VASSEV AND MIKE HINCHEY

Lero, the Science Foundation Ireland Centre for Software
University of Limerick, Ireland

(received: 11 June 2021; revised: 31 August 2021;
accepted: 2 September 2021; published online: 30 October 2021)

Abstract: Autonomic computing promises computer systems capable of self-management, which augurs great promise for unmanned spacecraft. Such spacecraft are extremely appropriate for deep space exploration missions because the former bring onboard intelligence and less reliance on control links. The Autonomic System Specification Language (ASSL) is a framework for developing autonomic systems. As part of our research on ASSL, we have successfully specified a utonomic p roperties, v erified th eir co nsistency, an d ge nerated im plementation for both the NASA ANTS (Autonomous Nano-Technology Swarm) concept mission and the NASA Voyager mission. This paper presents concrete results on the use of ASSL to develop a self-healing behavior model for NASA ANTS swarm-based exploration missions. Here, we present specification a nd i mplementation r esults. M oreover, w e e xperiment w ith t he ASSL-generated code to demonstrate that the implemented ANTS system is capable of self-management in respect of the specified s elf-healing model.

Keywords: intelligent swarms, formal methods, autonomic computing, ASSL, ANTS
DOI: https://doi.org/10.34808/tq2021/25.2/d

1. Introduction

Nowadays NASA exploration missions increasingly rely on the concepts of autonomic computing (AC), exploiting these to increase the survivability of remote missions, particularly when human tending is not feasible. AC has emerged as a promising approach to the development of large-scale self-managing complex systems [1–3]. The general idea of AC is the handling of complexity in computer systems through self-management based on high-level objectives. Since its first announcement in 2001 [2], AC has inspired a tremendous number of initiatives for self-management of complex systems. Such an initiative is ASSL [4–6], where we approach the problem of formal specification, v alidation, a nd c ode g eneration of autonomic systems [7–11] within a single framework. Being an autonomic system

(AS), the NASA Autonomous Nano Technology Swarm (ANTS) [12–15] concept mission follows the principles of AC and provides self-management properties to ensure appropriate behavior and quality in the face of changing configurations and external conditions, based on automatic problem-determination algorithms. In the course of this research, we applied ASSL to specifying a self-healing behavior model for ANTS and subsequently to generate an operational Java application skeleton of the same. Note that although operational, the code generated by the ASSL framework is a skeleton; i.e., some parts are generated as empty methods and classes. The implementation results presented here are produced with the generated code only; i.e., without any additional implementation.

The remainder of this paper is organized as follows. In the next section we briefly present ASSL and ANTS and introduce the research problem, justification and impact, thus helping to familiarize the reader with the background technologies and our research goals. In Section 3, we describe the ASSL specification model for self-healing for ANTS. Section 4 presents the generated implementation for that model and test results. In Section 5, we assess our approach in terms of shortcomings and possible improvements. Section 6 overviews related work in the area of formal development. Finally, in Section 7 we summarize and conclude.

2. Background and Research Problem

In this section, we present both ASSL and ANTS as necessary background for the remaining content of the paper. In addition, we introduce the research problem we tackle in the course of this research.

2.1. ASSL

By its nature, the Autonomic System Specification Language (ASSL) [4, 5] provides both formal notation and tools for building software mechanisms for self-management of complex systems where the problem of formal specification, validation, and code generation of ASs is approached within a framework. Here, being a formal method dedicated to AC, ASSL helps AC researchers with problem formation, system design, system analysis and evaluation, and system implementation. A powerful and domain-specific formal notation is provided to specify required features and to model high-level models of ASs incorporating those features. Moreover, suitable mature tool support is provided to allow ASSL specifications to be edited and validated and Java code to be generated from any valid specification.

2.1.1. ASSL Specification Model

The ASSL formal notation [4–6] is based on a specification model exposed over hierarchically organized formalization tiers (cf. Table 1). This specification model provides both infrastructure elements and mechanisms needed by an AS. Each tier of the ASSL specification model is intended to describe different aspects of the AS in question, such as service-level objectives, self-management policies, interaction protocols, events, actions, autonomic elements, etc. This helps to specify an AS at different levels of abstraction (imposed by the ASSL tiers) where the AS in question is composed of special autonomic elements (AEs) interacting over special interaction protocols.

Table 1. ASSL Multi-tier Specification Model

AS		AS Service-Level Objectives
		AS Self-Management Policies
		AS Architecture
		AS Actions
		AS Events
		AS Metrics
ASIP		AS Messages
		AS Channels
		AS Functions
AE		AE Service-Level Objectives
		AE Self-Management Policies
		AE Friends
	AEIP	AE Messages
		AE Channels
		AE Functions
		AE Managed Elements
		AE Recovery Protocols
		AE Behavior Models
		AE Outcomes
		AE Actions
		AE Events
		AE Metrics

The AS Tier specifies an AS in terms of service-level objectives (AS SLO), self-management policies, architecture topology, actions, events, and metrics. The AS SLO is a high-level form of behavioral specification that establishes system objectives such as performance. The self-management policies could be the four self-management policies (the so-called self-CHOP) [1, 3] of an AS: self-configuring, self-healing, self-optimizing, and self-protecting, or they could be others. The metrics constitute a set of parameters and observables controllable by the AEs. At the AS Interaction Protocol tier, the ASSL framework specifies an AS-level interaction protocol (ASIP). ASIP is a public communication interface, expressed with channels, communication functions and messages. At the AE Tier, the ASSL formal model considers AEs to be analogous to software agents able to manage their own behavior and their relationships with other AEs. In this tier, ASSL describes the individual AEs of the AS.

2.1.2. Specifying ASs with ASSL

In general, it is not necessary to employ all of the ASSL tiers in order to model an AS. Instead, an ASSL specification must go around one or more self-management policies. This makes the ASSL specifications AC-driven and ASs are modeled taking into account the main goal of AC - self-management based on four main principles: self-configuring, self-healing, self-optimizing, and self-protecting (self-CHOP). ASSL addresses these self-CHOP principles as self-management policies specified at both AS and AE tiers (cf. Table 1). Policies are specified with special constructs called fluents and mappings [4, 5]:

· a fluent sets specific conditions determining when a self-management policy is activated;
· mappings map particular fluents to particular actions to be undertaken by the specified AS.

Fluents are expressed with fluent-activating and fluent-terminating events, i.e., the self-management policies are driven by events. In order to express mappings, conditions and actions are considered, where the former determine the latter in a deterministic manner. The following ASSL code presents a sample specification of a self-healing policy.

```
ASSELF_MANAGEMENT {
    SELF\_HEALING {
        FLUENT inLosingSpacecraft {
            INITIATED_BY { EVENTS.spaceCraftLost }
            TERMINATED_BY { EVENTS.earthNotified }
        }
        MAPPING {
            CONDITIONS { inLosingSpacecraft  }
            DO_ACTIONS { ACTIONS.notifyEarth }
        }
    }
} // ASSELF_MANAGEMENT
```

As shown, this policy is activated when a worker is lost and the system needs to notify Earth about this loss.

2.1.3. Managed Elements

An AE typically controls a managed resource specified with ASSL in the form of managed elements [4, 5], which are considered as being functional units (hardware or software) controlled by an AE. In an ASSL-developed AS, a managed element is specified with a set of special interface functions intended to provide control functionality over the managed resource. ASSL can specify and generate interfaces controlling a managed element (generated as a stub), but not the real implementation of these interfaces. Although this is just fine for prototyping, when deploying an AS the generated interfaces must be manually programmed to deal with the appropriate API of the managed resource.

2.2. NASA Swarm-based Exploration Missions

The Autonomous Nano Technology Swarm (ANTS) concept sub-mission PAM (Prospecting Asteroids Mission) is a novel approach to asteroid belt resource exploration [12–15]. By its nature, ANTS provides extremely high autonomy, minimal communication requirements with Earth, and a set of very small explorers with few consumables. These explorers, forming the swarm, are pico-class, low-power, and low-weight spacecraft units, yet capable of operating as fully autonomous and adaptable agents for multiple years in space.

2.2.1. Emergent Behavior

The agents in a swarm are able to interact with each other, thus helping them to self-organize based on the emergent behavior of the simple interactions [16]. The swarm exhibits self-organization since there is no external force directing its behavior and no single agent has a global view of the intended macroscopic behavior. Such type of behavior is observed in insects and flocks of birds. Bonabeau and Theraulaz [17], who studied self-organization in social insects, state that "complex collective behaviors may emerge from interactions among individuals that exhibit simple behaviors" and describe emergent behavior as "a set of dynamic mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components".

2.2.2. ANTS Organization

Figure 1 gives an overview of the organization of the ANTS concept mission. Here, each spacecraft in an ANTS swarm is equipped with a solar sail, which means it relies primarily on power from the Sun, using only tiny thrusters to navigate independently [14]. Moreover, each spacecraft also has onboard computation, AI (artificial intelligence), and heuristics systems for control at the individual and team levels. Spacecraft use low bandwidth to communicate within the swarm and high bandwidth for data transfer back to Earth [14].

As Figure 1 shows, ANTS teams consist of spacecraft units from three classes of spacecraft, and members in each class combine in certain ways to form teams that explore particular asteroids [12]:

- Workers, up to 80 percent of the swarm, bear the instruments and gather data. Instruments can include a magnetometer, x-ray, gamma-ray, visible/infrared, or neutral mass spectrometers. A worker gathers only its assigned data types.
- Rulers coordinate data gathering through the use of rules about what asteroid types and data are of interest.
- Messengers coordinate communications among the workers, rulers, and mission elements on Earth. Messengers, for example, can alert NASA to send replacement spacecraft from Earth or spacecraft with additional instruments.

The internal organization of an ANTS swarm depends on the global task to be performed and on the current environmental conditions. In general, a

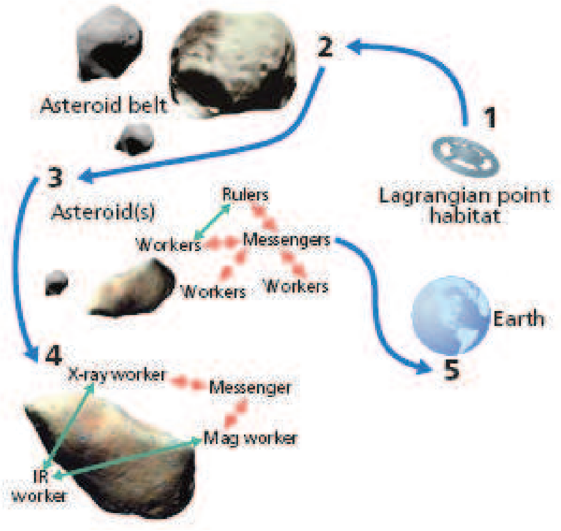


Figure 1. ANTS Mission Concept [12]

swarm consists of several sub-swarms, which are temporal teams organized to perform a particular task. Usually, a team comprises workers carrying a specialized instrument, a ruler playing the role of a team leader, and optionally one or more messengers.

2.2.3. ANTS Autonomic Properties

For ANTS, individual autonomy is not crucial, but the mission cannot succeed unless each team has all of the autonomic properties. There are four such properties, which by their nature do not have clear boundaries [12].

Self-configuration. ANTS must be able to adapt to changes in the system. Moreover, ANTS must be fully reconfigurable to support concurrent exploration and examination of hundreds of asteroids. Reconfiguration may also be required because of failure or anomaly of some sort.

Self-optimizing. ANTS must be able to improve their performance on the fly. Self-optimization is important to mission efficiency. Leaders can use the gained experience to self-optimize, thus improving their ability to identify asteroids. Messengers, strive to find the best position to improve the communication among the swarm units. Workers also self-optimize through learning and experience.

Self-healing. ANTS must be able to recover from errors or damage. Any ANTS unit, teams, sub-swarms, and the entire swarm must be able to recover from both mistakes and failures, including those caused by damage due either to a solar storm or to a collision with an asteroid or another spacecraft.

Self-protecting. ANTS must be to anticipate and cure intrusions. For example, ANTS must protect itself from solar storms, where charged particles can degrade sensors and electronic components, or destroy the solar sails.

On-going research, by the authors and others, is establishing other self-properties, often termed self-* properties.

2.3. Research Problem and Impact

In general, ANTS must afford autonomous operation without intervention from Earth, while operating under harsh conditions in space. ANTS poses many challenges related to its heterogeneous architecture, the need of continuous re-planning, re-configuration, and re-optimization. Thus, considering the hostile environment in which it must survive, we need to design and implement ANTS as a system able to perform an arbitrary number of in-space exploration tasks over multiple years and also able to autonomously manage itself, by integrating at least the baseline AC self-management policies: self-configuring, self-healing, self-optimizing and self-protecting (cf. Section 2.2.3). Therefore, the need for prototyping and formal modeling that will aid in the design and implementation of ANTS are becoming increasingly necessary and important as the urgent need for high levels of assurance regarding correctness and autonomic behavior persists in the ANTS requirements [16]. With ASSL, we are aiming at 1) modeling prototype models for ANTS's autonomic properties; 2) generating the implementation of these models; and 3) testing the autonomic behavior under simulated conditions. Prototype models for space-exploration systems (e.g., ANTS) can make tremendous social, technological and economic impact. Note that such prototypes may be used to perform relatively cheap in-lab experiments avoiding the risk of: 1) loss of life; 2) personal injury; 3) damage to the natural environment; 4) loss of important data; and 5) significant economic costs. Moreover, such prototypes help to find design and implementation flaws at early stages of software lifecycle, which helps to complete the overall system or project objectives.

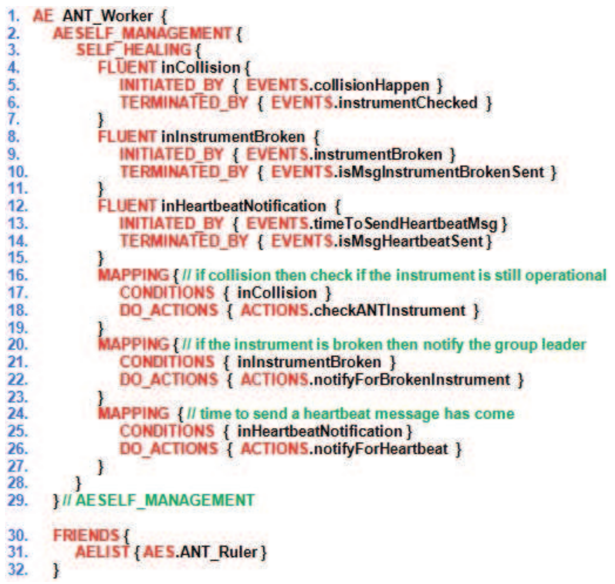
3. Self-healing Specification Model for ANTS

In ANTS, self-healing is about recovering from failures, including those caused by damage due to a crash or an outside force. In our scenario, we assume that each worker sends, on a regular basis, heartbeat messages to the ruler. The latter can use these messages to determine when a worker is not able to continue its operation, due to a crash or malfunction in its communication device. Moreover, a worker sends a notification message to the ruler if its instrument started malfunctioning or it has been broken, due to a crash with an asteroid or another spacecraft. Thus, a ruler is notified in two ways for a worker loss:

- a heartbeat message from the worker has not been received;
- a message from the worker, notifying for a broken instrument, has been received.

Once the loss of an operational unit has been detected, the ruler checks if the number of workers is below the critical minimum, and if so, it requests a replacement from another ruler. If such a replacement is not possible it may notify the ground control on Earth of the situation and may request a replacement or further instructions. Note that the current self-healing specification at the AS-tier (swarm level) handles situations, where a spacecraft unit is lost (cf.

Appendix A). An ASSL specification of the ANTS self-healing behavior requires a specification at the AS tier for the global ANTS behavior and at the AE tier for the self-healing behavior of every ANT_Worker and ANT_Ruler. Here we present the specification of the ANT_Worker. Please, cf. Appendix A for this specification at the AS tier and for thus of the ANT_Ruler. In order to specify the self-healing autonomic property of a worker, we use the SELF_HEALING self-management policy (cf. Figure 2). The self-healing policy is specified as a set of fluents and mappings (cf. Section 2.1.2), where the latter map the fluents to actions. Moreover, we specify the necessary actions (cf. Figure 3), events, metrics, and the AE interaction protocol (cf. Appendix A). The latter comprises the messages that can be exchanged among the worker and its ruler, the communication functions, and a communication channel - all needed by the self-healing policy.



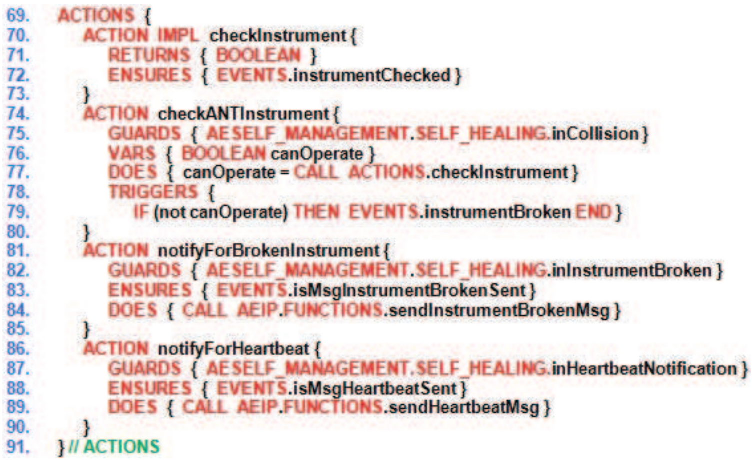
```
AE ANT_Worker {
  AESELF_MANAGEMENT {
    SELF_HEALING {
      FLUENT inCollision {
        INITIATED_BY { EVENTS.collisionHappen }
        TERMINATED_BY { EVENTS.instrumentChecked }
      }
      FLUENT inInstrumentBroken {
        INITIATED_BY { EVENTS.instrumentBroken }
        TERMINATED_BY { EVENTS.isMsgInstrumentBrokenSent }
      }
      FLUENT inHeartbeatNotification {
        INITIATED_BY { EVENTS.timeToSendHeartbeatMsg }
        TERMINATED_BY { EVENTS.isMsgHeartbeatSent }
      }
      MAPPING { // if collision then check if the instrument is still operational
        CONDITIONS { inCollision }
        DO_ACTIONS { ACTIONS.checkANTInstrument }
      }
      MAPPING { // if the instrument is broken then notify the group leader
        CONDITIONS { inInstrumentBroken }
        DO_ACTIONS { ACTIONS.notifyForBrokenInstrument }
      }
      MAPPING { // time to send a heartbeat message has come
        CONDITIONS { inHeartbeatNotification }
        DO_ACTIONS { ACTIONS.notifyForHeartbeat }
      }
    }
  } // AESELF_MANAGEMENT

  FRIENDS {
    AELIST { AES.ANT_Ruler }
  }
```

Figure 2. ANT_Worker Self-healing Policy Specification

In addition, at the AEIP tier, we specify a managed element (cf. Section 2.1.3) called worker, which provides a getDistanceToNearestObject interface function needed by the metric distanceToNearestObject to measure distance. Note that the ANT_Ruler is listed as a friend of the ANT_Worker, (cf. FRIENDS ... clause in Appendix A) and thus, it can use the ANT_Worker's private messages and channels (specified at the AEIP sub-tier). Note that this is a semantic rule in ASSL [4, 5]. The following elements reveal some details of the self-healing policy specification.

inCollision. This fluent takes place when the worker crashes into an asteroid or into another spacecraft, but is still able to perform self-checking operations. The fluent is initiated by a collisionHappen event, which is prompted immediately after a collision with another object. Moreover, this fluent terminates when the instrumentChecked event happens (cf. Figure 2), i.e., when the worker has performed an instrument-checking operation. Further, this fluent is mapped to the checkANTInstrument action (cf. Figure 3).



```
69.  ACTIONS {
70.     ACTION IMPL checkInstrument {
71.        RETURNS { BOOLEAN }
72.        ENSURES { EVENTS.instrumentChecked }
73.     }
74.     ACTION checkANTInstrument {
75.        GUARDS { AESELF_MANAGEMENT.SELF_HEALING.inCollision }
76.        VARS { BOOLEAN canOperate }
77.        DOES { canOperate = CALL ACTIONS.checkInstrument }
78.        TRIGGERS {
79.           IF (not canOperate) THEN EVENTS.instrumentBroken END }
80.     }
81.     ACTION notifyForBrokenInstrument {
82.        GUARDS { AESELF_MANAGEMENT.SELF_HEALING.inInstrumentBroken }
83.        ENSURES { EVENTS.isMsgInstrumentBrokenSent }
84.        DOES { CALL AEIP.FUNCTIONS.sendInstrumentBrokenMsg }
85.     }
86.     ACTION notifyForHeartbeat {
87.        GUARDS { AESELF_MANAGEMENT.SELF_HEALING.inHeartbeatNotification }
88.        ENSURES { EVENTS.isMsgHeartbeatSent }
89.        DOES { CALL AEIP.FUNCTIONS.sendHeartbeatMsg }
90.     }
91.  }// ACTIONS
```

Figure 3. ANT_Worker Self-healing Specification: Actions

The checkANTInstrument action uses an IMPL action to perform a check operation on the instrument. In case the instrument is not operational, an instrumentBroken event is prompted (cf. line 79 in Figure 3).

inInstrumentBroken. This fluent (cf. line 8 in Figure 2) is triggered when the instrumentBroken event is prompted (see the specification of the checkANTInstrument action), and terminates with the isMsgInstrumentBrokenSent event (cf. Appendix A). This event occurs when the instrumentBrokenMsg message, notifying of a broken instrument, is sent to the ruler (the notifyForBrokenInstrument action calls the function that sends this message). This message is specified in the AEIP.MESSAGES section (cf. Appendix A) together with the HBW_link channel and the sendInstrumentBrokenMsg function. The former is the ASSL specification of the HBW communication link [14] used for communication between the worker and its ruler.

inHeartbeatNotification. This fluent (cf. line 12 in Figure 2) is triggered when the timeToSendHeartbeatMsg event is fired. This event is a timed event, i.e., it is fired repeatedly after a particular amount of time has elapsed (in this case 1 minute). The fluent terminates with an isMsgHeartbeatSent event, which is fired when the heartbeat message is sent to the ruler. Moreover, this fluent is mapped to the notifyForHeartbeat action. This action uses the sendHeartbeatMsg AEIP function to send the heartbeatMsg message, over the HBW_link channel, to the ruler (cf. the AEIP specification section in Appendix A). Note that this action can be performed only if the self-healing policy is currently operating in the inHeartbeatNotification fluent (cf. the GUARDS clause of the notifyForHeartbeat action in Figure 3).

distanceToNearestObject. This metric is to measure the distance to the nearest object - an asteroid or a spacecraft unit. A threshold class is specified to define a minimum value acceptable by the metric. The collisionHappen event is fired when this metric has changed its value and the threshold class is not held anymore, i.e., the distance goes below the bare minimum, which is considered as a collision (cf. the GUARDS clause in the collisionHappen event). In addition, the metric source [4, 5] (METRIC_SOURCE clause) is attached to the getDistanceToNearestObject interface function of the worker managed element (cf. Appendix A). The latter specifies the interface needed by the metric to get that distance.

4. Implementation and Runtime Behavior

In this section, we discuss implementation results in terms of ASSL-generated code for the ASSL-specified self-healing model for ANTS and runtime self-management behavior. The behavior results presented here were obtained by evaluating the successfully generated code for the ASSL self-healing model for ANTS.

4.1. Code Generation Statistics

ASSL generates a Java code where the classes of an ASSL specification are grouped into hierarchically ordered Java packages. The ASSL framework generated 93 Java files for this specification (one per generated class or interface), which were distributed by the framework into 32 Java packages. The total number of generated lines of code including comments was 8159. Compared to the ASSL self-healing specification model for ANTS, with 293 lines of ASSL code, we specified the self-healing policy at three levels: 1) the AS tier level; 2) the ANT_Worker AE level; and 3) the ANT_Ruler AE level (cf. Appendix A). Therefore, the efficiency ratio in terms of lines of code (Java-generated code versus ASSL specification code) is:

$$28 \approx 8159/293$$

Thus, the ASSL code is significantly shorter, and hence more comprehensible, as one would expect in the case of an appropriate specification language for the domain.

4.2. Testing Self-healing Behavior

In this exercise, we experimented with the generated code for the ASSL self-healing specification model for ANTS (cf. Section 3). Note that by default, all Java application skeletons generated with the framework generate run-time log records [4, 5]. The latter show important state-transition operations ongoing in the system at runtime. Thus, we can easily trace the behavior of the generated system by following the log records generated by the same. Here we evaluated the log records produced by the generated Java application skeletons for three different versions of the ASSL self-healing specification model for ANTS. Thus, we modified the original version of the ANTS self-healing model to explore all aspects of the specified and generated self-healing behavior. The following subsections present test experiments performed with the code generated for three different versions of the ASSL self-healing specification model for ANTS.

4.2.1. Test 1: Original Specification

In this test, we generated the Java application skeleton for the original ASSL self-healing specification model for ANTS (cf. Appendix A), compiled the same with Java 1.6.0, and ran the compiled code. The application ran smoothly with no errors. First, it started all system threads as it is shown in the following log records. Note that starting all system threads first is a standard running procedure for all Java application skeletons generated with the ASSL framework.

Log Records "Starting System Threads"

```
**********************************************************
********************* INIT ALL TIERS *********************
**********************************************************
******************** START AS THREADS ********************
**********************************************************
1) METRIC 'generatedbyassl.as.aes.ant_ruler.metrics.DISTANCETONEARESTOBJECT': started
2) EVENT 'generatedbyassl.as.aes.ant_ruler.events.INSTRUMENTLOST': started
3) EVENT 'generatedbyassl.as.aes.ant_ruler.events.MSGINSTRUMENTBROKENRECEIVED': started
4) EVENT 'generatedbyassl.as.aes.ant_ruler.events.SPACECRAFTCHECKED': started
5) EVENT 'generatedbyassl.as.aes.ant_ruler.events.TIMETORECEIVEHEARTBEATMSG': started
6) EVENT 'generatedbyassl.as.aes.ant_ruler.events.INSTRUMENTOK': started
7) EVENT 'generatedbyassl.as.aes.ant_ruler.events.MSGHEARTBEATRECEIVED': started
8) EVENT 'generatedbyassl.as.aes.ant_ruler.events.RECONFIGURATIONDONE': started
9) EVENT 'generatedbyassl.as.aes.ant_ruler.events.RECONFIGURATIONFAILED': started
10) EVENT 'generatedbyassl.as.aes.ant_ruler.events.COLLISIONHAPPEN': started
11) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': started
12) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INCOLLISION': started
13) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INTEAMRECONFIGURATION': started
14) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INCHECKINGWORKERINSTRUMENT': started
15) POLICY 'generatedbyassl.as.aes.ant_ruler.aeself_management.SELF_HEALING':
 started
16) AE 'generatedbyassl.as.aes.ANT_RULER': started
**********************************************************
17) METRIC 'generatedbyassl.as.aes.ant_worker.metrics.DISTANCETONEARESTOBJECT': started
18) EVENT 'generatedbyassl.as.aes.ant_worker.events.ISMSGHEARTBEATSENT': started
19) EVENT 'generatedbyassl.as.aes.ant_worker.events.INSTRUMENTCHECKED': started
20) EVENT 'generatedbyassl.as.aes.ant_worker.events.ISMSGINSTRUMENTBROKENSENT': started
21) EVENT 'generatedbyassl.as.aes.ant_worker.events.COLLISIONHAPPEN': started
22) EVENT 'generatedbyassl.as.aes.ant_worker.events.INSTRUMENTBROKEN': started
23) EVENT 'generatedbyassl.as.aes.ant_worker.events.TIMETOSENDHEARTBEATMSG': started
24) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': started
25) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
ININSTRUMENTBROKEN': started
26) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INCOLLISION': started
27) POLICY 'generatedbyassl.as.aes.ant_worker.aeself_management.SELF_HEALING': started
28) AE 'generatedbyassl.as.aes.ANT_WORKER': started
**********************************************************
29) EVENT 'generatedbyassl.as.ants.events.SPACECRAFTLOST': started
```

```
30) EVENT 'generatedbyassl.as.ants.events.EARTHNOTIFIED': started
31) FLUENT 'generatedbyassl.as.ants.asself_management.self_healing.
INLOSINGSPACECRAFT': started
32) POLICY 'generatedbyassl.as.ants.asself_management.SELF_HEALING': started
33) AS 'generatedbyassl.as.ANTS': started
**********************************************************
***************** AS STARTED SUCCESSFULLY ****************
**********************************************************
```

Here, records 1 through to 16 show the ANT_RULER autonomic element startup, records 17 through to 28 show the ANT_WORKER autonomic element startup, and records 29 through to 33 show the last startup steps of the ANTS autonomic system. After starting up all the threads, the system ran in idle mode for 60 seconds, when the timed event timeToSendHeartbeatMsg occurred. This event is specified in the ANT_Worker to run on a regular time basis every 60 sec (cf. Appendix A). The occurrence of this event activated the self-healing mechanism as shown in the following log records.

Log Records "Self-healing Behavior - Original"

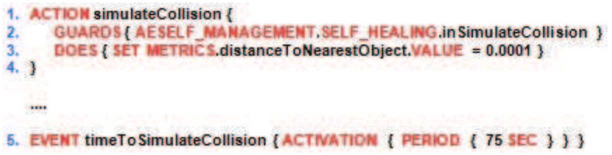
```
**********************************************************
***************** AS STARTED SUCCESSFULLY ****************
**********************************************************
34) EVENT 'generatedbyassl.as.aes.ant_worker.events.TIMETOSENDHEARTBEATMSG': has occurred
35) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been initiated
36) ACTION 'generatedbyassl.as.aes.ant_worker.actions.NOTIFYFORHEARTBEAT':
 has been performed
37) EVENT 'generatedbyassl.as.aes.ant_worker.events.ISMSGHEARTBEATSENT':
has occurred
38) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been terminated
39) EVENT 'generatedbyassl.as.aes.ant_ruler.events.TIMETORECEIVEHEARTBEATMSG':
 has occurred
40) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been initiated
41) ACTION 'generatedbyassl.as.aes.ant_ruler.actions.CONFIRMHEARTBEAT':
has been performed
42) EVENT 'generatedbyassl.as.aes.ant_ruler.events.MSGHEARTBEATRECEIVED':
 has occurred
43) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been terminated
44) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INCHECKINGWORKERINSTRUMENT': has been initiated
45) ACTION 'generatedbyassl.as.aes.ant_ruler.actions.CHECKWORKERINSTRSTATUS':
 has been performed
46) EVENT 'generatedbyassl.as.aes.ant_ruler.events.INSTRUMENTOK':
 has occurred
47) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INCHECKINGWORKERINSTRUMENT': has been terminated
48) EVENT 'generatedbyassl.as.aes.ant_worker.events.TIMETOSENDHEARTBEATMSG':
has occurred
49) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been initiated
50) ACTION 'generatedbyassl.as.aes.ant_worker.actions.NOTIFYFORHEARTBEAT':
has been performed
51) EVENT 'generatedbyassl.as.aes.ant_worker.events.ISMSGHEARTBEATSENT':
 has occurred
52) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been terminated
```

```
53) EVENT 'generatedbyassl.as.aes.ant_ruler.events.
TIMETORECEIVEHEARTBEATMSG':has occurred
54) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been initiated
55) EVENT 'generatedbyassl.as.aes.ant_worker.events.
TIMETOSENDHEARTBEATMSG':has occurred
56) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been initiated
57) ACTION 'generatedbyassl.as.aes.ant_ruler.actions.CONFIRMHEARTBEAT':
 has been performed
58) ACTION 'generatedbyassl.as.aes.ant_worker.actions.NOTIFYFORHEARTBEAT':
has been performed
59) EVENT 'generatedbyassl.as.aes.ant_ruler.events.MSGHEARTBEATRECEIVED':
has occurred
60) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been terminated
61) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INCHECKINGWORKERINSTRUMENT': has been initiated
62) EVENT 'generatedbyassl.as.aes.ant_worker.events.ISMSGHEARTBEATSENT':
has occurred
63) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been terminated
64) ACTION 'generatedbyassl.as.aes.ant_ruler.actions.CHECKWORKERINSTRSTATUS':
has been performed
65) EVENT 'generatedbyassl.as.aes.ant_ruler.events.INSTRUMENTOK':
 has occurred
66) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INCHECKINGWORKERINSTRUMENT': has been terminated
```

As we see from the log records, the self-healing behavior correctly followed the specification model. Records 34 through to 38 show the initiation and termination of the INHEARTBEATNOTIFICATION fluent. This resulted in the execution of the NOTIFYFORHEARTBEAT action (cf. record 36) that sends a heartbeat message to ANT_Ruler (cf. record 37). Records 39 through to 43 show how this message is handled by the ANT_Ruler. As it is specified (cf. Appendix A), the timeToReceiveHeartbeatMsg event occurred after 90 seconds running time (cf. record 39). This initiated the INHEARTBEATNOTIFICATION fluent (cf. record 40), which prompted the execution of the CONFIRMHEARTBEAT action (cf. record 41). The latter called a communication function to receive the heart beat message (if any). The message was received and prompted the MSGHEARTBEATRECEIVED event (cf. record 42). The latter terminated the INHEARTBEATNOTIFICATION fluent (cf. record 43). Records 44 through to 47 show how the INCHECKINGWORKERINSTRUMENT fluent is handled by the system. This fluent is initiated by the MSGHEARTBEATRECEIVED event (cf. Appendix A and record 44). Next the CHECKWORKERINSTRSTATUS action is performed (cf. record 45), which resulted into the INSTRUMENTOK event (cf. record 46). The latter terminated the INCHECKINGWORKERINSTRUMENT fluent (cf. record 47). Records 48 through to 66 show that the system continued repeating the steps shown in records 34 though to 47. This is because the policy-triggering events are periodic timed events and the system did not encounter any problems performing the executed actions, which could possibly branch the program execution. Note that records 48 through to 66 are not ordered in the same way as records 34 though to 47. This is due to both multithreading nature of the generated application and different periods of the timed events (60 sec and 90 sec). Thus, while the ANT_Ruler was handling the second heartbeat message (cf. record 53), the ANT_Worker was sending the third one (cf. record 55). This experiment demonstrated that the generated code had correctly followed the specified self-healing policy by reacting to the occurring self-healing events and, thus, providing appropriate self-healing behavior.

4.2.2. Test 2: Simulating Loss of Worker Instrument

In this test, we changed the original ASSL self-healing model for ANTS to simulate the loss of an instrument by the ANT_Worker. Thus, we specified a new inSimulateCollision fluent in the SELF_HEALING policy of the ANT_Worker autonomic element. In addition, we mapped the inSimulateCollision fluent to a newly specified simulateCollision action. The latter sets the value of the distanceToNearestObject metric to a number violating the metric's threshold class (cf. Figure 4). This causes the collisionHappen event attached to this metric to be prompted and consecutively to initiate the inCollision fluent.



```
ACTION simulateCollision {
    GUARDS { AESELF_MANAGEMENT.SELF_HEALING.inSimulateCollision }
    DOES { SET METRICS.distanceToNearestObject.VALUE = 0.0001 }
}

....

EVENT timeToSimulateCollision { ACTIVATION { PERIOD { 75 SEC } } }
```

Figure 4. Action simulateCollision Event timeToSimulateCollision

In addition, in order to initiate the inSimulateCollision fluent we specified a timeToSimulateCollision event. The latter is a timed event specified to occur on a regular time basis every 75 seconds (cf. Figure 4). Another change that we made in the specification model was in the checkANTInstrument action. We modified the action specification to report that the instrument is broken and to trigger the instrumentBroken event. The following log records show the run-time behavior of the new self-healing model for ANTS. Note that we omitted the startup part of the record, which we have already discussed in Test 1.

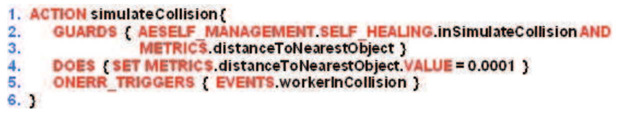
Log Record "Self-healing: Simulated Loss of Worker Instrument"

```
**********************************************************
***************** AS STARTED SUCCESSFULLY ****************
**********************************************************
34) EVENT 'generatedbyassl.as.aes.ant_worker.events.TIMETOSENDHEARTBEATMSG': has occurred
35) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been initiated
36) ACTION 'generatedbyassl.as.aes.ant_worker.actions.NOTIFYFORHEARTBEAT':
has been performed
37) EVENT 'generatedbyassl.as.aes.ant_worker.events.ISMSGHEARTBEATSENT': has occurred
38) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been terminated
39) EVENT 'generatedbyassl.as.aes.ant_worker.events.TIMETOSIMULATECOLLISION': has occurred
40) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INSIMULATECOLLISION': has been initiated
41) ACTION 'generatedbyassl.as.aes.ant_worker.actions.SIMULATECOLLISION':
```

```
has been performed
42) EVENT 'generatedbyassl.as.aes.ant_worker.events.COLLISIONHAPPEN': has occurred
43) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INCOLLISION': has been initiated
44) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INSIMULATECOLLISION': has been terminated
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
45) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
46) ACTION 'generatedbyassl.as.aes.ant_worker.actions.CHECKINSTRUMENT': has been performed
47) EVENT 'generatedbyassl.as.aes.ant_worker.events.INSTRUMENTCHECKED': has occurred
48) ACTION 'generatedbyassl.as.aes.ant_worker.actions.CHECKANTINSTRUMENT':
has been performed
49) EVENT 'generatedbyassl.as.aes.ant_worker.events.
INSTRUMENTBROKEN': has occurred
50) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
ININSTRUMENTBROKEN': has been initiated
51) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INCOLLISION': has been terminated
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
52) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
53) ACTION 'generatedbyassl.as.aes.ant_worker.actions.NOTIFYFORBROKENINSTRUMENT':
has been performed
54) EVENT 'generatedbyassl.as.aes.ant_worker.events.ISMSGINSTRUMENTBROKENSENT':
has occurred
55) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
ININSTRUMENTBROKEN': has been terminated
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
56) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
57) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
58) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
59) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
60) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
61) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
62) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
63) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
64) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
65) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
66) EVENT 'generatedbyassl.as.aes.ant_ruler.events.TIMETORECEIVEHEARTBEATMSG':
 has occurred
67) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been initiated
```

```
68) ACTION 'generatedbyassl.as.aes.ant_ruler.actions.CONFIRMHEARTBEAT': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
69) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
70) EVENT 'generatedbyassl.as.aes.ant_ruler.events.MSGHEARTBEATRECEIVED':
 has occurred
71) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been terminated
72) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INCHECKINGWORKERINSTRUMENT': has been initiated
73) ACTION 'generatedbyassl.as.aes.ant_ruler.actions.CHECKWORKERINSTRSTATUS':
has been performed
74) EVENT 'generatedbyassl.as.aes.ant_ruler.events.INSTRUMENTOK': has occurred
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
75) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
76) EVENT 'generatedbyassl.as.aes.ant_ruler.events.MSGINSTRUMENTBROKENRECEIVED':
has occurred
77) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INCHECKINGWORKERINSTRUMENT': has been terminated
78) EVENT 'generatedbyassl.as.aes.ant_ruler.events.INSTRUMENTLOST': has occurred
79) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INTEAMRECONFIGURATION': has been initiated
80) ACTION 'generatedbyassl.as.aes.ant_ruler.actions.RECONFIGURETEAM':
has been performed
81) EVENT 'generatedbyassl.as.aes.ant_ruler.events.RECONFIGURATIONDONE':
 has occurred
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
82) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
83) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INTEAMRECONFIGURATION': has been terminated
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
84) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
85) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
```

The following text explains records 34 through to 85. Records 34 through to 38 are identical with those in Test 1. Records 39 through to 44 show the initiation and termination of the INSIMULATECOLLISION fluent. This fluent was initiated by the TIMETOSIMULATECOLLISION event (cf. record 39) and prompted the execution of the SIMULATECOLLISION action (cf. record 41). Next, due to that action changing the distanceToNearestObject metric's value, the COLLISIONHAPPEN event was triggered (cf. record 42). This event terminated the INSIMULATECOLLISION fluent (cf. record 44) and initiated the INCOLLISION fluent (cf. record 43). Records 45, 52, 56 through to 65, 69, 75, 82, 84, and 85 show that the control loop of the ANT_Worker uncovered a problem with the metric DISTANCETONEARESTOBJECT, and attempted to fix that problem by executing actions. Because, there was no action set to fix the metric, the control loop executed a generic action that simply prints a message highlighting that problem (this is the default for all control loops generated with the ASSL framework [4, 5]). In ASSL, control loops monitor and work to fix metrics and service-level-objectives (SLO) of the system. Here, the



```
ACTION simulateCollision{
   GUARDS { AESELF_MANAGEMENT.SELF_HEALING.inSimulateCollision AND
                METRICS.distanceToNearestObject }
   DOES { SET METRICS.distanceToNearestObject.VALUE = 0.0001 }
   ONERR_TRIGGERS { EVENTS.workerInCollision }
}
```

Figure 5. Action simulateCollision (Modified)

DISTANCETONEARESTOBJECT metric was discovered as invalid and thus needed to be fixed, because its current value was violating the metric's threshold class. The presence of multiple records of the same type shows that the control loop was constantly trying to fix that problem. Records 46 through to 51 show the process of checking the ANT_Worker instrument. This resulted in prompting the INSTRUMENTBROKEN event (cf. record 49) and consecutively initiating the ININSTRUMENTBROKEN fluent (cf. record 50). Next, this fluent prompted the action NOTIFYFORBROKENINSTRUMENT (cf. record 53). Records 66, 67, 68, 70, and 71 show that the ANT_Ruler received the heartbeat message sent by the ANT_Worker (cf. records 34 through to 38). Records 72, 73, 74, and 77 show the instrument check performed by the ANT_Ruler after receiving the heartbeat message. Note that this check reported the INSTRUMENTOK event (cf. record 74) because the check was based on the heartbeat message sent before the collision. Record 76 shows that the ANT_Ruler received at that point the message sent by the ANT_Worker and notifying that the worker instrument is broken. This prompted the INSTRUMENTLOST event (cf. record 78) and consecutively initiated the INTEAMRECONFIGURATION fluent (cf. record 79). The latter prompted the execution of the RECONFIGURETEAM action (cf. record 80), which finished with prompting the RECONFIGURATIONDONE event (cf. record 81). Similar to Test 1, this experiment demonstrated that the generated code had correctly followed the modified self-healing policy by reacting as before to the occurring self-healing events and thus, providing appropriate self-healing behavior.

4.2.3. Test 3: Simulating Worker Loss

In this test, we changed the original ASSL self-healing model for ANTS to simulate loss of an instrument by the ANT_Worker. Thus, we specified a new inSimulateCollision fluent in the SELF_HEALING policy of the ANT_Worker autonomic element. In this test, we changed the ASSL self-healing model for ANTS from Test 2 to simulate loss of the ANT_Worker. The changes we made in the specification code are as following:

- We set the activation time of the timeToSimulateCollision timed event to 45 seconds, thus simulating a collision before sending the heartbeat message (every 60 seconds).
- We changed the GUARDS clause of the simulateCollision action (cf. Figure 5) to ensure that this action will be performed only once. Thus, we added to that clause the evaluation of the distanceToNearestObject metric, i.e., the action could not perform if that metric is invalid (holds a value that contradicts its threshold class) [4, 5].

Similarly, we changed the GUARDS clause of both the notifyForHeartbeat action and the checkANTInstrument action. This prevented both actions from executing once the distanceToNearestObject metric became invalid. The following log records show the run-time behavior of the modified self-healing model for ANTS. Note that startup part of the records (discussed in Test 1) is omitted here.

Log Record "Self-healing with Simulated Worker Loss"

```
**********************************************************
***************** AS STARTED SUCCESSFULLY ****************
**********************************************************
34) EVENT 'generatedbyassl.as.aes.ant_worker.events.TIMETOSIMULATECOLLISION': has occurred
35) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INSIMULATECOLLISION': has been initiated
36) ACTION 'generatedbyassl.as.aes.ant_worker.actions.SIMULATECOLLISION':
has been performed
37) EVENT 'generatedbyassl.as.aes.ant_worker.events.COLLISIONHAPPEN': has occurred
38) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INCOLLISION': has been initiated
39) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INSIMULATECOLLISION': has been terminated
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
40) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
41) ACTION 'generatedbyassl.as.aes.ant_worker.actions.CHECKANTINSTRUMENT':
has been prevented by GUARDS
42) EVENT 'generatedbyassl.as.aes.ant_worker.events.INSTRUMENTNOTCHECKED':
has occurred
43) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INCOLLISION': has been terminated
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
44) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
45) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
46) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
47) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
48) EVENT 'generatedbyassl.as.aes.ant_worker.events.TIMETOSENDHEARTBEATMSG':
has occurred
49) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been initiated
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
50) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
51) ACTION 'generatedbyassl.as.aes.ant_worker.actions.NOTIFYFORHEARTBEAT':
has been prevented by GUARDS
52) EVENT 'generatedbyassl.as.aes.ant_worker.events.HEARTBEATMSGNOTSENT':
has occurred
53) FLUENT 'generatedbyassl.as.aes.ant_worker.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been terminated
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
54) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
```

```
55) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
56) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
57) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
58) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
59) EVENT 'generatedbyassl.as.aes.ant_ruler.events.TIMETORECEIVEHEARTBEATMSG':
has occurred
60) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been initiated
61) ACTION 'generatedbyassl.as.aes.ant_ruler.actions.CONFIRMHEARTBEAT':
has failed to fulfill ENSURES post-conditions
62) EVENT 'generatedbyassl.as.ants.events.SPACECRAFTLOST': has occurred
63) FLUENT 'generatedbyassl.as.aes.ant_ruler.aeself_management.self_healing.
INHEARTBEATNOTIFICATION': has been terminated
64) FLUENT 'generatedbyassl.as.ants.asself_management.self_healing.
INLOSINGSPACECRAFT': has been initiated
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
65) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
66) ACTION 'generatedbyassl.as.ants.actions.NOTIFYEARTH': has been performed
There is no action set to fix the invalid metric generatedbyassl.as.aes.ant_worker.
metrics.DISTANCETONEARESTOBJECT'
67) ACTION 'generatedbyassl.as.ASSLACTION': has been performed
68) EVENT 'generatedbyassl.as.ants.events.EARTHNOTIFIED': has occurred
69) FLUENT 'generatedbyassl.as.ants.asself_management.self_healing.
INLOSINGSPACECRAFT': has been terminated
```

These log records again show that the generated Java application skeleton provided correct behavior conforming to the specified self-healing policy. This time, the first event that occurred in the system was the TIMETOSIMULATE-COLLISION event (cf. record 34). The latter initiated the INSIMULATECOLLISION fluent, which as in Test 2 simulated a collision in ANT_Worker (cf. records 35 through to 39). This initiated the INCOLLISION fluent (cf. record 38), but his time the CHECKANTINSTRUMENT action did not execute due to its modified GUARDS clause (cf. record 41). Similarly, the NOTIFYFORHEARTBEAT action did not execute due to its GUARDS clause (cf. record 51) and thus, no heartbeat message was sent (cf. record 52) to the ANT_Ruler. Records 59 through to 63 show that the ANT_Ruler unable to receive that heartbeat message triggered the SPACECRAFTLOST AS-level event (cf. record 62). The latter initiated the ANTS INLOSINGSPACECRAFT fluent, which notified Earth about the problem (cf. records 64, 66, 68, and 69). Note that similar to Test 2, the control loop of the ANT_Worker was constantly trying to fix the invalid metric (cf. records 40, 44, 45, etc.).

5. Discussion

In this section, we assess the completeness of the ASSL-developed self-healing for ANTS and the effectiveness of using ASSL to model this model. It is important to mention, that these tests (Test 1, 2, and 3 – cf. Section 4.2) not only provided strong evidence of valid self-management behavior of the generated code, but also demonstrated the ASSL communication system. Here, messages were successfully sent from one autonomic element (ANT_Worker) and received by another one (ANT_Ruler). In addition, we have demonstrated the effectiveness of the event-driven self-management policy model, where ASSL events can be associated with messages, metrics, other events, time etc. These events initiate and terminate fluents. The latter prompt the execution of actions. Moreover, we have demonstrated the effectiveness of the ASSL secure action approach. With conditions specified in the action GUARDS and ENSURES clauses we require certain conditions to be met before and after the action's execution. In our current self-healing model for ANTS, we specify the self-healing policy only from the worker's viewpoint. For a complete specification, we need to specify this policy also on the ruler's side and for the entire swarm (AS tier). Moreover, the instrument checking operation should check also for the instrument's performance; i.e., the instrument can be still operational but its performance can be degraded. This will allow self-optimization, where low performing workers will be replaced with high performing ones. In addition, in order to complete the model, we also need to specify self-checking on the worker's navigation and communication systems, and self-testing of the worker's computational unit. Part of the self-healing process could be assigning a new worker with an identical instrument to the team when a malfunctioning worker has been discovered. This will prompt self-configuration. Moreover, as is stated in [12], a worker with a malfunctioning instrument can be transformed into a ruler. This can be specified in the self-healing policy as an increase in the total number of rulers and as a decrease in the total number of workers. This can be handled by metrics conscious of the number of rulers and the number of workers in the entire swarm and for each team. Another shortcoming here is that the self-healing model does not take into consideration recovery from mistakes, e.g., position displacement. A better specification shall include all the possible mistakes per spacecraft and their appropriate recovery actions or intrinsically specified recovery protocol [4, 5].

6. Related Work

A NASA developed formal approach, named R2D2C (Requirements to Design to Code) is described in [18]. In this approach, system designers may write specifications as scenarios in constrained (domain-specific) natural language, or in a range of other notations (including UML use cases). These scenarios are then used to derive a formal model that fulfills the requirements stated at the outset, and which is subsequently used as a basis for code generation. R2D2C relies on a variety of formal methods to express the formal model under consideration. The latter can be used for various types of analysis and investigation, and as the basis for fully formal implementations as well as for use in automated test case generation. IBM has developed a framework called Policy Management for AC (PMAC) [19] that provides a standard model for the definition of policies and an environment for the development of software objects that hold and evaluate policies. For writing and storing policies, PMAC uses a declarative XML-based language called AC Policy Language (ACPL) [19, 20]. A policy written in ACPL provides an XML specification defining the following elements:

- condition - when a policy is to be applied;
- decision - observable behavior or desired outcome;
- result - a set of named and typed data values;
- action - invokes an operation;
- configuration profile - unifies result and action;
- business value - the relative priority of a policy;
- scope - the subject of the policy.

The basis of ACPL is the AC Expression Language (ACEL) [19, 20]. ACEL is an XML-based language developed to describe conditions when a policy should be applied to a managed system.

7. Conclusions

In the most basic of terms, experiments are said to be valid if they do what they are supposed to do. In that context, the experiments and test results described here are valid and they conform to our belief that ASSL framework provides a valid approach for building and validating autonomic systems. Unfortunately, it is far easier to demonstrate validity of our approach than to demonstrate conclusively its completeness. In part, this is because completeness is at heart a relative rather than an absolute concept. Therefore, more experiments and results are needed and it is our intention to come up with a more complete ASSL specification model for ANTS emphasizing different autonomic features and to consecutively generate a more complete Java application skeleton for ANTS. Next, we will complete that generated skeleton to arrive at the first experimental prototype of ANTS. The latter could be extremely useful when undertaking further investigation based on practical results and will help us to test different aspects of autonomic behavior under more simulated conditions.

Acknowledgement

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094_P2 and co-funded under the European Regional Development Fund through the Southern Eastern Regional Operational Programme to Lero - the Science Foundation Ireland Research Centre for Software (www.lero.ie), by the NASA Software Assurance Research Program (SARP), and NASA Goddard Space Flight Center.

```
Appendix A: ASSL Self-healing Model for ANTS

//==== autonomic system ANTS - SELF-HEALING =================
//============== original specification ===================
AS ANTS {
    ASSELF_MANAGEMENT {
        SELF_HEALING {
            FLUENT inLosingSpacecraft {
                INITIATED_BY { EVENTS.spaceCraftLost }
                TERMINATED_BY { EVENTS.earthNotified }
            }
            MAPPING {
                CONDITIONS { inLosingSpacecraft }
                DO_ACTIONS { ACTIONS.notifyEarth }
            }
        }
    } // ASSELF_MANAGEMENT

    ASARCHITECTURE {
        AELIST {AES.ANT_Worker, AES.ANT_Ruler}
        DIRECT_DEPENDENCIES {
            DEPENDENCY AES.ANT_Worker { AES.ANT_Ruler }
        }
        TRANSITIVE_DEPENDENCIES {
            DEPENDENCY AES.ANT_Ruler  {AES.ANT_Worker }
        }
        GROUPS {
            GROUP explorerOne {
                MEMBERS { AES.ANT_Worker, AES.ANT_Ruler }
                COUNCIL { AES.ANT_Ruler }
            }
        }
    }

    ACTIONS {
        ACTION notifyEarth { //notify Earth for the lost spacecraft
            GUARDS {
ASSELF_MANAGEMENT.SELF_HEALING.inLosingSpacecraft }
            DOES { CALL ASIP.FUNCTIONS.sendSpacecraftLostMsg }
        }
    }

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

} // AS ANTS

//==================== AS interaction protocol ============
ASIP {
    MESSAGES {
        MESSAGE msgSpacecraftLost {
            SENDER { ANY }
            RECEIVER { ANY }
            PRIORITY { 1 }
            MSG_TYPE {    TEXT }
            BODY { "lost spacecraft" }
        }
    }
    CHANNELS {
        CHANNEL LBW_link {
            ACCEPTS { ASIP.MESSAGES.msgSpacecraftLost }
            ACCESS { SEQUENTIAL }
            DIRECTION { INOUT } }
        }
        FUNCTIONS {
            FUNCTION sendSpacecraftLostMsg {
                DOES { ASIP.MESSAGES.msgSpacecraftLost
                >> ASIP.CHANNELS.LBW_link }
            }
        }
    }

    //==================== autonomic elements ===================
    AES {
        //==================== ANT_Worker =======================
        AE ANT_Worker {
             AESELF_MANAGEMENT {
                SELF_HEALING {
                    FLUENT inCollision {
                        INITIATED_BY  {  EVENTS.collisionHappen  }
                        TERMINATED_BY  {  EVENTS.instrumentChecked  }
                    }
                    FLUENT inInstrumentBroken  {
                        INITIATED_BY  {  EVENTS.instrumentBroken  }
                        TERMINATED_BY  {  EVENTS.isMsgInstrumentBrokenSent  }
                    }
                    FLUENT inHeartbeatNotification  {
                        INITIATED_BY  {  EVENTS.timeToSendHeartbeatMsg }
                        TERMINATED_BY  {  EVENTS.isMsgHeartbeatSent }
                    }
                    MAPPING {
                        // if collision then check if the instrument is still operational
                        CONDITIONS  {  inCollision  }
                        DO_ACTIONS  {  ACTIONS.checkANTInstrument  }
                    }
                    MAPPING {
                        // if the instrument is broken then notify the group leader
                        CONDITIONS  {  inInstrumentBroken  }
                        DO_ACTIONS  {  ACTIONS.notifyForBrokenInstrument  }
                    }
                    MAPPING  {
                        // time to send a heartbeat message has come
                        CONDITIONS  {  inHeartbeatNotification }
                        DO_ACTIONS  {  ACTIONS.notifyForHeartbeat  }
                     }
                }
            } // AESELF_MANAGEMENT

            FRIENDS {
                AELIST { AES.ANT_Ruler }
            }

            AEIP {
                MESSAGES {
                    FINAL MESSAGE instrumentBrokenMsg  {
                        SENDER  { AES.ANT_Worker  }
                        RECEIVER  { AES.ANT_Ruler  }
                        MSG_TYPE  { TEXT }
                        BODY { "broken instrument" }
                    }
                    FINAL MESSAGE heartbeatMsg  {
                        SENDER  { AES.ANT_Worker }
                        RECEIVER  { AES.ANT_Ruler }
                        MSG_TYPE  { TEXT }
                        BODY  { "alive" }
            }
          }
        CHANNELS {
            CHANNEL HBW\_link  {
                ACCEPTS  { AEIP.MESSAGES.instrumentBrokenMsg ,
                 AEIP.MESSAGES.heartbeatMsg  }
                ACCESS  { SEQUENTIAL  }
                DIRECTION  { INOUT }
            }
        }
        FUNCTIONS {
            FUNCTION sendInstrumentBrokenMsg  {
                  DOES {  AEIP.MESSAGES.instrumentBrokenMsg  >>
                AEIP.CHANNELS.HBW_link  }
            }
            FUNCTION sendHeartbeatMsg  {
                DOES {  AEIP.MESSAGES.heartbeatMsg  >>  AEIP.CHANNELS.HBW_link }
            }
        }
        MANAGED_ELEMENTS {
            MANAGED_ELEMENT worker {
                INTERFACE_FUNCTION getDistanceToNearestObject {
                     RETURNS { DECIMAL }
                 }
            }
        }
    } // AEIP

    ACTIONS {
        ACTION IMPL checkInstrument {
            RETURNS {  BOOLEAN  }
            TRIGGERS {  EVENTS.instrumentChecked }
        }
        ACTION checkANTInstrument {
            GUARDS {AESELF_MANAGEMENT.SELF_HEALING.inCollision}
    VARS { BOOLEAN canOperate }
            DOES { canOperate = CALL ACTIONS.checkInstrument }
            TRIGGERS {
                IF (not canOperate) THEN EVENTS.instrumentBroken END
            }
        }
        ACTION notifyForBrokenInstrument {
            GUARDS { AESELF_MANAGEMENT.SELF_HEALING.inInstrumentBroken }
            DOES { CALL AEIP.FUNCTIONS.sendInstrumentBrokenMsg }
        }
        ACTION notifyForHeartbeat {
            GUARDS { AESELF_MANAGEMENT.SELF_HEALING.inHeartbeatNotification }
            DOES { CALL AEIP.FUNCTIONS.sendHeartbeatMsg }
        }
    } // ACTIONS

    EVENTS { // these events are used in the fluent specifications
        EVENT collisionHappen {
            GUARDS { not METRICS.distanceToNearestObject }
            ACTIVATION  {
                CHANGED  { METRICS.distanceToNearestObject }
             }
        }
        EVENT isMsgInstrumentBrokenSent {
            ACTIVATION  {
                SENT { AEIP.MESSAGES.instrumentBrokenMsg }
            }
        }
        EVENT instrumentBroken { }
        EVENT instrumentChecked { }
        EVENT timeToSendHeartbeatMsg {
            ACTIVATION  {  PERIOD  {  1 min }     }
        }
        EVENT isMsgHeartbeatSent {
            ACTIVATION  {
                SENT { AEIP.MESSAGES.heartbeatMsg }
            } } } // EVENTS

    METRICS {
        METRIC distanceToNearestObject  {
            METRIC_TYPE  {  RESOURCE  }
            METRIC_SOURCE {     AEIP.MANAGED_ELEMENTS.worker.
                           getDistanceToNearestObject }
            DESCRIPTION  { "measures the distance to the nearest space object" }
            MEASURE_UNIT  { "KM" }
            VALUE {    100 }
            THRESHOLD_CLASS  {  DECIMAL  [0.001 ~ )  }
         }
    }

} // AE ANT_Worker

//==================== ANT_Ruler ========================    AE ANT_Ruler {

     AESELF_MANAGEMENT {
        SELF_HEALING {
            FLUENT inCollision {
                INITIATED_BY { EVENTS.collisionHappen }
                TERMINATED_BY { EVENTS.spacecraftChecked, AS.EVENTS.spaceCraftLost }
            }
            FLUENT inHeartbeatNotification {
                INITIATED_BY { EVENTS.timeToReceiveHeartbeatMsg }
                TERMINATED_BY { EVENTS.msgHeartbeatReceived,
                      AS.EVENTS.spaceCraftLost }
            }
            FLUENT inCheckingWorkerInstrument {
                INITIATED_BY { EVENTS.msgHeartbeatReceived }
                TERMINATED_BY { EVENTS.instrumentOK, EVENTS.instrumentLost }
            }
            FLUENT inTeamReconfiguration {
                INITIATED_BY { EVENTS.instrumentLost }
                TERMINATED_BY { EVENTS.reconfigurationDone,
                      EVENTS.reconfigurationFailed }
            }
            MAPPING {
                // if collision then check if the spacecraft is still operational
                CONDITIONS  { inCollision }
                DO_ACTIONS  { ACTIONS.checkSpacecraft }
            }
            MAPPING  {
                // time to receive a heartbeat message from the worker
                CONDITIONS  { inHeartbeatNotification }
                DO_ACTIONS  { ACTIONS.confirmHeartbeat }
             }
            MAPPING  {
                // time to check for a "worker broken instrument" message
      CONDITIONS  { inCheckingWorkerInstrument }
                DO_ACTIONS  { ACTIONS.checkWorkerInstrStatus }
             }
```

```
                MAPPING  {
                    // need to adapt to the new situation
                    CONDITIONS  { inTeamReconfiguration }
                    DO_ACTIONS  { ACTIONS.reconfigureTeam }
                 }
            }
        } // AESELF_MANAGEMENT

        AEIP {
            FUNCTIONS {
                FUNCTION receiveHeartbeatMsg  {
                     DOES { AES.ANT_Worker.AEIP.MESSAGES.heartbeatMsg
                  << AES.ANT_Worker.AEIP.CHANNELS.HBW_link }
                }
                FUNCTION receivedInstrumentBrokenMsg  {
                      DOES { AES.ANT_Worker.AEIP.MESSAGES.instrumentBrokenMsg
                  << AES.ANT_Worker.AEIP.CHANNELS.HBW_link }
                }
            }
            MANAGED_ELEMENTS {
                MANAGED_ELEMENT ruler {
                    INTERFACE_FUNCTION getDistanceToNearestObject    {
                        RETURNS { DECIMAL }
                    }
                } } } // AEIP
        ACTIONS {
            ACTION IMPL checkSpacecraft {
                TRIGGERS { EVENTS.spacecraftChecked }
                ONERR_TRIGGERS { AS.EVENTS.spaceCraftLost }
            }
            ACTION confirmHeartbeat {
                GUARDS { AESELF_MANAGEMENT.SELF_HEALING.inHeartbeatNotification }
                DOES { CALL AEIP.FUNCTIAS
     AS Service-Level Objectives
     AS Self-Management Policies
     AS Architecture
     AS Actions
     AS Events
     AS Metrics
     ASIP
     AS Messages
     AS Channels
     AS Functions
     AE
     AE Service-Level Objectives
     AE Self-Management Policies
     AE Friends
     AEIP
     AE Messages
     AE Channels
     AE Functions
     AE Managed Elements
     AE Recovery Protocols
     AE Behavior Models
     AE Outcomes
     AE Actions
     AE Events
     AE MetricsNS.receiveHeartbeatMsg }
                ONERR_TRIGGERS { AS.EVENTS.spaceCraftLost }
            }
            ACTION checkWorkerInstrStatus {
                GUARDS { AESELF_MANAGEMENT.SELF_HEALING.inCheckingWorkerInstrument }
        DOES { CALL AEIP.FUNCTIONS.receivedInstrumentBrokenMsg }
TRIGGERS {
            IF EVENTS.msgInstrumentBrokenReceived THEN
                EVENTS.instrumentLost
            END ELSE
                EVENTS.instrumentOK
            END
        }
    }
    ACTION IMPL reconfigureTeam {
        GUARDS { AESELF_MANAGEMENT.SELF_HEALING.inTeamReconfiguration }
        TRIGGERS { EVENTS.reconfigurationDone }
        ONERR_TRIGGERS { EVENTS.reconfigurationFailed }
    }
}

EVENTS { // these events are used in the fluents? specification
    EVENT collisionHappen {
        GUARDS { not METRICS.distanceToNearestObject }
        ACTIVATION  {
            CHANGED  { METRICS.distanceToNearestObject }
         }
    }
    EVENT spacecraftChecked { }
    EVENT timeToReceiveHeartbeatMsg {
        ACTIVATION  {  PERIOD  {  90 sec } }
    }
    EVENT msgHeartbeatReceived {
        ACTIVATION  {
            RECEIVED {
                AES.ANT_Worker.AEIP.MESSAGES.heartbeatMsg
            }
        }
    }
    EVENT msgInstrumentBrokenReceived {
        ACTIVATION  {
            RECEIVED {
                AES.ANT_worker.AEIP.MESSAGES.instrumentBrokenMsg
            }
         }
    }
    EVENT instrumentOK { }
    EVENT instrumentLost {
        ACTIVATION  {
            OCCURRED { EVENTS.msgInstrumentBrokenReceived }
        }
    }
    EVENT reconfigurationDone { }
    EVENT reconfigurationFailed { }
} // EVENTS

METRICS {
    METRIC distanceToNearestObject  {
        METRIC_TYPE  {  RESOURCE  }
        METRIC_SOURCE {     AEIP.MANAGED_ELEMENTS.ruler
                .getDistanceToNearestObject }
        DESCRIPTION  { "measures the distance to the nearest space object" }
        MEASURE_UNIT  { "KM" }
        VALUE { 100 }
        THRESHOLD_CLASS  {  DECIMAL  [0.001 ~ )  }
     }
}
```

```
        } // AE ANT_Ruler

    } // AES
```

References

[1] IBM Corporation 2006 *An Architectural Blueprint for Autonomic Computing (4h ed.)*, White paper

[2] Horn P 2001 *Autonomic Computing: IBM's Perspective on the State of Information Technology*, *Proceedings of the IEEE*, IBM T. J. Watson Laboratory

[3] Murch R 2004 *Autonomic Computing: On Demand Series*, *Proceedings of the IEEE*, IBM Press

[4] Vassev E 2008 *Towards a Framework for Specification and Code Generation of Autonomic Systems - Ph.D Thesis*, Department of Computer Science and Software Engineering, Concordia University

[5] Vassev E 2009 *ASSL: Autonomic System Specification Language - A Framework for Specification and Code Generation of Autonomic Systems*, LAP Lambert Academic Publishing

[6] Vassev E and Hinchey M 2009 *ASSL: A Software Engineering Approach to Autonomic Computing*, *IEEE Computer* **42** (6) 106

[7] Vassev E, Hinchey M and Paquet J 2008 *A Self-Scheduling Model for NASA Swarm-Based Exploration Missions using ASSL*, *Proc. of the Fifth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASe'08) IEEE Computer Society Press* 54

[8] Vassev E, Hinchey M and Paquet J 2008 *Towards an ASSL Specification Model for NASA Swarm-Based Exploration Missions*, *Proc. of the 23rd Annual ACM Symposium on Applied Computing (SAC 2008) - AC Track*, ACM 1652

[9] Vassev E and Hinchey M 2009 *Modeling the Image-processing Behavior of the NASA Voyager Mission with ASSL*, *Proc. of the 3rd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'09) IEEE Computer Society* 246

[10] Vassev E and Mokhov S A 2009 *Self-Optimization Property in Autonomic Specification of Distributed MARF with ASSL*, *Proc. of the 4th International Conference on Software and Data Technologies (ICSOFT 2009)*, INSTICC, **1** 331

[11] Mokhov S A and Vassev E 2009 *Autonomic Specification of Self-Protection for Distributed MARF with ASSL*, *Proc. of C* Conference on Computer Science Software Engineering (C3S2E '09)*, ACM 175

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

[13] Hinchey M, Rash J, Truszkowski W, Rouff C and Sterritt R 2005 *Autonomous and Autonomic Swarms*, *Proc. of 8th Biennial Conference on Real Time in Sweden (RTiS)* 65

[14] Curtis S A et al. 2000 *ANTS (Autonomous Nano-Technology Swarm): An Artificial Intelligence Approach to Asteroid Belt Resource Exploration*, *Proc. of the 51st Congress of International Astronautical Federation*, International Astronautical Federation

[15] Hinchey M, Dai Y, Rash J, Truszkowski W and Madhusoodan M 2007 *Bionic Autonomic Nervous System and Self-healing for NASA ANTS-like Missions*, *Proc. of the 2007 ACM Symposium on Applied Computing (SAC 2007)* 90

[16] Rouff C A, Hinchey M G, Rash J L and Truszkowski W F 2005 *Towards a Hybrid Formal Method for Swarm-Based Exploration Missions*, *Proc. of the 29th Annual IEEE/NASA Software Engineering Workshop (SEW2005)* 253

[17] Bonabeau E and Threraulaz G 2000 *Swarm smarts*, *Scientific American* 72

[18] Hinchey M, Rash J and Rouff C 2005 *Requirements to Design to Code: Towards a Fully Formal Approach to Automatic Code Generation*, *Technical Report TM-2005-212774*, NASA Goddard Space Flight Center
[19] IBM Tivoli 2005 *Autonomic Computing Policy Language (Tutorial)*, IBM Corporation
[20] Agrawal D et al. 2005 *Autonomic Computing Expressing Language (Tutorial)*, IBM Corporation

Dr. Emil Vassev has a long-standing experience as both software engineer and research scientist in various fields of computer science and software engineering. He received his M.Sc. in Computer Science (2005) and Ph.D. in Computer Science (2008) from Concordia University, Montreal, Canada where he is an Affiliate Assistant Professor at the Department of Computer Science and Software Engineering.
For over 15 years, Dr Vassev has been actively pursuing research in autonomous systems, artificial intelligence, data science, software engineering, formal methods and compilers. His extensive publishing record (over 150 peer-reviewed scientific publications, including 3 books) and his methodical and meticulous work in various fields of computer science and software engineering have made him one of the notable scientists in these fields. Dr Vassev's collaboration with NASA inspired two NASA patents.

Mike Hinchey is Professor of Software Engineering and Head of the Department of Computer Science Information Systems at University of Limerick, Ireland, where he is also Emeritus Director of Lero, the Science Foundation Ireland Research Centre for Software. Prior to joining the university, Professor Hinchey was the Director of the NASA Software Engineering Laboratory. In 2009, he was awarded NASA's Kerley Award as Innovator of the Year and is one of only 36 people recognized in the NASA Inventors Hall of Fame. Professor Hinchey holds a B.Sc. in Computer Systems from University of Limerick, M.Sc. in Computation from University of Oxford and a PhD in Computer Science from University of Cambridge. He is Editor-in- Chief of Innovations in Systems and Software Engineering: a NASA Journal and Journal of the Brazilian Computer Society and Associate Editor of ACM Computing Surveys. Professor Mike Hinchey is President of IFIP, the International Federation for Information Processing (www.ifip.org) for 2016-2022. He is also President of the Irish Computer Society and Past Chair of the IEEE UK Ireland Section.