Towards Certifiable Autonomic Computing Systems Part II: Measuring and Rating Autonomic Computing Systems

Haffiz Shuaib and Richard John Anthony
Autonomics Research Group
School of Computing and Mathematical Sciences,The University of Greenwich.
Park Row, Greenwich, London SE10 9LS, UK
Email: haffiz.shuaib@yahoo.com, R.J.Anthony@gre.ac.uk

***Abstract*—Autonomic computing systems are a promising technology for bending the cost curve associated with information and communication technology (ICT) service management and for aiding the growth and evolution of complex computing systems. Indeed, this has motivated a significant amount of research. However, a central plank to achieving fully-fledged autonomic computing systems is missing i.e., the ability to certify these systems. The certification process will provide a basis; for assessing the quality of autonomic systems with similar functionalities, for assessing the current capability of the system and its suitability to the problem, to assess the impact of a certified component on a system and to resolve legal liability, if the autonomic computing systems were to fail.**

In this second part of a two-part paper, several steps to rate or certify autonomic computing systems within the context of the targeted application domain are proposed. In the first instance, the autonomic manager architecture proposed in the first part of this work is associated with indices that indicate how mature an autonomic machine is. The maturity index, the layer configuration of the machine and the implemented autonomic self-management properties are used to derive a mathematical expression that describes the machine in qualitative terms. These qualitative metrics in turn point to what quantitative measures or performance characteristics can be obtained from the machine under an evaluation scenario. The proposed quantitative metrics are based on the International Standard Organization's software quality specification i.e., ISO/IEC 9126. Using the software engineering standard for product evaluation i.e., ISO/IEC 14598-4, the four steps for certifying an autonomic computing system are outlined. Finally, an Ant Colony Optimization (ACO) application called Path Finder (PF) is used to demonstrate the proposals in this work.

Keywords-Autonomic computing systems; Certification; Performance; Verification; Measurement;

I. Introduction

A true Autonomic Computing System (ACS) is one that is able to automate the management decision-making process and reflect on the quality of the decisions made. This, it must do regardless of the environmental context and within the goals set by the human operator. The ultimate aim of autonomic computing systems is to allow complex Information Technology (IT) infrastructure to evolve to handle more difficult tasks or change in their immediate environment, without significantly increasing the cost of management.

As with most critical or increasingly complex systems, an ACS should and must be certified on the basis of its expected characteristics before it goes live, as these systems have applications from the financial to the space exploration industries. However, no progress has been made in the area of autonomic computing certification i.e., there is no framework to guide the process by which two or more autonomic machines are rated in relative terms, assuming these machines target the same application domain and no standard measure of performance for these systems [1]. A crucial aspect of correctly assessing the quality of an autonomic computing system is knowing what to measure and where to take these measurements. This task is often very difficult [2]. An attempt to address this difficulty is the primary objective of this paper. To this end, the following are proposed;

1) Qualitative measures that convey the complexity, intelligence and functionality of the autonomic machine.
2) Quantitative metrics that allow the autonomic machine to be measured based on specific performance attributes.
3) A fixed number of evaluation planning and execution steps that will lead to a final certification statement for an ACS.

These three objectives are dealt with within the context of the four cardinal self-management properties identified for ACSs i.e., self-configuration, self-optimization, self-protection and self-healing [3]. The idea is that since the four self-management properties to varying extents are applicable to all autonomic computing application domains, the certification framework can be made domain agnostic by defining it around these properties. This framework, once defined, is applied to an autonomic application use-case i.e., Path Finder (PF). The PF application is an Ant Colony Optimization (ACO) application in which an autonomic manager guides a managed object or robot along a gridded map. As with all ACO applications, the primary objective is to get the robot to the food source from the nest and back to the nest using the shortest route. This application is written in the C-Sharp programming language.

This paper collates together the findings of a detailed research project whose roadmap can be found in [1] and more extensive details in [4].

The rest of this paper is structured as follows; in the section following, a brief recap of the intelligent machine design (IMD) architecture covered in the first part of this paper is presented. In Section III, an expression that conveys a qualitative measure of an autonomic computing machine is derived. Quantitative metrics based on the normative framework of the International Standard Organization/International Electrotechnical Commission software quality specification [5] are discussed in Section IV. How these metrics apply to autonomic computing systems is presented in Section V. Section VI contains the steps for software evaluation based on ISO/IEC 14598 [6] that should lead to a final certification statement for an evaluated ACS. The proposals of Sections III - VI are demonstrated using the PF application in Sections VIII - XI. See Section XII for the conclusion. The appendix discusses special cases of the mathematical expression proposed in Section III.

II. The Intelligent Machine Design Architecture: A Recap

For completeness, a technical summary of the intelligent machine design (IMD) architecture is presented in this section. See Section II-B of the first part of this two-part paper [7] for a more exhaustive discussion on, and the philosophy behind this architecture.

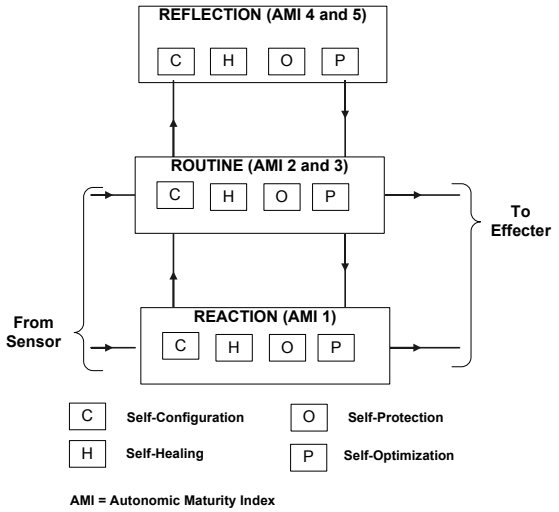


Fig. 1. An Autonomic Computing expression of the IMD (The AMI is discussed at length in Section III)

The IMD is made up of three distinct layers viz; The Reaction (R1), Routine (R2) and Reflection (R3) layers as shown in Figure 1. The Reaction layer is the least intelligent of all three, in that it accepts information from the sensory input and effects a change through its singular implemented policy rule. If this policy rule is unable to handle the input information to the machine, R1 passes control over to the Routine layer.

The Routine layer on the other hand, implements more than one policy rule and can select and apply the best rule from amongst these for the input context. As a result, the speed at which the Routine layer reacts to changes is expected to be relatively slower when compared to the Reaction layer. If the Routine layer is unable to find a suitable policy rule to effect a corresponding change or if two or more policy rules apply to an input context, it defers to the Reflection layer.

The algorithms implemented in the Reflection layer attempt either one of two things; (1) use of complex techniques e.g., artificial intelligence mechanisms (Fuzzy Logic etc.), to resolve policy rule conflicts flagged by the R2 layer or (2) create a new policy rule on the fly when none of the policy rules in the repository apply to the extant input context. The Reflection layer is the only one of the three that does not have sensory (S) inputs or effecter (E) outputs. The Reflection layer can inhibit or excite the activities of the Routine layer if required. The Routine layer can also do the same to the Reaction layer.

The three-layer structure of the IMD can thus be aligned with a simplified view of human problem solving in which there is a need to achieve rapid pre-programmed response in some situations, whilst being able to take a longer time to determine the best of several possible options in other situations, and ultimately being able to reason and learn new strategies in new situations. Using car driving as a means to explain this, consider emergency braking where the pre-programmed Reaction is faster than thinking speed; choosing which lane to enter at a fuel station, based on observing queue length (Routine); and studying the map to try to avoid an area of heavy congestion (Reflection).

III. Qualitative Metrics for Autonomic Computing Systems

It can be envisaged that autonomic computing products will offer a range of competing management services in the near future. If multiple systems are targeted at a specific application domain, then a means by which these systems are qualified from an autonomic perspective is required. Knowing the relative autonomic level at which a system operates will reveal what quantitative measurements can be extracted e.g., efficiency, latency etc. and what results to expect within the bounds of the complexity of a goal. This section contains the relevant discussions and proposals in this regard.

In the first subsection, the related works carried out in this area are discussed, together with apparent disadvantages. A five-level autonomic index that can be aligned with the IMD is proposed in Subsection III-B. These proposed indices are linked with the four cardinal self-management properties of autonomic computing systems in Subsection III-C. This link is to ensure that the proposed indices have relevance to most autonomic application domains.

A. Measuring Autonomicity

The term by which systems are ranked based on their particular autonomic capabilities go by a number of names in autonomic literature. For example, it is referred to as Autonomic Control Levels (ACL) in [2], Levels of Autonomy (LOA) in [8] and Degree of Autonomicity [9]. Still other papers term it the Autonomic Adoption Model (AAM) [3] or the Autonomic Computing Maturity Index (AMI) [10]. For consistency in this work, the preferred term will be AMI.

Several attempts have been made to describe criteria for which the AMI is to be based. For instance, [11] proposes using the following as the basis for assessing autonomic capabilities; the complexity of the objective, the operating environment and the level of human interaction with the machine. The motivation for the scale above was to have a consistent measure by which costs and suitability of a proposed robotic system for military operations can be ascertained.

In [12], a 10-point AMI that ranges from High (10) to Low (0) that depends on the relative influence of the participating entities i.e., Man or Machine on the following attributes was proposed;

- Information Acquisition: Reading, sorting, filtering and aggregating input data.
- Information Analysis: Performing complex computation on the acquired data e.g., prediction, data integration etc.
- Source of Decision: Making decisions based on the analyzed data. And
- Source of Action: Take an action based on the decision made.

The more the machine handles any of the above listed attributes, the higher up the scale the system is assessed to be.

An 11-point autonomic scale is presented in [2] based on similar attributes to [12]. The only difference is that they are labelled differently. The attributes of [12] i.e., Information Acquisition, Information Analysis, Source of Decision and Source of Action are called Observe, Orient, Decide and Act, respectively in [2].

The AMI proposed in [3] is characterized by what parts of the system's autonomic management activities are automated versus those that are manually implemented; The resulting five level autonomic scale is as delineated below;

- Manual Level: At this level all autonomic management activities are handled by the human operator.
- Instrument and monitor: Here, the autonomic system is responsible for the collection of information: This collected/aggregated information is analyzed by the human operator and guides future actions of the operator.
- Analysis Level: On this level, information is collected and analyzed by the system. This analyzed data is passed to the human administrator for further action(s).

- Closed loop Level: This works in the same way as the Analysis level, only this time the system's dependence on the human is minimized i.e., the system is allowed to action certain policies.
- Closed loop with business processes Level: At this level, the input of the administrator is restricted to creating and altering business policies and objectives. The system will operate independently using these objectives and policies as guides.

The AMI system proposed in [11] is targeted specifically at robots, thus limiting its application domain. At the time of its publication i.e., [11], its definition of the autonomic scale was a work in progress. The AMI in [12] is based on who makes the decisions and how these decisions are executed. Clough's AMI definition is also based on these two criteria [2]. From a certification perspective, the place of a system on the autonomic computing maturity index should be defined by who/what makes the decisions and the quality of the decisions themselves. After all, this is how human managers that autonomic systems are supposed to steadily replace would be evaluated. Both metrics will engender a certain level of trust in the system. The scales in [3] is said to be narrowly defined and technically vague [9]. This makes it difficult to align an autonomic system with these maturity indices [13]. These concerns do not help the certification process along. In the next subsection, an AMI that includes some of the advantages of the autonomic ranking system discussed in this section is proposed.

B. Autonomic Computing Maturity Indices (AMI)

The architecture shown in Figure 1 can be associated with the AMI. To do this, an attempt is made to expressly define what each Maturity Index means from a technical perspective, and further relate each index to the layers of the IMD. The Five maturity indices are thus interpreted as:

- **Maturity Index 1:** Here, only one policy action is executed in response to all input signals and encountered contexts. Complex operations are referred to the human operator or to the immediate higher layer. This maturity index corresponds to the Reaction layer.
- **Maturity Index 2:** This index corresponds to the Routine layer. If the Routine layer is unable to find a suitable policy rule from a policy repository or if there is a policy rule ambiguity, it relies on the human administrator to provide a new solution or resolve the policy rule conflict.
- **Maturity Index 3:** This is similar to Maturity Index 2, only that this time, the Routine layer consults the Reflection layer to solve its policy rule problems.
- **Maturity Index 4:** This index corresponds to the Reflection layer. The Reflection layer of a Machine in this index will attempt to solve the policy rule problem of the Routine layer, and monitor the implementation of this new policy rule. If the policy rule fails in its objective or if a new policy rule cannot be created, the human administrator is required to intervene.
- **Maturity Index 5:** This is similar to index 4, but rather than defer to the human administrator, if a suitable policy rule is not found or created, the algorithm within the Reflection layer will continually attempt to create a new policy rule or resolve the policy rule conflict. This index should be used to define autonomic machines that will be unable to get in touch with the human manager, a craft in deep space for example. Another possible example for this index is a scenario where the human intervention cannot be timely enough due to the complexities in the system.

In effect, the autonomic maturity level 1 corresponds to the Reaction layer, levels 2 and 3 correspond to the Routine layer, levels 4 and 5 correspond to the Reflection layer. The position of an autonomic computing system on the defined maturity indices above provides a possible basis for verifying the source of the decision making process and the quality of the decisions made. For instance, if a system in question specifies a Maturity Index of 2, the certification process would know that the 'court of last instance' is the human administrator. The certification process would now seek to verify the qualification of skilled personnel for the system to be awarded an index of 2. If the system seeks to be tagged with an index of 5 i.e., the decision making process is handled ultimately by the machine itself, the algorithm implemented in the Reflection layer must be shown to be robust enough to handle this task.

C. Autonomic Self-Management AMI Qualification

Regardless of the application domain targeted by an autonomic computing system, it is expected that some or all four of the self-management properties be implemented. The AMI proposed in the last subsection is able to achieve application domain agnosticism by being associated with these self-management properties, as opposed to tying it to a specific application.

Consequently, a mathematical expression that describes an autonomic manager that implements the IMD architecture is derived. The benefits of this expression include:

1) The verification of the characteristics of the autonomic manager at a glance. Examples of such characteristics are the complexity, implemented functionality of the machine, level of intelligence etc.
2) An indication of the extent to which a manager conforms to the IMD specification.
3) An indication of the self-management properties implemented in the manager.
4) Assisting the design of automated architectural verification algorithms of the implemented machine, if required.
5) It indicates the relevant quantitative metrics that ought to be measured. The certifier can use this as a guide during the evaluation process. Quantitative metrics are discussed in Section V.

Before deriving the expression describing the machine, it is instructive to restate here that only five layer configurations are allowed for the IMD. Section VI of Part I has shown why Expressions (1) - (5) are the only legal IMD configurations, and as such the explanation here is kept brief. All IMD configurations must contain either R1 or R2, as these layers are the only ones connected to the sensory and effecter mechanisms. If the R3 layer is implemented then the R2 layer must also be present, as R3 cannot communicate with R1 directly.

$$R1 \leftrightarrows \emptyset \leftrightarrows \emptyset \tag{1}$$

$$R1 \leftrightarrows R2 \leftrightarrows \emptyset \tag{2}$$

$$R1 \leftrightarrows R2 \leftrightarrows R3 \tag{3}$$

$$\emptyset \leftrightarrows R2 \leftrightarrows \emptyset \tag{4}$$

$$\emptyset \leftrightarrows R2 \leftrightarrows R3 \quad (5)$$

Where R1, R2 and R3 represent the Reaction, Routine and Reflection layers, respectively and $\leftrightarrows$ represents a connection between the layers of the machine.

Observe from Figure 1 that all four self-management properties can be implemented in all layers of the IMD, although with varying degrees of intelligence, speed and complexity. Expression (6) is used to symbolically represent the self-management properties implemented by a specific layer of the IMD.

$$SM_{RX} = \{SC_X, SH_X, SO_X, SP_X\} \quad (6)$$

where $X = \{1, 2, 3\}$ is the level of the IMD where the relevant self-management property is implemented. Thus SC_X, SH_X, SO_X, SP_X represent the implementation of the self-configuration, self-healing, self-optimization and self-protecting properties at layer X, respectively.

If all four self-management properties are implemented in the Routine layer (R2) then symbolically it is represented by $SM_{R2} = \{SC_2, SH_2, SO_2, SP_2\}$. Similarly, if the Reaction layer (R1) does not implement the self-configuration property then the appropriate representation is $SM_{R1} = \{\emptyset, SH_1, SO_1, SP_1, \}$. In other words, if a self-management property is not implemented, its corresponding symbol is replaced with $\emptyset$ in the enclosed set of Expression (6).

Based on the discussion so far, an IMD compatible Autonomic Manager can be described by a combination of its AMI, IMD layer configuration and the implemented self-management properties as shown in Expression (7).

$$AM_z = \left\{ \begin{array}{l} AMI; R1 \leftrightarrows R2 \leftrightarrows R3; \\ SM_{R1} = \{SC_1, SH_1, S_{O1}, SP_1\} \\ SM_{R2} = \{SC_2, SH_2, SO_2, SP_2\} \\ SM_{R3} = \{SC_3, SH_3, SO_3, SP_3\} \end{array} \right\} \quad (7)$$

where $AMI = \{1...5\}$.

Consider the AMs described by Expressions (8), (9), (10) and (11).

$$AM_1 = \left\{ \begin{array}{l} 4; R1 \leftrightarrows R2 \leftrightarrows \emptyset; \\ SM_{R1} = \{SC_1, SH_1, SO_1, SP_1\} \\ SM_{R2} = \{SC_2, SH_2, SO_2, SP_2\} \\ SM_{R3} = \{SC_3, SH_3, SO_3, SP_3\} \end{array} \right\} \quad (8)$$

$$AM_2 = \left\{ \begin{array}{l} 4; \emptyset \leftrightarrows \emptyset \leftrightarrows R3; \\ SM_{R1} = \{SC_1, SH_1, SO_1, SP_1\} \\ SM_{R2} = \{SC_2, SH_2, SO_2, SP_2\} \\ SM_{R3} = \{SC_3, SH_3, SO_3, SP_3\} \end{array} \right\} \quad (9)$$

$$AM_3 = \left\{ \begin{array}{l} 4; R1 \leftrightarrows R2 \leftrightarrows R3; \\ SM_{R1} = \{SC_1, SH_1, SO_1, SP_1\} \\ SM_{R2} = \{SC_2, SH_2, SO_2, SP_2\} \\ SM_{R3} = \{SC_3, SH_3, SO_3, SP_3\} \end{array} \right\} \quad (10)$$

$$AM_4 = \left\{ \begin{array}{l} 2; R1 \leftrightarrows R2 \leftrightarrows \emptyset; \\ SM_{R1} = \{SC_1, SH_1, SO_1, SP_1\} \\ SM_{R2} = \{SC_2, SH_2, SO_2, SP_2\} \\ SM_{R3} = \{\emptyset, \emptyset, \emptyset, \emptyset\} \end{array} \right\} \quad (11)$$

AM_1 in Expression (8) is invalid because it specifies an AMI value of 4 but does not implement the Reflection layer (R3). Recall from the last subsection that AMI 4 and 5 reside at the Reflection layer. Since R3 is not implemented in AM_1, it should specify $SM_{R3} = \{\emptyset, \emptyset, \emptyset, \emptyset\}$ not $SM_{R3} = \{SC_3, SH_3, SO_3, SP_3\}$. AM_2 (see Expression 9) is invalid since the layer configuration $\emptyset \leftrightarrows \emptyset \leftrightarrows R3$ is not among those allowed for the IMD (see Expressions 1-5 at the beginning of this section). AM_3 and AM_4 in Expressions (10) and (11) conform to the IMD rules and thus valid.

If all four AMs were to go through a certification process, then AM_1 and AM_2 would have failed the first test and resources need not be expended to verify them or measure other attributes further. If AM_3 and AM_4 were designed in a way that both targeted the same application domain, from Expressions (10) and (11), one can tell that AM_3 will be expected to adapt to contexts that deviate from the norm, as it implements a Reflection layer (R3). Tests to verify this superior management capability should be carried out on AM_3 but not on AM_4, as it does not have an R3. The AMI of AM_3 also points to the fact that it is more mature in relative autonomic terms. These are just a few examples of how an expression describing an AM can point to what can and cannot be measured quantitatively (see the next section for discussions on quantitative metrics).

There are certain AM implementations that can appear to reveal some inconsistencies when attempts are made to describe them using Expression (7). These apparent inconsistencies are treated in the appendix.

IV. Quantitative Metrics for Software Evaluation

The Qualitative metrics presented in the last section while relevant to the certification process, lack a means by which an autonomic system is measured on a scale of magnitudes. A number of quantitative metrics must be derived to address the above. The International Standard Organization software quality specification i.e., ISO/IEC 9126-1998 is the basis on which the autonomic quantitative measures presented in the next section are defined.

ISO/IEC 9126 [5] defines six main characteristics that can be used to assess the quality of a software product, including; Functionality, Usability, Portability, Reliability, Efficiency and Maintainability. These normative characteristics and their attributes can be used to pose certain questions to the evaluation of an autonomic computing machine. The answers to these questions, which may be boolean or numerical ratios can be used to derive a single value or set of values that form the basis for which a system to be certified is rated.

In terms of the ***Functionality*** characteristic the following questions are posed:

1) *Suitability*: Does a function exist within the implemented system that provides for a specifically stated or implied need?
2) *Accuracy*: If it does, how well does it meet that need?
3) *Interoperability*: Is it able to interact with other systems e.g., AMs deployed in the same environment?
4) *Security*: How well does it prevent unauthorized access to the system data?

Questions relating to the ***Reliability*** characteristic include:

1) *Maturity*: What is the Mean Time to Failure of the system?
2) *Fault Tolerance*: Can the system maintain a specific level of performance in the face of a fault? i.e., how robust is the system?
3) *Recoverability*: Can the system regain peak performance after the impact of a failed component is mitigated?

The ***Usability*** characteristic deals with how user-friendly the system is. This characteristic is not discussed further in this work because it is subjective and directly dependent on how the system is developed and the system's targeted application domain.

The attributes of the ***Efficiency*** characteristic ask the following questions:

1) *Time (temporal) behaviour* : How much time does it take to complete a task or does the system meet the hard or soft execution time constraints?
2) *Resource Utilization*: How much resources in terms of memory space, CPU cycles and network bandwidth are committed to achieving the task?

The ***Maintainability*** characteristic deals with the following questions:

1) *Analysability*: How well are system faults and their causes recognized and understood?
2) *Changeability*: Can the system or part of it be modified easily?
3) *Stability* : When the system is modified, how well does it perform thereafter?
4) *Testability* :Regardless of whether changes are made can the system be validated?
5) *Modularity and Coupling*: Can the system be expressed in specific component parts and can these parts be joined together efficiently?

Finally, the questions associated with the Portability characteristic deals with:

1) *Adaptability*: Can the software be adapted for another environment using only components contained within?
2) *Installability*: How easy is it to install?
3) *Co-Existence*: When installed in a new environment can it co-exist with other installed components?
4) *Replaceability*:Can it efficiently replace another software designed for the same purpose?

A *Compliance* attribute applies to all six characteristics discussed above. This attribute seeks the answer to the following question; How well does the autonomic machine conform to specified standards/conventions etc.?

These six software quality characteristics are applicable to two types of metric namely ***Internal*** and ***External***. The Internal metrics are applicable to the quality of the actual code, while the External metrics apply to the operational behaviour of the software code. While both metrics are equally important, evaluation of the Internal metric should be left to the autonomic computing system developer. The ACS certifier should only be concerned with the External metric for the following reasons:

1) The developer of the code might not want to reveal the internal logic, thereby protecting intellectual property rights and trade secrets. This must not pose a barrier to certification.
2) The operational external behaviour of the code under a rigorous test is sufficient to inform on whether the autonomic machine does what it says it does.
3) Points 1 and 2 allow the containers of the code e.g., the three layers of the IMD to act as black boxes that can be tested, thereby lessening the amount of work on the certifying authority without compromising the quality of the certification process.

V. Quantitative Metrics for Autonomic Computing Systems

The broad ISO/IEC software evaluation characteristics discussed in the last section are applied to autonomic computing systems in this section. Specifically, the methods for computing these quantitative metrics, the outputs of these computations and the interpretation of these outputs are proposed and presented.

A. Functionality

There are several dimensions to measuring the functionality characteristic in this work. In the first, the autonomic machine or system is measured in terms of the self-management properties it implements. The second dimension of this metric involves the level at which these management properties reside e.g., R1, R2 or R3. Note that the functionality behaviour expected at each of the three levels of the IMD will differ. For instance, at the R3 level, the assessor will want to verify if the Reflection layer is able to create a new policy or resolve a policy conflict on behalf of the Routine layer (R2). At the R2 level, the assessor will be looking at how well the Routine layer engages the Reflection layer when a context deviates from the norm, how well does the implemented policy selection algorithm execute given a specific context and how well it regulates the behaviour of the Reaction layer (R1). R1 will be evaluated based on how well it executes its implemented policy and how well it is able to engage R2 when the extant context cannot be dealt with.

From the above the following tasks and measurement types are derived based on the attributes of the functionality metric:

1) Suitability:

- **R1 Functional Suitability**
  Task: Find out if the self-management property i.e., self-configuration, self-healing, self-optimization or self-protection is supposed to be and is implemented in R1. Note that this can be deduced from the expression describing the machine, specifically SM_{R1} in Expression (7).

  Output Metric: 1 for Yes and 0 for No for each of the applicable self-management properties.

- **R2 Functional Suitability**
  Task: Find out if the self-management property i.e., self-configuration, self-healing, self-optimization or self-protection is supposed to be and is implemented in R2. Note that this can be deduced from the expression describing the machine, specifically SM_{R2} in Expression (7).

  Output Metric: 1 for Yes and 0 for No for each of the applicable self-management properties.

- **R3 Functional Suitability**
  Task: Find out if the self-management property i.e., self-configuration, self-healing, self-optimization or self-protection is supposed to be and is implemented in R3. Note that this can be deduced from the expression describing the machine, specifically SM_{R3} in Expression (7).

  Output Metric: 1 for Yes and 0 for No for each of the applicable self-management properties.

2) Accuracy: A functional self-management property that evaluates to 'Yes' after the suitability check is tested a number of times to verify its functional accuracy when it executes. In order to compute the measure of accuracy (A) the total number of times the self-management property is tested is counted and assigned to a variable N_{total}. The number of tries in N_{total} for which it executed as expected are counted and assigned to the variable $N_{success}$. Correspondingly, N_{fail} holds the total number of times the self-property fails the task. The accuracy of the task is given as the ratio of $N_{success}$ and N_{total} i.e., $A = \frac{N_{success}}{N_{total}}$. Functional elements that evaluate to 'No' at the functional suitability stage are awarded

a value of 0. This implies that the range of values for accuracy is $0.0 \leq A \leq 1.0$

- **R1 Functional Accuracy**
  Task 1: What is the accuracy with which the singular policy of R1 executes?

  Output Metric 1: $A_1 = \frac{N_{success}}{N_{total}}$

  Metric 1 Interpretation: The closer the value of A_1 is to 1.0 the better.

  Task 2: How accurate is the logic at R1 that passes control from R1 to R2, if the context cannot be properly handled by R1?

  Output Metric 2: $A_2 = \frac{N_{success}}{N_{total}}$

  Metric 2 Interpretation: The closer the value of A_2 is to 1.0 the better.

- **R2 Functional Accuracy**
  Task 1: Count the number of times R2 succeeds ($N_{success}$) in selecting the best policy for the specific context after N_{total} number of tries.

  Output Metric 1: $A_1 = \frac{N_{success}}{N_{total}}$

  Metric 1 Interpretation: The closer the value of A_1 is to 1.0 the better.

  Task 2: Count the number of times R2 correctly engages R3 or the human operator to solve policy conflicts or request the creation of a new policy.

  Output Metric 2: $A_2 = \frac{N_{success}}{N_{total}}$

  Metric 2 Interpretation: The closer the value of A_2 is to 1.0 the better.

  Task 3: Count the number of times R2 correctly moderates the actions of R1 by overwriting the policy executed by R1.

  Output Metric 3: $A_3 = \frac{N_{success}}{N_{total}}$

  Metric 3 Interpretation: The closer the value of A_3 is to 1.0 the better.

- **R3 Functional Accuracy**
  Task 1: Verify how accurate R3 is at creating a new policy or resolving policy conflicts reported by R2.

  Output Metric 1: $A_1 = \frac{N_{success}}{N_{total}}$

  Metric 1 Interpretation: The closer the value of A_1 is to 1.0 the better.

  Task 2: How accurate is R3 when signalling to the human operator that it is unable to solve a problem reported by R2. This task assumes that the machine operates at AMI level 4 (see Section III-B).

  Output Metric 2: $A_2 = \frac{N_{success}}{N_{total}}$

  Metric 2 Interpretation: The closer the value of A_2 is to 1.0 the better.

3) Interoperability: In Section X of the first part of this paper, a number of interoperability mechanisms where proposed to aid management coordination. These mechanisms included proposals for; (1) establishing administrative relationships, (2) resolving management conflict, (3) monitoring autonomic elements, (4) support for granting and requesting services, (5) policy sharing and (6) reliable remote policy communication.

Task: Let $O_{success}$ be the number of interoperability mechanisms implemented and O_{total} be the total number of interoperability mechanisms required for the application under consideration.

Output Metric: $I = \frac{O_{success}}{O_{total}}$

Note that $O_{total} \leq 6$.

Metric Interpretation: The closer the value of O is to 1.0 the better.

4) Security: In Section X of Part I of this work, a mechanism for establishing administrative and security relationships between elements of an ACS was presented. These relationships are established through the *I-4* interfaces of the IMD. The IMD has four different connections that use the *I-4* interface (see Figure 3 in Part I). This Security attribute verifies that all the active *I-4* interfaces are secure, as per the standard security procedure laid out for establishing security relationships.

Task: Count the total number of active *I-4* interfaces that conform to the security conventions and standards and store this value in $I_{success}$. Let I_{total} be the total number of active I-4 interfaces.

Output Metric: $S = \frac{I_{success}}{I_{total}}$

where $I_{total} \leq 4$.

Metric Interpretation: The closer the value of S is to 1.0 the better.

B. Reliability

Notice from Section IV, that the maturity, fault tolerance and recoverability attributes of the Reliability characteristic correspond to the self-management properties of self-healing and self-protection. Since the presence and accuracy of these self-management properties are already dealt with in Section V-A, there is no need to further define metrics for them here.

C. Usability

The usability metric is not discussed within the context of this work for the reasons given in Section IV.

D. Efficiency

The Efficiency attributes of Time behaviour and Resource utilization are measured during the period in which the functional accuracy tests are carried out. Note that the only relevant values for efficiency are those obtained when the measured accuracy tallies with the expected outcome i.e., $N_{success}$.

1) Time Behavior: ***Task***: This is a measure of how much time it takes to complete a self-management task i.e., T_{SM}. The time behaviour will naturally be impacted by memory access times, computation duration and network delays.

Output Metric:

$$T_{avg} = \frac{\Sigma^{N_{success}} T_{SM}}{N_{success}}$$

where T_{avg} is the average latency of an executed self-management property and $N_{success}$ the number of times for which the executed self-management property behaves as expected. The T_{avg}value can be linked to real-time managerial constraints in autonomic computing applications.

Metric Interpretation: The lower the value of T_{avg} the better.

2) Resource Utilization: ***Task***: Measure the amount of resources, specifically network bandwidth consumed (R_{ntw}), the amount of memory (R_{mem}) and the CPU utilization (R_{cpu}) used to achieve the self-management task. This metric enables the ACS or AM to be tested for compliance with the resources available on a given platform.

Output Metric: R_{ntw}, R_{mem} and R_{cpu} in bits/second, bytes and %, respectively.

Metric Interpretation: Generally, the lower the values of R_{ntw}, R_{mem} and R_{cpu}, the better.

E. Maintainability

The Maintainability characteristic is associated primarily with the architecture implemented, in this case the IMD reference architecture. As such the attributes for the Maintainability characteristic is discussed within the context of the IMD.

1) Analyzability: Since this attribute has to do with reporting failure or operational anomalies, the reporting agent must be able to precisely describe what the problem is and alert the agent in charge of addressing anomalies. Recall from Section IX of Part I that an autonomic element is able to write its undertakings to an operational log (see Figure 26 in Part I). Tabs can be placed on an autonomic element and corresponding actions taken by monitoring its operational logs. This, of course, assumes that the right administrative relationships have been established.

Task:Verify that faults injected into the system are properly logged and responded to for an autonomic element. Let MA_{total} be the total number of injected faults and $MA_{success}$ the total number of times the faults are resolved.

The analyzability metric is denoted as MA.

Output Metric:

$$MA = \frac{MA_{success}}{MA_{total}}$$

where $0 \leq MA \leq 1$

Metric Interpretation: The closer the value of MA is to 1.0 the better.

2) Coupling and Modularity: Within the context of this work, coupling and modularity (C & M) has to do with how the layers are arranged or configured based on the need of the targeted application domain. The Expressions for the five valid machine configurations are given in Section III-C. If these configurations are violated, then an autonomic manager fails the modularity and coupling test.

Task: Verify that the implemented configuration is valid.

Output Metric: 1 for Yes and 0 for No.

3) Stability: Assuming a layer of a machine implementing the IMD as its reference architecture was to go out of commission, without violating the permitted layer configurations, would other layers of the machine operate as usual.If the answer to this question is yes, the machine is deemed to be stable.

Task: Decommission 1 or 2 layers of the machine without violating the valid layer configurations, to see if the remaining layer or layers operate as expected.

Output Metric: 1 for Yes and 0 for No.

4) Testability: If the relevant functional characteristics of the ISO 9126-1998 specification can be applied to the machine under test, then it is testable. The output metric for this attribute is 1 for Yes and 0 for No.

F. Portability

1) Adaptability: This attribute relates to the ability of the autonomic machine to be adapted from one operating environment to another without modification. For example, if the autonomic system has been written using a programming language that runs on a virtual machine available to a number of considered hardware/software platforms, then the autonomic system is said to be adaptable. Adaptability is also aided in the proposed scheme by the fact that the IMD uses already standardized protocols, including the Policy Core Information Model (PCIM) [14][15] and the Lightweight Directory Access Protocol (LDAP) [16].

Task: Count the number of hardware/software platforms being considered. Store this value in P_{total}. Identify the number of platforms on which the autonomic system will run without modification. Store this value in P_{run}.

Output Metric: $P = \frac{P_{run}}{P_{total}}$

Metric Interpretation: The closer the value of P is to 1.0 the better.

2) Co-Existence: This is a measure of the impact a deployed ACS has on other systems running within the same resource domain. Note that the generation of this metric, and its interpretation, are highly system dependent.

Task: Count the number of applications that are impacted negatively by a deployed ACS and set this value as C_{total}.

Output Metric: C_{total}

Metric Interpretation: The lower the value of C_{total}, the better.

3) Installability and Replaceability: These attributes are not treated in this work.

VI. Certifying Autonomic Computing Systems

The International Organization for Standardization/ International Electrotechnical Commission defines software evaluation as a four-step process in ISO/IEC 14598 [6]. These four steps include; Establishing the evaluation requirements, Specifying the evaluation, Designing the evaluation and Executing the evaluation. These four steps are discussed here and subsequently applied to the evaluation of an application use-case later in this paper.

A. Establish the evaluation requirements

There are three tasks associated with this step. The first of these tasks has to do with establishing the purpose of the evaluation. This will involve a description of the autonomic computing application, specifically its goals. The second task is to identify the type of software product to be evaluated. In other words, is this a complete product or a component of a larger software application or a product still undergoing development? Specifying the quality characteristics that are to be measured is the last task for this step. Note that these characteristics are the six defined in ISO/IEC 9126 discussed in Section IV.

B. Specify the evaluation

This step also consists of three tasks including: (1) Selecting the quality metric type i.e., External or Internal (see last paragraph of Section IV). This task also involves establishing a common procedure for assigning values for measured attributes. The same measurements under the same conditions should produce similar and consistent values. (2) Establishing a rating level for each of the selected metric. This task mandates the certifier to state what specific quantifiable values for the measured quality characteristics are acceptable and those that are not. For example, the functional accuracy for a self-management property acceptable for one application might be 0.9 and for another 0.5 depending on the targeted application (see Section V-A2 for the interpretation of the functional accuracy metric). (3) Assigning weights to certain measurements. For instance, if the targeted application domain is one where computing resources are in abundance but operations need to be completed within a tight interval, then it is sensible that attributes relating to resource utilization should be assigned a lower priority and those relating to time behaviour assigned a relatively higher weight.

C. Design the evaluation

The design of the evaluation procedure will depend on the autonomic application. For instance, the evaluation for an application targeted at space exploration will almost certainly be carried out within a simulated environment. Other applications may be amenable to real time testing in the field. Regardless, the evaluation plan and design must be such that the most significant environmental test factors are considered. The plans and designs for evaluating an autonomic application is done in this step.

D. Execute the evaluation

This activity involves measuring the relevant quality characteristics identified in Section VI-A based on the evaluation plan set out. The computed results are matched against the acceptability rating criteria established in Section VI-B. With the ratings, a final verdict is pronounced on the system. For example, three possible certification statements are:

The system meets all the specified and implied needs of the end user.
or
The system meets all the specified and implied needs of the end user but requires more hardware to meet the acceptable efficiency characteristic rating.
or
The system does not adhere to relevant specifications and conventions and therefore does not meet the specific and implied need of the end user.

VII. The Application Use-Case

In Part I of this paper, an application called Path Finder (PF) was used to demonstrate some of the technical proposals made. For consistency, PF is used to demonstrate how the IMD and the certification process proposed previously can be applied to an autonomic application.

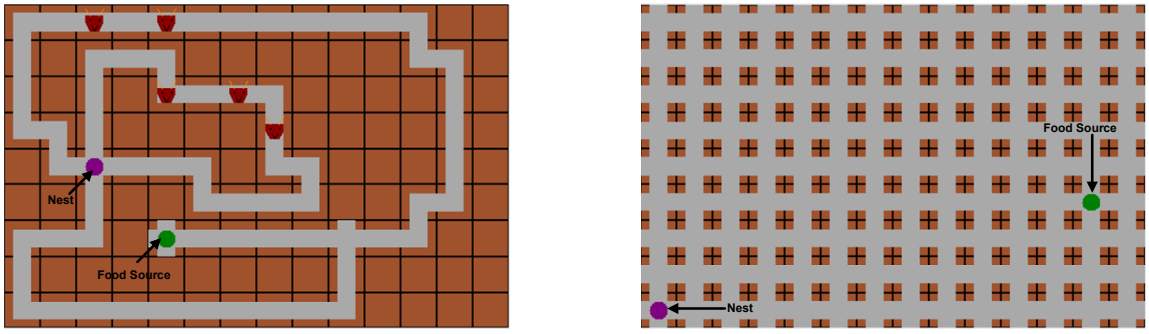
The PF which is an Ant Colony Optimization (ACO) application was selected as a use-case not only because it was sufficiently complex to demonstrate the proposed technical mechanisms but also because ACOs have a wide variety of applications. For instance, it is relevant to several domains, including but not limited to robotics, engineering, computer networks, finance, resource and job scheduling etc. Specifically, in this section, a relatively detailed description of the application is given. Its architectural configuration with respect to the IMD and its implemented policy framework is defined.

A. Application Description

The PF application in which robots are guided to and fro between a base (nest) and a target (food source) is used as a means of demonstrating the process of evaluation and comparison of AMs, using the techniques described in this work. A number of AMs are devised to navigate a robot in a maze from its base, to a target and back again in a simulated environment. The AMs have differing sophistication, using a variety of techniques to find the best route. The evaluation involves measuring a number of aspects of the AM's performance but the purpose of the exercise is not to find the absolute performance of these AMs, rather, the emphasis here is on showing how the evaluation of the AMs is performed and how they can be rated in terms of their suitability for purpose, accuracy and efficiency.

In PF, a robot begins from the nest and tries to find its way to the food source on a gridded map. When the food source is found, the robot must then navigate its way back to the nest. This process is repeated for the duration of the experiment. Regardless of how many times the robot has found the food source, when it gets to the nest it forgets the position of the food source and begins afresh to locate it. The reason for this is to mimic varying food source locations. Each robot has only local knowledge and only stigmergic communication occurs between robots i.e., robots are only able to influence one another indirectly through pheromones left on the paths. The lack of global knowledge requires that robots depend on intelligent search and navigational algorithms to find the food source or the nest. Each robot is controlled by a separate AM instance. In autonomic parlance, the robot is the Managed Resource while the AM is the Manager Element.

At each time step or clock tick, an AM must decide and then move a robot to the next square on the deployed map. The maps considered in this work are shown in Figures 2(a) and 2(b). The AM is allowed to move the robot in one of four available directions i.e., Top, Bottom, Left or Right square. The ability of a manager to steer its robot efficiently from the nest to the food source and back is linked to the level of Intelligence of the search algorithm within the manager. The implemented algorithms are discussed in detail in Section VIII-A. The position of the food source and the nest can be



(a) Closed Map (b) Open Map

Fig. 2. Closed and Open Maps

seen on both maps. It is worth mentioning that the closed map in Figure 2(a) is less complicated to navigate than that shown in Figure 2(b). The reason for this is that the open map has more pathways, thus creating relatively more opportunities for the robot to wander or get lost when searching for a target. Within the context of this work, a round trip from the nest to the food source and back to the nest is known as a ***Home Run***. It should be noted that the collective behaviour of the deployed robots and their managers as it relates to the objective is what is important, not the performance of the individual robots.

B. Application Use-Case Architectural Design

Recall from Figure 1, that all layers of the IMD are able to implement the four self-management properties. The evaluation focuses on the self-optimization management property i.e., it is the duty of the self-optimization mechanism of the implemented IMD to attempt to find the optimal path between the nest and the food source targets and move the robot accordingly. In Section VII-D3, it is shown that moves to the Top, Bottom, Right and Left Square on the gridded map are realized by four different policy rules. From the maps shown in Figure 2, on any clock tick, at least two moves (of the four valid moves) are possible. Recall from Section V of Part I, that two or more valid rules for a Context always generates a *ContextAmbiguity* event message which must then be handled by the R3 layer of the AM. Since every Context will generate a *ContextAmbiguity* event message, the R1 layer of the IMD is not required. As a result, Configuration V of the IMD, shown in Figure 3 and its message sequence shown in Figure 4, is the most appropriate for the PF application (see Section VI of Part I).

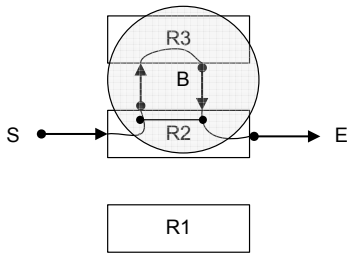


Fig. 3. Configuration IV ($\emptyset \leftrightarrows R2 \leftrightarrows \emptyset$)

With the description given above, the AM structure for the PF application can be described by Expression (12).

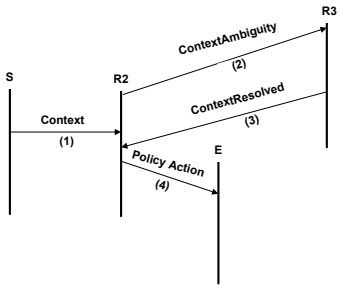


Fig. 4. Message Sequence for Configuration V

$$AM = \left\{ \begin{array}{l} 5; \emptyset \leftrightarrows R2 \leftrightarrows R3; \\ SM_{R1} = \{\emptyset, \emptyset, \emptyset, \emptyset\}, SM_{R2} = \{\emptyset, \emptyset, SO_2, \emptyset\} \\ SM_{R3} = \{\emptyset, \emptyset, SO_3, \emptyset\} \end{array} \right\} \quad (12)$$

The general form of the Expression can be found in Section III-C i.e., Expression 7. From the expression it can be seen that since R1 is omitted from the *PF* application, all self-management properties of R1 i.e., SM_{R1} are set to null. For R2 and R3 only the self-optimization property is implemented for the reasons given earlier. The self-configuration, self-healing and self-protection properties are set to null for R2 and R3, as can be seen in SM_{R2} and SM_{R3} in Expression (12). The numeral 5 in the Expression means that the machine operates at the Autonomic Maturity Index (AMI) of five (5) i.e., there is no human involvement in the operation of the autonomic managers for this application.

C. Application I-1 and I-2 Interface Information Structure

In Section IV of Part I, it was said that the information for the Sensory (S) interface, I-1 and that for the Effecter (E) interface, i.e., I-2 must conform to the object class structure stipulated in RFC 2252 [17]. The object classes: *pfSensory* and *pfEffecter* defined in Sections IV-A and IV-B of Part I of this paper are used for the I-1 and I-2 interfaces of the *PF* AM, respectively.

D. Application Use-Case Policy Framework Design

The Policy rules, conditions and actions to be used by any AM targeting this application domain are to follow the standardized policy core information model (PCIM) framework specified in RFC 3060 and 3460. The compliant rules, conditions and actions that should be implemented are defined in the subsections that follow.

1) Policy Condition Format: The defined PCIM policy condition class for this application i.e., ***pfCondition*** has six attributes, namely:*isActive*, *isValidMove*, *topDirection*, *bottomDirection*, *rightDirection* and *leftDirection*. All these attributes are of Boolean data types and are discussed in Section VII-B of Part I of this paper.

$$C = \{isActive, isValidMove, topDirection, bottomDirection, rightDirection, leftDirection\} \quad (13)$$

then the structures of the four realizable policy conditions are;

$$Policy\ Conditions = \begin{cases} C1 = \{True, True, True, False, False, False\} \\ C2 = \{True, True, False, True, False, False\} \\ C3 = \{True, True, False, True, False, False\} \\ C4 = \{True, True, False, False, False, True\} \end{cases}$$

An explanation of how a policy condition of type *pfCondition* object class is evaluated can be found in Section VII of Part I.

2) Policy Action Format: The policy action implemented in the *PF* application is of type *pfAction* and it consists of a single object attribute i.e., *functionID*. As noted in VII-C of Part I, it is of data type string and it points to the function that creates an instance of the *pfEffecter* object class.

If *A* in Equation (14) is the general structure of a policy action for this application;

$$A = \{functionID\} \quad (14)$$

then the four acceptable instances of the *pfAction* class are;

$$Policy\ Actions = \begin{cases} A1 = \{moveTop\} \\ A2 = \{moveBottom\} \\ A3 = \{moveLeft\} \\ A4 = \{moveRight\} \end{cases}$$

3) Policy Rule Format: Recall from Part I, that a policy rule consists of one or more policy conditions and actions and attributes that govern how these conditions are evaluated and how actions are to be executed. Conditions and actions are associated with a policy rule by adding the distinguished names (*DNs*) of the relevant policy conditions and actions to the rule's *ConditionList* and *ActionList* attribute, respectively. If the condition within a policy rule evaluates to true, all actions within that policy rule are executed subject to the other attributes of the rule. For the *Path Finder* application considered in this report, only four policy rules are required. These rules are based on the four conditions and actions described in the last two subsections.

Assume that Equation (15) represents the general structure of a rule;

$$R = \{C, A\} \quad (15)$$

where *C* is the *DN* of a policy condition and *A* the *DN* of a policy action.

The valid policy rules for the *PF* are;

$$Policy\ Rules = \begin{cases} R1 = \{C1, A1\} \\ R2 = \{C2, A2\} \\ R3 = \{C3, A3\} \\ R4 = \{C4, A4\} \end{cases}$$

VIII. Evaluation Requirements for Use-Case

This is the first of four defined steps set out by ISO /IEC 14598 that must be carried out when attempting to evaluate a software system (see Section VI-A). There are three tasks associated with this activity. These tasks are discussed within the context of the *PF* application in Sections VIII-A- VIII-C.

A. Purpose of Evaluation

The purpose of this evaluation is to establish how well in relative terms six different autonomic managers (AM) are able to achieve the objectives of the *pathfinder* (*PF*) application described in Section VII-A. All six AMs considered in this work implement a Fuzzy Logic algorithm in the R3 layer. This algorithm helps the AM decide the next move for its robot. Note that placing the Fuzzy Logic algorithm in this layer is in line with the idea that the R3 layer is the most intelligent of all three layers and houses the artificial intelligence mechanism, if implemented. The difference between the six evaluated AMs (described later) is in their implemented path search method and the quality of the inputs to the Fuzzy Logic algorithm.

Before discussing the AMs , it is pertinent to mention the common features of the managed robots:

- When a robot has found food, it deposits pheromones on the squares of the map it transverses on its way back to the nest. Depending on the navigational algorithm implemented in the AM, these pheromones may or may not be used by other robots as inputs to the Fuzzy Logic algorithm. Deposited pheromones evaporate after a controlled number of ticks of the clock.
- The robots do not have a global view of the considered map.
- Every time a robot sets out from the nest, it has no idea where the food source is located. The idea is to mimic multiple food sources.

The distinguishing characteristics of the evaluated AMs are presented below:

1) **Autonomic Manger-Basic (AM-B)**: In order to locate the food source or the nest, this AM uses an algorithm called **Basic**. In the **Basic algorithm**, the AM remembers the position of the last four squares crossed by its robot leading up to its current position. The current location of the robot, the available squares for the next move, the information relating to the last four squares crossed and the pheromones detected in each of these squares are passed to the *R3* layer by the *R2* layer of the manager using a *contextAmbiguity* message. The Fuzzy logic algorithm in *R3* uses this information as input and then outputs what it considers the best move for the robot. This output is sent to R2 using a *contextResolved* message. *R2* selects the policy rule that corresponds to that move and passes the associated policy action to the Effecter (E). The effecter moves the robot accordingly. The message sequence for this process is shown in Figure 4. Note that the Basic algorithm does not consider the relative strength of the pheromones on the squares when deciding the next move.
2) **Autonomic Manger-Pheromones (AM-P)**: This AM implements an algorithm similar to AM-B, only this time the square with the strongest pheromones are given higher weights when the decision is being made for the next move towards the

food source. The algorithm implemented by AM-P is called **Pheromones**.

3) **Autonomic Manger-Memory(AM-M)**: AM-Memory implements a variant of the Basic algorithm called the **Memory algorithm**. The difference here is that rather than remembering the last four squares traversed, the AM remembers all squares its robot passed through before getting to the food source. Once the food source is found, it traces its way back to the nest using the same path i.e., the one stored in its memory
4) **Autonomic Manger-Hill Climbing 1 (AM-HC1)**: This AM implements an algorithm called the **Hill Climbing 1 algorithm.** Like the Memory algorithm, it remembers the squares crossed to get to the food source, but unlike the Memory algorithm, it does not follow the route in its memory blindly back to the nest. It uses the stored information more selectively by applying the Hill Climbing search algorithm. The output of the Hill Climbing algorithm is used as one of the inputs to the Fuzzy Logic algorithm when the robot is trying to find its way home. This AM does not use pheromones.
5) **Autonomic Manger-Hill Climbing 2 (AM-HC2)**: AM-Hill Climbing 2 implements the **Hill Climbing 2 algorithm**. This algorithm is similar to that implemented in AM-HC1. The difference is that AM-HC2 additionally uses pheromones in the environment as inputs to its Fuzzy logic algorithm in exactly the same way as the Basic algorithm used in AM-B.
6) **Autonomic Manger-Hill Climbing 3 (AM-HC3)**: Implemented in this AM is an algorithm called **Hill Climbing 3**. This utilizes the Hill Climbing algorithm to find the shortest path back to the nest while giving higher weights to trails with the strongest pheromones when selecting the path to the food source in the same manner as AM-P.

B. Type of Product

The autonomic managers targeted at this application are self-contained and part of a larger system. They do not require additional components to carry out their activities.

C. Product Quality Model

The software quantitative metrics evaluated are:

- The Functionality attributes of *Suitability* to the objective and Accuracy with which the objective is achieved, if achieved at all (see Sections V-A1 and V-A2).
- Two attributes of the Maintainability characteristics are dealt with in this evaluation i.e., the Coupling and Modularity (C & M) attributes (see Section V-E2).

IX. Evaluation Specification for Use-Case

There are three tasks associated with this second evaluation activity as discussed in Section VI-B. These tasks are applied to the *PF* application and are presented in Sections IX-A- IX-C.

A. Metrics Selection

From the description of the PF application in Section VII, it is clear that the *R2* layer is mostly dependent on the *R3* layer for the policy rule choice for the next move of the robot. Put more succinctly; the machine is heavily dependent on the *R3* layer. As a result, only the Functional *Suitability* and *Accuracy* of R3 as it relates to the machine's self-optimization property is evaluated. Note that it is only when the Functional *Suitability* is confirmed to be a 1 is the *R3* Functional *accuracy* computed. All six AMs evaluated are functionally *suitable*.

The procedure for computing the Functional *Accuracy* metric of a self-management property at the *R3* layer is given in Section V-A2 as;

$$A_1 = \frac{N_{success}}{N_{total}} \tag{16}$$

where N_{total} is the total number of times the *R3* layer was called upon to resolve the a policy rule conflict/ambiguity by the *R2* layer and $N_{success}$ the total number of times these *R2* requests were successfully resolved.

Within the context of the *PF* application, $N_{success}$ is interpreted as the number of home runs (N_{hr}) achieved within the time considered. N_{total} represents the total number of steps i.e., the total number of squares (C_{sqr}) crossed by the robots, as instructed by the navigational algorithm in R3 to achieve N_{hr}. N_{hr} and C_{sqr} cannot be plugged into A1 in Equation (16) directly as these are two different quantities. A means to convert N_{hr} to steps or squares is required. To do this, the lowest (ideal) number of squares that can be traversed to achieve a single home run on a map is found and this number is the value of a single home run in squares (C_{hr}) for that map. For instance, for the Closed map in Figure 2(a), the lowest number of squares between the nest and the food source is 21 squares. Therefore, *42* (2×21) squares must be crossed for a round trip or 1 home run i.e., $nest \mapsto food \mapsto nest$ in the best case scenario. This implies that for the Closed map $C_{hr} = 42$ squares. For the Open map in Figure 2(b), $C_{hr} = 30$ squares. With the above, the R3 functional accuracy for the closed and open map can now be computed using Equation (17);

$$A_1 = \frac{N_{success}}{N_{total}} = \frac{C_{hr} * N_{hr}}{C_{sqr}} \tag{17}$$

The other metric evaluated in this work are the Coupling and Modularity attributes of the Maintainability characteristic (see Section V-E2). For the *PF* application, if the AM conforms to the configuration shown in Figure 3, the metric for these attributes is assigned a Yes or *1*, otherwise it is assigned a *0*. All six AMs evaluated conform to the configuration.

B. Rating Levels

A Functional *Suitability* metric value of 1(Yes) is a must for all AMs considered. For the purpose of this evaluation, if an AM is able to achieve a value of R3 Functional accuracy greater or equal to 0.4 for the self-optimization property, it is accepted. A Coupling and modularity value of 1(Yes) is mandatory for all evaluated AMs.

C. Criteria for Assessment

All three attributes discussed in Section IX-A to be measured i.e., *Suitability*; *Accuracy*; *Coupling and Modularity* are to be assigned equal weights in the final certification.

X. Evaluation Design for Use-Case

To assess the relative capabilities of the AMs with respect to the objective i.e., maximizing the number of home runs, each type of AM is tested in a simulated environment. The AMs guide their respective robots through the Closed and Open maps shown in Figures 2(a) and 2(b) for a set amount of time. Each simulation is run 30 times. At the end of each set of 30 simulation runs, the mean number of home runs (N_{hr}) is computed and extracted at a 95% confidence interval. Also extracted from the experiments are the number of squares traversed (C_{sqr}) to achieve the home run (N_{hr}) values. The C_{sqr} and the mean ordinate of the N_{hr} values, in conjunction with the C_{hr} value of the map under consideration are subsequently used to compute

the Functional *Accuracy* (A1). This is done using Equation (17). The Functional *Accuracy* along with the other two relevant metrics are compared to the threshold set in Section IX-B to arrive at a final certification statement for each type of AM.

It should be noted that pheromones deposited on the squares of the map are set to evaporate after 10 ticks for the evaluation carried out.

As mentioned in Section VII-A, the metrics evaluated are considered based on the collective behaviour of AMs of a similar type, not on individual AM performance.

XI. Evaluation Execution for Use-Case

The execution of the evaluation process defined in Sections VIII-X is repeated for two scenarios. In the first set of evaluations, the execution run time is fixed and the number of robots is varied for each type of AM. For the second set of execution, the simulation run time is varied, while the number of deployed robots is fixed.

A. Evaluation I

As noted previously, a robot is moved to the next square *en route* to the food source or nest at the tick of the clock. In this first evaluation, the number of ticks is held constant at 1200 ticks for each robot on both the Closed and Open maps. The number of robots deployed is steadily increased from 20 to 100 in intervals of 20. Since it is the collective behaviour of the robots that is being evaluated, if 20 robots are deployed in a scenario then there will be 24000 moves in total i.e., 20 robots $\times$ 1200 ticks. Table I shows the cumulative number of squares crossed (C_{sqr}) or moves by the robots based on their deployed numbers for both maps.

TABLE I
Cumulative Number of Squares (C_{sqr}) traversed by robots on the Closed and Open map

No. of deployed robots	Total No. of Moves
20	24000
40	48000
60	72000
80	96000
100	120000

From the graph shown in Figure 5, it can be seen that robots controlled by managers of types AM-P, AM-HC2 and AM-HC3 have the best performance in terms of the number of Home Runs (N_{hr}) achieved on the Closed map. The relative superior performance of these three AMs has to do with the quality of the mechanism by which the food source is found and the algorithm that guides the robot back to the nest. For instance, for the Closed map, managers of type AM-P and AM-HC3 have an advantage when it comes to finding the food source, as their navigational algorithms are biased towards paths with the strongest Pheromones. Type AM-HC2 and AM-HC3 managers are better at finding the nest on the return trip, primarily because both rely on the *Hill Climbing* algorithm to achieve this. Although, managers of type AM-HC1 also employ the *Hill Climbing* algorithm when trying to find the nest, its ability to achieve a comparative number of home runs when compared to AM-HC2 and AM-HC3 is hampered by the fact that it does not rely on Pheromones to find the food source.

The poor performance of managers of type AM-M is due to the fact that if its robots take the non-optimal route to find the food source, they will also take the non-optimal route back to the nest, as dictated by the **Memory** algorithm. The inability of AM-B to compete favourably with the other AM types is due to the mechanism

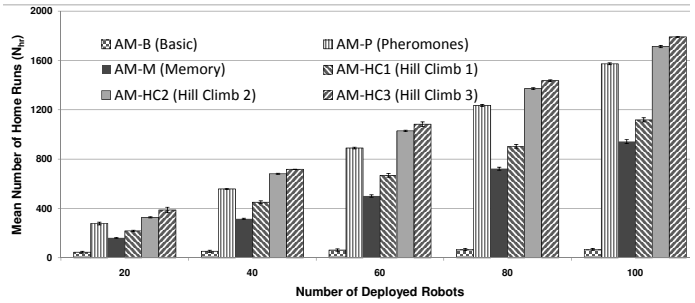


Fig. 5. Home Runs (N_{hr}) for the Closed Map

of the implemented **Basic** algorithm. Recall that managers with this algorithm have short-term memory with respect to where the robots have been. Another disadvantage is that the algorithm does not give higher weights to paths with the strongest pheromones. As expected for most of the AM types save AM-B, an increase in the number of deployed robots increases the number of home runs achieved.

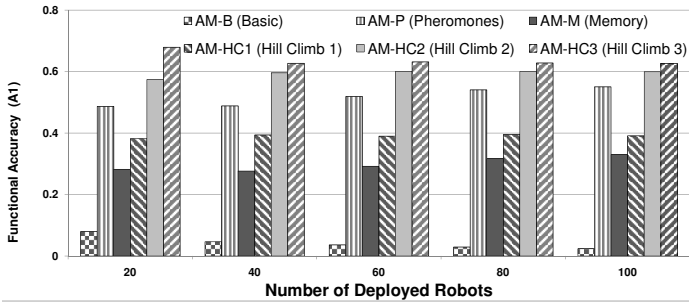


Fig. 6. Functional *Accuracy* (A1) for Closed Map

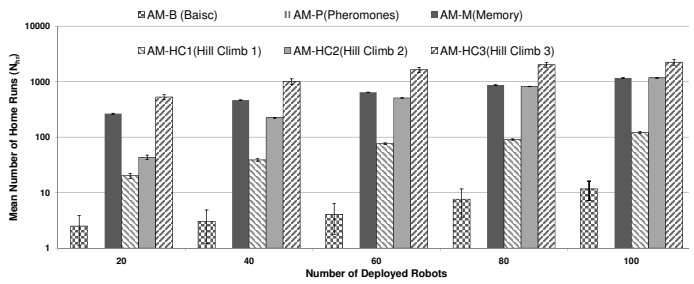


Fig. 7. Home Runs (N_{hr}) for the Open Map

The Functional *Accuracy* (A1) of the 6 types of AMs is computed and shown in Figure 6. These values of A1 are computed using Equation (17) with the mean ordinates of the Home Runs of Figure 5 as N_{hr}, the corresponding values of C_{sqr} in Table I and the C_{hr} value for the Closed map as inputs.

From the Figure and for the Closed map, it can be seen that only AM-P, AM-HC2 and AM-HC3 meet the 0.4 threshold set for R3 Functional *Accuracy* in Section IX-B.

The performances of the AMs on the Open map tell a slightly different story from the Closed map. The most significant difference is in the performance of the managers of type AM-P. In Figure 7, the number of home runs (N_{hr}) achieved by the AMs implementing the **Pheromones** algorithm is Zero. Recall that the Open map is a more complicated map, in that robots wander about; as there are many more paths here than those on the Closed map. A close examination of how

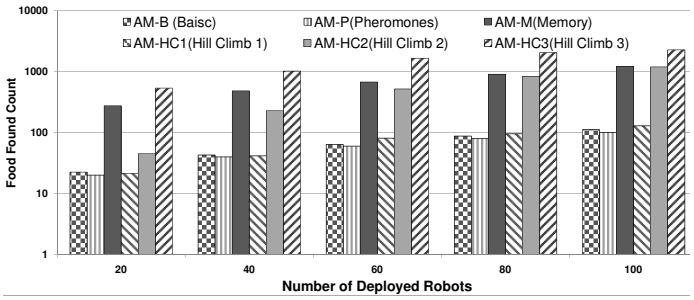


Fig. 8. Food Found Count for Open Map

the **Pheromones** navigational algorithm works revealed the reason for this subpar performance. Figure 8 shows the amount of times the food source was found by the various navigational algorithms. Notice that the AMs with the **Pheromones** algorithm found the food source a number of times. This suggests that the problem has to do with the robots not being able to find their way to the nest afterwards. This problem is further compounded by the fact that given the increased opportunity to roam on the Open map, robots with food are unable to find their way home will deposit pheromones on every square crossed. This in turn will lead other robots looking for the food source astray, as type AM-P managers depend heavily on paths with the strongest pheromones. Note that type AM-HC3 managers also have a bias towards paths with stronger pheromones but unlike the AM-P, the *Hill Climbing* algorithm implemented in managers of type AM-HC3 intelligently guides robots back to the nest. Hence, the reason for the higher home run counts for AM-HC3 shown in the figure. Another significant change of note is the performance of robots controlled by type AM-M managers. It appears the ability of the Memory algorithm to trace the path back to the nest, albeit suboptimally, was an advantage in terms of home runs achieved relative to the **Pheromones** algorithm as shown in Figure 7.

Again, the performance of robots managed by AM-B was relatively poor for the same reasons given for its performance in Figure 5. However, the non-reliance on the path with the strongest pheromones by the **Basic** algorithm is the reason for its relative higher performance in terms of home runs achieved when compared to those of managers of type AM-P.

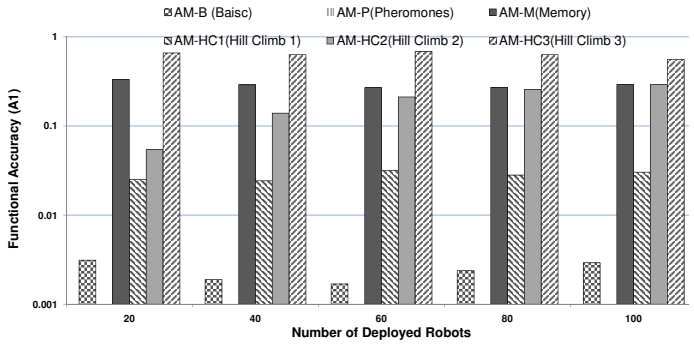


Fig. 9. Functional *Accuracy* for the Open Map

The Functional *Accuracy* (A1) for the Open map in this evaluation is shown in Figure 9. The computation of A1 was done in the same manner as that shown in Figure 6. Note that the $C_{sqr} = 30$ for the Open map (see Section IX-A). From Figure 9, it can be seen that only managers of type AM-HC3 meet the required threshold of 0.4 for Functional *Accuracy* set in Section IX-B.

B. Evaluation II

For the evaluation contained in this section, the number of robots was held constant at 20 and the number of clock ticks increased steadily from 24000 to 72000 in steps of 12000. As the robots are allowed to cover more ground (or squares) given the increased number of ticks, one would expect that the number of home runs achieved would increase as well. This is the case as shown in Figure 10 and 12 for the Closed and Open map, respectively. Even though the parameter varied in this evaluation is different from that varied in the evaluation of Section XI-A, the analysis of Evaluation I applies here as well. This can be verified from the graphs depicted in Figures 10 - 14.

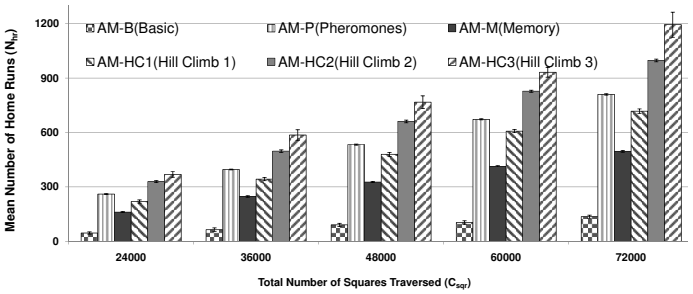


Fig. 10. Home Runs (N_{hr}) for the Closed Map

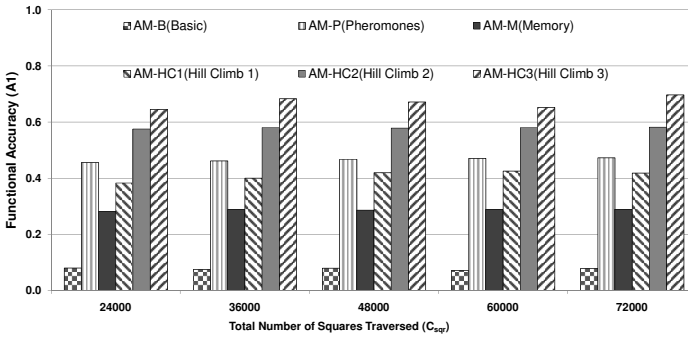


Fig. 11. Functional *Accuracy* for Closed Map

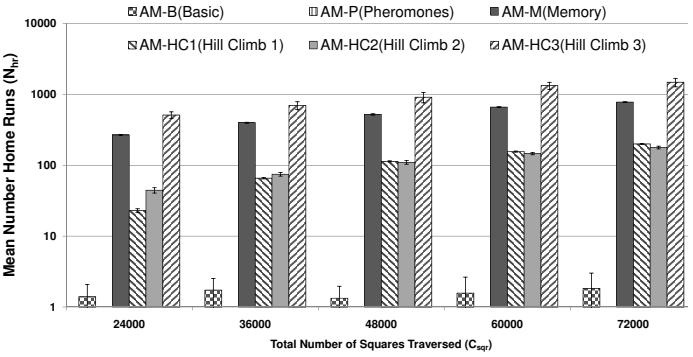


Fig. 12. Home Runs (N_{hr}) for the Open Map

From Figure 11, it can be seen that for the Closed map AM-P, AM-CH2 and AM-CH3 consistently meet the Functional *Accuracy* threshold set in Section IX-B i.e., 0.4. Managers of type AM-M meet

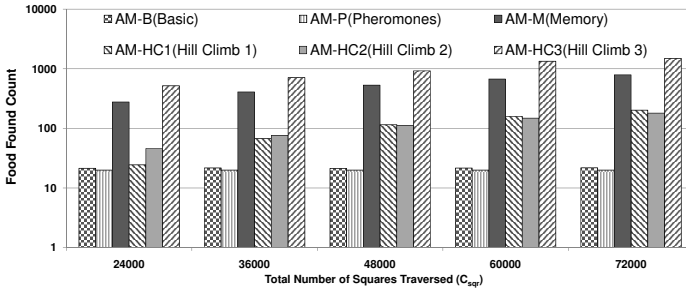


Fig. 13. Food Found Count for Open Map

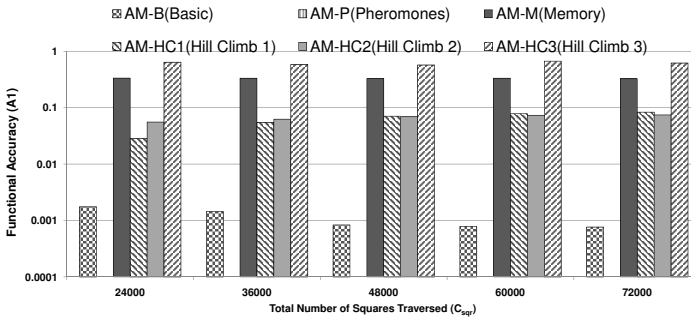


Fig. 14. Functional *Accuracy* for Open Map

this threshold only when the number of steps exceeded 24000. For the Open map, only managers of type AM-CH3 meet this requirement (see Figure 14).

C. Evaluated Autonomic Manager Acceptability

In Section IX, three metrics were identified as the basis for which an AM targeted at the PF application is accepted. These metrics were; Functional *Suitability* (FS), *R3* Functional *Accuracy* (A1) and Coupling and Modularity (C & M). The thresholds for acceptability for these three metrics were set in Section IX-B. All AMs targeting the PF application must meet the FS requirement (i.e., a 'Yes' value). For the A1 metric, all AMs must achieve a value of 0.4 or better in terms of the objective of the PF. The C & M, like the FS must have a value of 1 or a Yes.

TABLE II
MANAGER RATINGS FOR EVALUATION I (NUMBER OF DEPLOYED ROBOTS VARIED ON THE CLOSED MAP

AM Type	C & M	FS	A1 (Closed map)
AM-B(Basic)	✓	✓	
AM-P(Pheromones)	✓	✓	✓
AM-M (Memory)	✓	✓	
AM-HC1 (Hill Climb 1)	✓	✓	
AM-HC2 (Hill Climb 2)	✓	✓	✓
AM-HC3 (Hill Climb 3)	✓	✓	✓

From Section IX-A, it was made clear that all the AMs evaluated followed the architectural configuration of Figure 3, and as such, the C & M metrics is achieved (i.e., 'Yes') for all of them. The AMs were also said to be functionally suitable to the objective of the PF (see Section IX-A), therefore they all meet the FS attribute. The different performances of the AMs as it relates to the Functional *Accuracy* were presented in Sections XI-A and XI-B. Based on this, Tables II, III, IV and V show how each type of AM measured up to expectations with regards to the selected metrics for Evaluation I and II. Where a metric is met, the column for that metric is ticked for an AM type, and where it falls short, the column is left blank.

TABLE III
MANAGER RATINGS FOR EVALUATION I (NUMBER OF DEPLOYED ROBOTS VARIED ON THE OPEN MAP

AM Type	C & M	FS	A1 (Open map)
AM-B(Basic)	✓	✓	
AM-P(Pheromones)	✓	✓	
AM-M (Memory)	✓	✓	
AM-HC1 (Hill Climb 1)	✓	✓	
AM-HC2 (Hill Climb 2)	✓	✓	
AM-HC3 (Hill Climb 3)	✓	✓	✓

TABLE IV
MANAGER RATINGS FOR EVALUATION II (NUMBER OF CLOCK TICKS VARIED ON THE CLOSED MAP

AM Type	C & M	FS	A1 (Closed map)
AM-B (Basic)	✓	✓	
AM-P (Pheromones)	✓	✓	✓
AM-M (Memory)	✓	✓	
AM-HC1 (Hill Climb 1)	✓	✓	
AM-HC2 (Hill Climb 2)	✓	✓	✓
AM-HC3 (Hill Climb 3)	✓	✓	✓

TABLE V
MANAGER RATINGS FOR EVALUATION II (NUMBER OF CLOCK TICKS VARIED ON THE OPEN MAP

AM Type	C & M	FS	A1 (Open map)
AM-B (Basic)	✓	✓	
AM-P (Pheromones)	✓	✓	
AM-M (Memory)	✓	✓	
AM-HC1 (Hill Climb 1)	✓	✓	
AM-HC2 (Hill Climb 2)	✓	✓	
AM-HC3 (Hill Climb 3)	✓	✓	✓

Based on the contents of Tables II, III, IV and V, it can be said that only managers of type AM-HC3 consistently meet the rating levels set by the PF certifiers.

XII. CONCLUSION

To address the lack of a framework for certifying autonomic computing systems (ACSs) a novel five level Autonomic Maturity Index (AMI) was proposed and applied to the Intelligent Machine Design (IMD) architecture. These defined indices reflect how independent the AM is i.e, the amount of resources the AM is expected to expend in terms of intelligence, computational complexity and speed before it engages the human operator to solve a management task. An Expression that describes an autonomic machine was derived. The parameters for this expression include the AMI, the layer configuration of the machine and the implemented self-management properties. As a consequence, the derived expression indicates if the machine conforms to the dictates of the IMD architecture or not. If it does not conform, no further certification activity is carried out on the machine. If the machine's expression satisfies the established architectural rules, the certification process continues by measuring the machine's attributes relating to performance using the proposed quantitative metrics. How these metrics are computed and how the results are interpreted was discussed. These quantitative metrics are based on the six software quality characteristics of the ISO/IEC 9126-1998 specification i.e., the Functionality, Usability, Portability, Reliability, Efficiency and Maintainability characteristics. Evaluation steps that utilize the proposed quantitative and qualitative metrics for autonomic computing certification purposes were presented. These steps were

guided by the ISO/IEC 14598 software evaluation specification.

To demonstrate applicability of the architectural and metric systems proposed for building and certifying Autonomic Computing Systems (ACSs), an Ant Colony Optimization application called Path Finder (PF) was chosen. The main goal of managers targeted at the *PF* application is to guide robots from the nest to a food source and back. Six different managers were designed and implemented, each with a different navigational algorithm. The purpose was to compare the performances of these AMs to a specified rating level. From an architectural perspective, the *PF* demanded that each manager conform to Configuration V of the Intelligent Machine Design (IMD). Since the objective of the *PF* application is to find the most optimal route between the nest and the food source, only the self-optimization property of the four autonomic self-management attributes is implemented in the machine.

With respect to certification, all four procedures leading up to a final certification statement on the AMs were followed. Three metrics were identified as relevant to the evaluation of AMs targeted at the PF application. These metrics include; Functional *Suitability*, Functional *Accuracy* and *Coupling and Modularity*. Given the conformity of each type of AM to Configuration V of the IMD, each type of AM scored full marks for the *Coupling and Modularity* metric. They were also awarded full marks for Functional *Suitability*. For Functional *Accuracy*, each type of AM was evaluated and rated within the context of how well they performed when directing their robots on two different maps. Based on these three metrics, it was shown that only managers of type AM-HC3 met the specified rating threshold consistently.

APPENDIX

This appendix deals with a couple of special cases that may arise when trying to define an autonomic manager using Expression (7) presented in Section III-C.

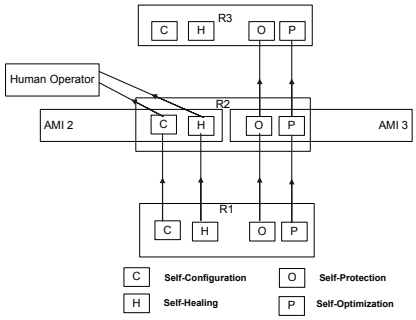


Fig. 15. Autonomic Manager 1 (AM_1) Before Normalization

Consider a situation where an AM is designed such that some of the self-management properties specify a particular AMI while others specify another. For instance, the designers of AM_1 shown in Figure 15, specify an AMI of 2 for the self-configuration and the self-healing properties in R2. The self-optimization and self-protection properties specify an AM_1 of 3. Since this machine specifies two AMIs it cannot be described using Expression 7 (see Subsection III-C).

$$AM_1 = \left\{ \begin{array}{l} ????; R1 \leftrightarrows R2 \leftrightarrows R3; \\ SM_{R1} = \{SC_1, SH_1, SO_1.SP_1\}, \\ SM_{R2} = \{SC_2, SH_2, SO_2, SP_2\} \\ SM_{R3} = \{\emptyset, \emptyset, SO_3, SP_3\} \end{array} \right\} \quad (18)$$

To rectify this anomaly, the machine shown in Figure 15 and partially described by Expression (18) must be normalized. The

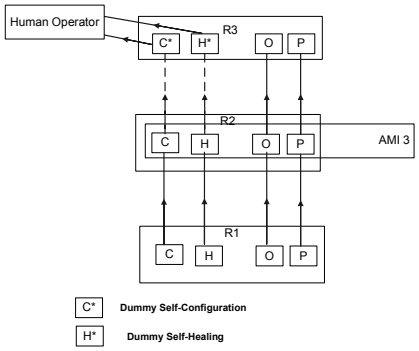


Fig. 16. Autonomic Manager 1 (AM_1) After Normalization

normalization process for AM_1 involves creating dummy implementations of the self-configuration and the self-protection functions in R3. These dummy implementations are empty functions that simply redirect any request from R2 to the human operator as shown in Figure 16. The normalized AM design is rated with the higher of the two AM_1 values it specified before normalization. The normalized expression for AM_1 is shown in (19).

$$AM_1 = \left\{ \begin{array}{l} 3; R1 \leftrightarrows R2 \leftrightarrows R3; \\ SM_{R1} = \{SC_1, SH_1, SO_1, SP_1\}, \\ SM_{R2} = \{SC_2, SH_2, SP_2, SP_2\} \\ SM_{R3} = \{SC_3^*, SH_3^*, SO_3, SP_3\} \end{array} \right\} \quad (19)$$

The asterisks in the SM_{R3} variable of Expression (19) signify dummy function implementation. A comparison between Figure 15 and 16 shows that the normalization process does not violate the intention of the AM designers.

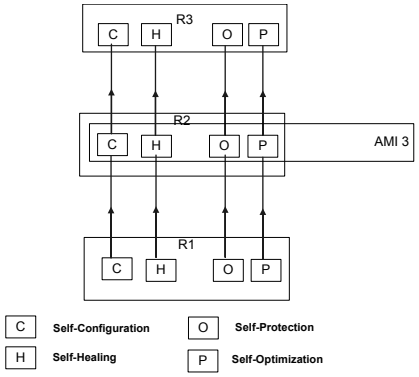


Fig. 17. Autonomic Manager 2 (AM_2)

$$AM_2 = \left\{ \begin{array}{l} 3; R1 \leftrightarrows R2 \leftrightarrows R3; \\ SM_{R1} = \{SC_1, SH_1, SO_1, SP_1\}, \\ SM_{R2} = \{SC_2, SH_2, SO_2, SP_2\} \\ SM_{R3} = \{SC_3, SH_3, SO_3, SP_3\} \end{array} \right\} \quad (20)$$

Recall that from the AMI of an expression describing an AM, one can quickly deduce a number of characteristics including how intelligent and complex the system being observed is. Speaking strictly from an AMI perspective, expressions (19) and (20) erroneously convey a sense of equivalence between AM_1 in Figure 16 and AM_2 in Figure 17. In fact, since AM_1 is able to rely on the human operator for some of its management function in R2; it should be judged lower than AM_2. In a case like this, some of the

quantitative metrics discussed in Section V will ensure that AM_2 is rated relatively superior to AM_1 at the end of the certification process. For instance, when the self-management functions at R3 of AM_1 are being measured for the functionality accuracy metric(A), the dummy functions will be rated with a zero value (see Subsection V-A2). This will not be the case for AM_2; thus ensuring its higher overall rating.

ACKNOWLEDGMENT

The authors would like to extend their appreciation to the anonymous contributor aptly named *pseudonym67* [18] for the navigational algorithm library and the user interface used in the evaluation aspect of this work.

REFERENCES

[1] H. Shuaib, R. J. Anthony, and M. Pelc, "A framework for certifying autonomic computing systems," *The Seventh International Conference on Autonomic and Autonomous Systems: ICAS 2011*, pp. 122–127, May 2011.

[2] B. T. Clough, "Metrics, schmetrics! how the heck do you determine a uavs autonomy anyway?," *Proceedings of the Performance Metrics for Intelligent Systems Workshop, Gaithersburg, Maryland*, 2002.

[3] IBM, "An architectural blueprint for autonomic computing," *IBM Whitepaper*, June 2006.

[4] Autonomic Research Group, University of Greenwich *http://cms1.gre.ac.uk/research/autonomics/tech.html. Latest Access: December 20th, 2012*.

[5] ISO/IEC 9126, "Software engineering Product quality (ISO/IEC 9126)," tech. rep., International Organization for Standardization/International Electrotechnical Commission, 1998.

[6] ISO/IEC 14598, " Information technology – Software product evaluation (ISO/IEC 14598)," tech. rep., International Organization for Standardization/International Electrotechnical Commission, 1999.

[7] H. Shuaib and R. Anthony, "Towards Certifiable Autonomic Computing Systems Part I: A Consistent and Scalable System Design," *International Journal On Advances in Intelligent Systems*, vol. 5, December 2012.

[8] R. W. Proud, J. J. Hart, and R. B. Mrozinski, "Methods for determining the level of autonomy to design into a human spaceflight vehicle: A function specific approach," *PerMIS 03. Proc. Performance Metrics for Intelligent Systems, NIST Special Publication 1014*, September 2003.

[9] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing degrees, models, and applications," *ACM Computing Surveys*, vol. 40(3), August 2008.

[10] IBM, "An architectural blueprint for autonomic computing," *IBM Whitepaper*, 2004.

[11] H.-M. Huang, E. Messina, R. Wade, R. English, B. Novak, and J. Albus, "Autonomy measures for robots," *Proceedings of IMECE: International Mechanical Engineering Congress (IMECE2004-61812)*, November 2004.

[12] R. Parasuraman, T. B. Sheridan, and C. D. Wickens, "A model for types and levels of human interaction with automation," *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICSPART A: SYSTEMS AND HUMANS*, vol. 30, pp. 286–297, MAY 2000.

[13] W. Truszkowski, L. Hallock, C. Rouff, J. Karlin, J. Rash, M. G.Hinchey, and R. Sterritt, *Autonomous and Autonomic Systems*. Springer, 2009.

[14] B. Moore, E. Ellesson, LongBoard-Inc., J. Strassner, A. Westerinen, and Cisco-Systems, "Policy core information model – version 1 specification (RFC 3060)," February 2001.

[15] B. Moore and IBM, "Policy core information model (PCIM) extensions (RFC 3460)," January 2003.

[16] K. Zeilenga and OpenLDAP Foundation, "Lightweight directory access protocol (LDAP): Technical specification road map (RFC 4510)," June 2006.

[17] M. Wahl, A. Coulbeck, T. Howes, S. Kille, Critical Angle Inc., Netscape Communications Corp., and Isode Limited, "Lightweight directory access protocol (v3): Attribute syntax definitions (RFC 2252)," December 1997.

[18] pseudonym67 *http://www.codeproject.com/KB/recipes/pathfinderBypseudonym67.aspx. Latest Access: December 20th, 2012*.