A Framework for Certifying Autonomic Computing Systems

Haffiz Shuaib, Richard J Anthony and Mariusz Pelc
School of Computing and Mathematical Sciences
The University of Greenwich
Park Row, Greenwich, London SE10 9LS, UK
Email: {*H.Shuaib,R.J.Anthony,M.Pelc*}*@gre.ac.uk*

***Abstract*—The growing cost associated with managing increasingly complex Information Technology (IT) infrastructure has brought about a new field of research – Autonomic Computing (AC). AC systems seek to mimic the management capabilities of biological systems notably, the Autonomous Nervous System (ANS), to bend the curve associated with management cost, while also allowing these managed systems to evolve and adapt, regardless of the operational context and deployment environment. The AC field attempts to bring together a number of disparate research fields in order to achieve its objectives. For this to be possible, a way to coherently link the imports of these dissimilar fields is required, if these systems are to be understood, and by extension trusted. To that end, this project's focus is on the proposal of mechanisms that will allow for proper certification of autonomic computing systems. This, if successful, will provide a consistent trust measure for these systems. Particularly, in this position paper the intelligent machine design architecture is extended and used as a basis for defining autonomic computing systems. Metrics contained in the International Standard Organization (ISO) 9126-1998 specification, are appropriated to AC systems. Based on all of the above, a foundation for achieving certification for AC systems is laid.**

***Keywords*-Autonomic Computing, Architectures, Frameworks.**

I. Introduction

A true Autonomic Computing (AC) system 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 evolve into more complex systems to handle more difficult tasks, without significantly increasing the cost of management.

The ability to self-configure, self-optimize, self-protect and self-heal, have been identified as the four cardinal characteristics of AC systems [1]. As with most critical or increasingly complex systems, an AC system should and must be certified on the basis of its expected characteristics before it goes live, as these systems have implications from the financial to the space exploration industries. Achieving this system certification task is made more challenging still, given that the AC field itself draws from a disparate number of well established fields, including artificial intelligence, telecommunications research, mathematics, software/hardware engineering, statistics etc. each with its own view of how a system is to be defined.

No attempt has been made towards the certification of autonomic computing systems thus far; therefore, the primary objective of this project is the provision of a technical and visible support structure for the relevant components of these disparate fields, to facilitate the certification process of AC systems. It is hoped that the final result of this project will provide a basis; for assessing 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.

The rest of this paper is organized as follows; the operational mechanisms of AC system are expected to mimic those of the biological Autonomic Nervous System (ANS), relevant aspects of the ANS are discussed in the next section. The state of the art as it relates to AC system architecture is discussed in Section III. Also presented in this section, is a proposed extension of the Intelligent Machine Design as a basis for AC system architecture. In Section IV, the challenges associated with the certification of AC systems are discussed. Further discussed are suggested metrics by which, compliant AC systems can be measured. Section V contains the conclusion and future work.

II. Autonomic Nervous System

The Autonomous nervous system (ANS) in part inspires the Autonomic Computing (AC) field; as a result, it is pertinent that it is discussed briefly in this paper. This discussion will be restricted to those components of the ANS that are believed to be relevant to AC systems.

The ANS relies on three major components to achieve its independent management objectives; Neurons, synapses, inter-neuron communication protocols and excitatory and inhibitory mechanism that modulate the output or response of the ANS for a given input signal. The Neurons are the processing units and the basic building blocks of the ANS. These neurons are connected to one another by directional links called synapses. Information is conveyed through the synapses in the form of Ions, which, have a similar external structure but may differ in internal chemical composition. The excitatory and inhibitory mechanism modulates the output or response of the ANS for a given input signal. In effect, regardless of the magnitude of the input signal, the output response can be graded to meet an overall objective.

A few salient concepts from the ANS that apply to ACs are as follows: A standard communication channel, similar to the ANS synapse that can carry information to and from the AC's processing entities or the AC's equivalent of neurons. This is especially important, as these entities may be spread over several physical locations or even spread over a number of disparate hardware and software platforms. Recall, that the synapses convey information in the form of ions, which, internally may differ in terms of chemical composition. A dynamic description language is required in this regard. As will be shown later in this work, a policy definition language based on the Extensible Markup Language (XML) is appropriate for this. A means to have an all or nothing response or a graded response based on the input information from the sensory mechanism(s) and acquired knowledge of the systems is also required. A means to marshal resources to areas most in need in an emergent scenario and back to a steady state is also desirable. These concepts when implemented technically must be portable and platform agnostic.

III. A Reference Architecture for an Autonomic Computing Element

An architectural standard is central to the process of the certification of a system. Any architecture that represents an autonomic computing system must not be narrowly defined such that it precludes the ability for the system to evolve or cater for new use cases. As noted in [2] and [3], the lack of an open standard is a challenge in the autonomic computing field. In this section, the most prevalent of all autonomic architectures i.e., IBM's MAPE architecture is discussed. Drawbacks relating to this architecture are also discussed, and a certifiable alternative architecture with similar functionalities is presented.

Before proceeding to the next sub-section a few definitions are in order;

Definition 1: **Autonomic Manager:** This component independently makes decisions that are then effected on a managed component.

Definition 2: **Autonomic Element:** This consists of both the management (autonomic manager) and managed components.

Definition 3: **Goal:** This is the overall objective of the autonomic system. For instance, the overall goal of a vehicle might be to get from a point A to a Point B. How this is achieved is left to low-level functionalities.

Definition 4: **Rule:** A rule is a ***Boolean*** statement that evaluates to true or false, with an action taken or not taken based on the outcome [4].

Definition 5: **Policy:** Finally, a policy is made up of a number of rules. The primary objective of policies is to achieve the overall system goal in the most efficient way possible [4].

A. IBM's Autonomic Architecture

The well known IBM MAPE (Monitor, Analyze, Plan, Execute) AC architecture consists of four main components which, form a loop, as shown in Figure 1.The first of these components is the Monitor. Its main duty is to monitor the surrounding environment, including system resources. The output of this Monitor is used for making decisions at later stages of the loop. The second component i.e., the Analyze component, uses a number of algorithms to anticipate problems and possibly proffer solutions to these problems. The Planning component uses the information available to the autonomic system to choose which, policies to execute. The Execution component, which, is the fourth component, effects the most appropriate policy/policies chosen by the system. This executed policy may cause a change in the physical environment e.g., moving the arm of a robot, or simply pass instructions or information to another element, possibly an autonomic one. The input to the MAPE architecture comes from the sensory mechanism, while the effector mechanisms carry out the dictates of the machine.

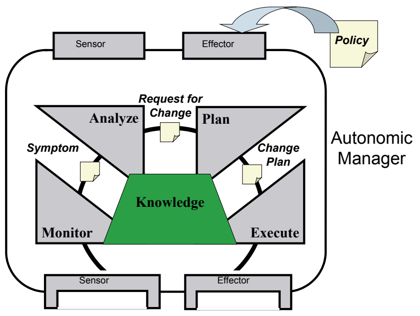

Figure 1. IBM Autonomic MAPE Architecture [1]

The functionality associated with any one of these component can be delegated wholly to the human manager or the autonomic system. The level of dependence on the machine or the human operator is calibrated using a metric called the Autonomic Maturity Index (AMI).

IBM defines five indices in this regard [1]. They are as follows;

- Manual (Level 1): At this level, the human operator is completely responsible for all functional components of the MAPE architecture.
- Instrument and monitor (Level 2): Here, the autonomic system is responsible for the collection of information (Monitoring). This collected/aggregated information is analyzed by the human operator and guides future actions of the operator.
- Analysis (Level 3): In this level, information is collected and analyzed by the system. This analyzed data is passed to the human administrator for further actions.
- Closed loop (Level 4): 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 5): 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 a guide.

In the early days of the AMI i.e., 2003, it was believed that most computing systems resided at the Basic level [5]. However, research prototypes conforming to higher levels are said to exist currently [6].

There is no consensus on whether the IBM MAPE architecture is a concrete architecture or a malleable concept. For instance, [7] defines the architecture as canon and then goes on to implement it to the letter. [8] and [9] both assume MAPE to be a concrete architecture that can be tweaked. For example, [8] collapses the four main components of the MAPE architecture into two groups. The Monitor/Analyze components are placed into one group, with this group given the responsibility of handling issues like fault diagnoses and anomaly detection. The Plan/Execute group deals with issues relating to resource-allocation and configuration. [9], in a similar manner to [8] breaks the architecture into two main groups, but this time the Monitor/Analyze components handle tasks that are reactive, while the Plan/Execute components are responsible for proactive adaptation. [10] adopts a slightly different approach to modifying the architecture. The authors of [10] divide the architecture into a global and local sub-architecture, with the Analyze/Planning and Monitor/Execute components implemented in the global and local sub-architectures, respectively.

As a concept, the IBM architecture is said to require concrete expressions for it to be applicable to the framework discussed in [11]. The MOSES framework [12] which, seeks to manage quality of service expectations in a volatile environment also falls under the concept school of thought. It maps each component of the MAPE architecture to its own architecture. [13] proposes to make the MAPE architecture real or concrete by generalizing the model for large-scale data processing infrastructures. In [14], the MAPE architecture is seen as a concept that can be applied to an architecture, not as an architecture itself.

Beyond the concept/concrete architecture argument, some researchers have stated that the MAPE architecture is flawed, or at least insufficient to describe autonomic systems. For example, [15] consider the architecture concrete, but too narrowly defined to apply to some autonomic systems e.g., multi-agent systems. [16] points out that the loop in the MAPE architecture is vulnerable to failure, which, in turn can precipitate the collapse of the management system all together. A solution to this problem might be a situation where an outer loop monitors the internal loop as was done in [8]. However, this begs the question, is another outer, outer loop needed to also monitor the outer loop and so on and so forth? Rather than using external and rigid feedback loops to make the MAPE loop more resilient, the authors of [16] propose the addition and the removal of loops on the fly. Their solution is inspired by biological emergent systems described in [17]. [18] also adds a partial outer loop to the MAPE architecture to allow the system to evolve.

Regardless of its wide applicability, the divergent views associated with the IBM MAPE architecture makes it ill-suited for the certification of autonomic systems.

The IBM's AMI, while critical to the certification process, is said to be narrowly defined and technically vague [5]. This makes it difficult to align an autonomic system with these maturity indices [19]. Obviously, these concerns do not help the certification process. An attempt is made in the next section to address the above. It is pertinent to mention here that several other forms of accessing the level of autonomicity have also been proposed in the past, notably those in [20]. Here, the level of autonomicity is defined by who makes the decisions and how these decisions are executed [21]. From a certification perspective, the position of a system on the autonomic maturity index should be defined by who makes the decisions and the quality of the decisions themselves. Both metrics will engender a certain level of trust in the system.

B. An Alternative Architecture

The appeal of the Intelligent Machine Design (IMD) architecture [22] to autonomic computing systems is that it is closely related to the way intelligent biological systems work. Indeed, this architecture has been suggested as a generic framework on which, autonomic systems can be built upon [23]. While this architecture is mentioned in some autonomic computing literature, nothing concrete from a technical perspective has been achieved relative to IBM's architecture (See Figure 2). In this section, an attempt is made to imbue this architecture with the self-management

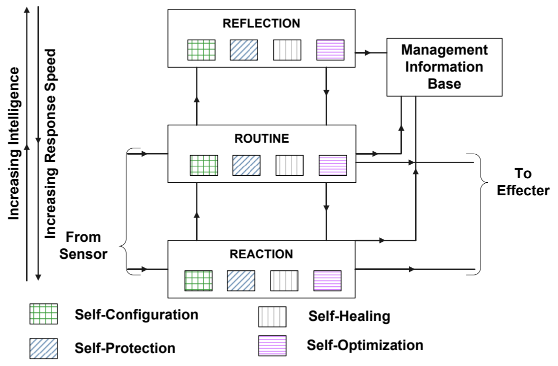

Figure 2. An Autonomic Computing expression of the IMD

autonomic properties and further relate it to a proposed AMI. Before going into the technical details, a quick overview of the make up of the architecture is presented.

The architecture supposes that an intelligent machine is made up of three distinct layers, i.e., Reaction, Routine and the Reflection level. Each layer can be characterized by the following attributes; the amount of resources consumed, their ability to activate/inhibit the functionality of a connected layer and their ability to be activated or inhibited by another layer. Attributes that deal with the preoccupation of each level are also added later in this section.

The lowest layer, the Reaction layer, is connected to the sensors and effectors. When it receives a sensory stimulus, it responds relatively faster than the other two layers. The primary reason for this is that its internal mechanisms are simple, direct and hardwired i.e., it has an automatic response to incoming signals. A simple IF-ELSE statement suffices here. In autonomic computing parlance, the human operator does much of the learning or monitoring/analysis. The Reaction layer takes precedence over all other layers and can trigger higher layer processing. This layer can also be inhibited/activated by the Routine layer. It consumes the least amount of resources. It might also have a lmited access to the working memory or the management information base (MIB).

The Routine (mid-level) layer is more learned and skilled when compared to the Reaction layer. It is expected to have access to the working memory (or MIB), which, contains a number of policy definitions that can be executed based on context, knowledge and self-awareness. As a result, it is comparatively slower than the Reaction level. Its activities can be activated or inhibited by the Reflection layer. Its input comes from both the sensory mechanism and the Reflection layer. Its output goes to the effector mechanism and the Reflection layer. When the Routine level is unable to find a suitable policy for an immediate objective, it hands control over to the human administrator or the Reflection layer.

While the Routine level's primary objective is to deal with expected situations whether learned or hardwired, the Reflection level, which, is the highest level, helps the machine deal with deviations from the norm. The Reflection level is able to deal with abnormal situations, using a combination of learning technologies (e.g., Artificial Neural Networks, genetic algorithms), partial reasoning algorithms (e.g., Fuzzy Logic, Bayesian reasoning), the machine's knowledge base, context and self-awareness. Technically, the Reflection Layer's ultimate aim, as it relates to autonomic computing systems, is to create and validate new policies at runtime that will be used at the Routine level. If the system is able to adapt to an unexpected situation as a result of the new policy, then the policy is stored in the working memory (or MIB). This new policy can be called upon if the situation is encountered in the future. Thus, making a formerly abnormal situation a routine one. The process of 'reasoning' out a new policy makes the Reflection layer the largest consumer of computing resources. This also means it has the slowest response time of all three layers. The Routine layer is the input source and output destination for the Reflection layer. The Reflection layer can inhibit/activate the processes of the Routine layer through new policy definitions. Higher layers are able to excite or inhibit the activities of the lower layers in a manner similar to how the ANS is modulated by knowledge or the conscious mind. Notice from Figure 2 that all layers will be able to implement the four cardinal self-management properties. Also, in the same way in which, the ANS communicates between neurons using ions irrespective of their chemical composition, autonomic computing elements implementing the architecture shown in Figure 2 should also be able to communicate with one another using a standard mechanism. In addition, the old and new policies must use a standard definition language to ensure consistency across the board. The policy framework defined in RFC 3060/3460 is the vehicle by which, the above is to be achieved.

C. The Alternative Architecture and the AMI

The architecture shown in Figure 2 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 machine. 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 level. This maturity index corresponds to the Reaction level.
- **Maturity Index 2:** This index corresponds to the Routine level. If the Routine level is unable to find a suitable policy from a policy repository or if there is a policy ambiguity, it relies on the human administrator to provide a new solution or resolve the policy 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 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 problem of the Routine layer, and monitor the implementation of this new policy. If the policy fails in its objective or if a new policy 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 is not found or created, the algorithm within the Reflection layer will continually attempt to create a new policy or resolve the policy 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, 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. Recall, that it was said that the source and quality of the decisions made is central to the trust placed on the system, and as a result relevant to the certification process (See the last paragraph of Section III-A).

IV. Verifying, Validating and Certifying Autonomic Computing Systems

The ultimate goal of this project is to define formal mechanisms, where possible, by which, autonomic systems can be certified. The road to certification, of course, is a long one that entails building, verifying and validating the system(s) in question. Before discussing the challenges associated with certifying autonomic systems, it is pertinent to define what is meant by validation, verification and certification. According to the IEEE standard glossary for Software Engineering terms [24], the ***Validation*** process seeks assurances that the system has been built as per the initial requirements set out. In other words, is the end-product fit for the purpose for which, it is built. The purpose of the ***Verification*** process is to ensure that the system performs correctly and consistently in terms of the input and output. ***Certification*** is defined as a written guarantee, a formal demonstration or the process of confirming that an offered component complies with specified requirements, and will work as expected in a defined environment. Clearly, certification encompasses the verification and validation processes.

The validation and verification techniques used for many large-scale software projects will suffice for some aspects of the autonomic systems, still other areas will require new constructs. For instance, classical testing techniques such as dynamic, static testing, formal methods will be well suited to levels 1 and 2 of the Autonomic maturity index defined in Section III-C. Nevertheless, these techniques are unsuitable for levels 3-5. The primary reason for this is that the behaviour of the system is expected to adapt, and change over time due to the activities of the Reflection layer. This implies that a means for these systems to self-validate within the context of the defined goal is required. One way to go about verifying/validating these systems is by separating the architecture from the algorithm contained within.This is similar to what was done in the DySCAS project [25], in which, a middleware to support self-configuring automotive systems was developed. Here the autonomic decision-making logic was developed and validated independently of the actual middleware mechanisms. For example, if the Reflection layer of a product implements an artificial neural

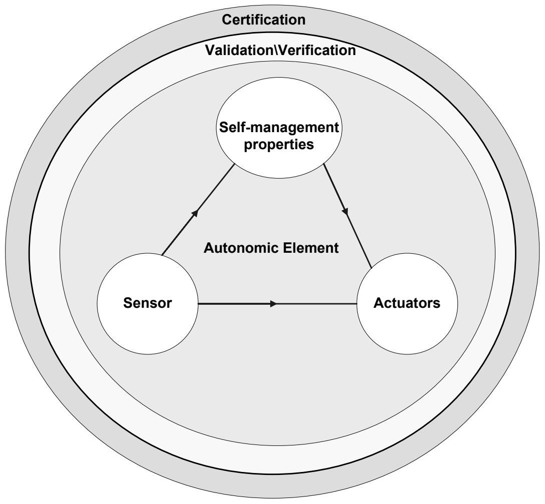

Figure 3. Verification, Validation and Certification

network algorithm, one can first check to see if the architecture implemented in the product meets a certain criteria e.g., portability, stability etc. using already established methods. If the architecture meets expectations, the arduous task of validating and verifying the algorithm follows. Otherwise, if the implemented architecture fails to meet expectations, the product should be deemed to have failed the test, and therefore there is no need to proceed to validate the algorithm implemented within. Validating/Verifying the algorithm e.g., artificial neural networks or genetic algorithms, on the other hand would require new formal methods of verification/validation or at least an extension of the known techniques. Some research efforts have been expended in this regard. Jacklin et al. [26] propose that any testing method for artificial neural networks, whether new or modified must take cognizance of the following; the level of detail and the correctness of the training data, the impact of data outside the training set on the system, the amount of memory available to hold the data set being operated upon, and the impact of the network algorithm on associated components. Needless to say, a strong grasp of the fundamentals of these learning algorithms is mandatory for this exercise.

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 as highlighted in [27]. Several metrics have been identified, using the ISO 9126-1998 standard as a guide [28], that can be measured within the context of the four cardinal self-* properties of autonomic systems i.e., self-configuration, self-protection, self-healing and self-optimization. [28] defines six main characteristics that could be used to assess the quality of a software product, including; Functionality, Usability, Portability, Reliability, Efficiency and Maintainability. The Usability, Reliability and Maintainability attributes are implicit in the definition of autonomic systems, because of this, only the attributes relating to Functionality, Portability and Efficiency will be considered here.

The Functionality attribute consists of the following characteristics; suitability to the problem and interoperability. The speed of response, processing times and throughput of the system are characteristics of the Efficiency attribute. Finally, the Portability component consists of adaptability, installability, co-existence and replaceability. All have a sub-attribute that deals with compliance to standards. These three main attributes can and should be applied to each of the four main self-* properties during the validation, verification and certification process.

V. Conclusion and Future Work

Relative to currently deployed IT systems, autonomic computing system are expected to exhibit superior control/management behaviour and high adaptability, regardless of the operational context. However, a means for measuring this superiority is lacking. In this paper, two areas that allow for these assessments were tackled i.e., the architecture and the metrics that will allow for certification.

In this paper, an evaluation of the way in which, MAPE is not well-suited for certifiable systems was provided. The intelligent machine design architecture was discussed, and shown to be amenable to certification. Five technical autonomic maturity indices, that are similar in broad strokes to those proposed by IBM, were also presented and aligned to the intelligent machine design architecture. As it relates to metrics of autonomic systems, three of the six metrics defined in ISO 9126-1998 i.e., Functionality, Efficiency and Portability, were shown to be relevant to assessing autonomic systems, and are under ongoing investigation.

The work presented in this paper has laid the foundation on which, the certification of autonomic systems can be pursued. However, there is still much work to be done. The next tasks include;

- Firmly establishing the intelligent machine design architecture for autonomic systems such that it enforces structure but does not restrict the self-management algorithms implemented within. Clearly defining interfaces between any two autonomic elements or managers, or between any two levels on the architecture.
- Propose and implement mechanisms that will allow for efficient management coordination among elements of an autonomic system. These mechanisms will allow for proper establishment of administrative relationships between elements, will provide components that allow for management conflict resolution, if two or more autonomic managers simultaneously effect changes on a single managed system etc.
- Define and demonstrate steps by which autonomic computing systems are to be rated within the context of the targeted application domain. In the first instance, qualitative measures that fully describe the architectural make up of each manager and their associated maturity indices will be proposed. These qualitative metrics in turn will point to what quantitative measures or performance characteristics can be obtained from the machine under an evaluation scenario. As discussed previously, the quantitative measures will be based on ISO 9126-1998.

References

[1] IBM, *An architectural blueprint for autonomic computing.* IBM Corporation, 4 ed., June 2006.

[2] M. Salehie and L. Tahvildari, “Autonomic computing: Emerging trends and open problems,” *DEAS’05. Workshop on the Design and Evolution of Autonomic Application Software*, vol. 30, pp. 1–7, 2005.

[3] R. Sterritt, “Autonomic computing,” *Innovations System Software Engineering (2005), Springer-Verlag*, vol. 1, no. 1, pp. 79–88, 2005.

[4] R. J. Anthony, “A policy-definition language and prototype implementation library for policy-based autonomic systems,” *ICAC ’06. IEEE International Conference on Autonomic Computing*, pp. 265 – 276, 2006.

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

[6] B. Dillenseger, T. Coupaye, M. Salaun, and M. J. Barros, “Autonomic computing and networking:the operators’ vision on technologies, opportunities, risks and adoption roadmap,” *Eurescom study report*, p. 21, 2009.

[7] C. Reich, K. Bubendorfer, and R. Buyya, “An autonomic peer-to-peer architecture for hosting stateful web services,” *CCGRID ’08. 8th IEEE International Symposium on Cluster Computing and the Grid*, 2008.

[8] C. Kennedy, “Decentralised metacognition in context-aware autonomic systems: some key challenges,” *In American Institute of Aeronautics and Astronautics (AIAA) AAAI-10 Workshop on Metacognition for Robust Social Systems, Atlanta, Georgia*, 2010.

[9] P. de Grandis and G. Valetto, “Elicitation and utilization of utility functions for the self-assessment of autonomic applications,” *ICAC ’09. International Conference on Autonomic Computing*, 2009.

[10] C. Dorn, D. Schall, and S. Dustdar, “A model and algorithm for self-adaptation in service-oriented systems,” *ECOWS ’09. Seventh IEEE European Conference on Web Services*, pp. 161 – 170, 2009.

[11] B. Pickering, S. Robert, S. Mnoret, and E. Mengusoglu, “Model-driven management of complex systems,” *MoDELS’08. Proceedings of the 3rd International Workshop on Models@Runtime , Toulouse, France*, pp. 277 – 286, October 2008.

[12] V. Cardellini, E. Casalicchio, V. Grassi, F. L. Presti, and R. Mirandola, “Qos-driven runtime adaptation of service oriented architectures,” *ESEC/FSE ’09. Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009.

[13] R. Nzekwa, R. Rouvoy, and L. Seinturier, “Modelling feedback control loops for self-adaptive systems,” *Third International DisCoTec Workshop on Context-Aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (2010) 7*, vol. 28, 2010.

[14] V. K. Naik, A. Mohindra, and D. F. Bantz, “An architecture for the coordination of system management services,” *IBM SYSTEMS JOURNAL*, vol. 43, no. 1, pp. 78–96, 2004.

[15] R. Quitadamo and F. Zambonelli, “Autonomic communication services: a new challenge for software agents,” *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 3, pp. 457–475, 2008.

[16] B. A. Caprarescu and D. Petcu, “A self-organizing feedback loop for autonomic computing,” *IEEE CS’09. In Proceedings of the Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, vol. 126–131, 2009.

[17] R. J. Anthony, “Emergence: a paradigm for robust and scalable distributed applications,” *ICAC’04. IEEE International Conference on Autonomic Computing*, pp. 132–139, 2004.

[18] C. Dorn and S. Dustdar, “Interaction-driven self-adaptation of service ensembles,” *CAiSE’10. 2nd International Conference on Advanced Information Systems Engineering*, 2010.

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

[20] T. B. Sheridan, *Telerobotics, Automation, and Human Supervisory Control*. The MIT Press, 1992.

[21] 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.

[22] D. A. Norman, A. Ortony, and D. M. Russell, “Affect and machine design: Lessons for the development of autonomous machines,” *IBM Systems Journal*, vol. 42, no. 1, pp. 38 – 44, 2003.

[23] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland, “A concise introduction to autonomic computing,” *Elsevier Journal on Advanced Engineering Informatics,*, pp. 181–187, 2005.

[24] IEEE Standards Board, “IEEE Standard Glossary of Software Engineering Terminology (IEEE Std. 610.121990),” Tech. Rep. 610.121990, IEEE, August 1990.

[25] R. Anthony, D. Chen, M. Pelc, M. Persson, and M. Torngren, “Context-aware adaptation in dyscas,” *CAMPUS’09. Proceedings of the Second International DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services*, vol. 19, 2009.

[26] S. A. Jacklin, M. R. Lowry, J. M. Schumann, P. P. Gupta, J. T. Bosworth, E. Zavala, J. W. Kelly, K. J. Hayhurst, and C. M. Belcastro, “Verification, validation, and certification challenges for adaptive flight-critical control system software,” *In American Institute of Aeronautics and Astronautics (AIAA) Guidance, Navigation and Control Conference and Exhibit, number AIAA 2004-5258, Providence, Rhode Island*, August 2004.

[27] 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.

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