Natural Engineering

Applying a Genetic Computing Model to Engineering Self-Aware Software

Abhi Sharma
Avaamo Inc.
Los Altos, CA, USA
a@sharma.com

Giovanni Moranna
C3DNA
Santa Clara, CA, USA
giovanni@c3dna.com

ABSTRACT

Current approaches to software engineering use a Turing machine implementation to intelligently monitor and adjust the internal environment of an algorithm in real-time. These same approaches however, fail to account for fluctuations in the external environment of the computation, leading to a gross underutilization of system resources or requiring a full restart with costly supervision and manual intervention. In this paper, we describe how we can provide the same intelligence for non-functional requirements as there exist for functional requirements in software applications by using the Distributed Intelligence Computing Element (DIME) computing model. By discussing this model in comparison to similar systems in nature, namely in the context of genetics, we develop the concept of services engineering with self-managed software. As a particularly salient example of this model in practice, we explore the potential for such an approach to improve the performance of machine and deep learning algorithms as a function of intelligent computing environments.

CCS CONCEPTS

• **Computing methodologies** → *Intelligent agents*; • **Software and its engineering** → **Cloud computing**;

KEYWORDS

Intelligent network, DIME, DIME network architecture, recursion hierarchy, connectivity, modularity, Turing Machine, Turing O-Machine

ACM Reference Format:
Abhi Sharma and Giovanni Moranna. 2018. Natural Engineering: Applying a Genetic Computing Model to Engineering Self-Aware Software. In *SE4COG'18: SE4COG'18:IEEE/ACM 1st International Workshop on Software Engineering for Cognitive Services , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, Article 5, 4 pages. https://doi.org/10.1145/3195555.3195560

1 INTRODUCTION

While the idea of endowing our technology with intelligence may not be particularly novel, what is novel, however, is our ability to actually accomplish this with some degree of success. The relatively quick ascent of artificial intelligence (AI) in modern science and society is due largely to the decreasing cost of computation and increasing availability of system resources via the oft-cited "Moore's Law". What began as a rather modest application of AI to solve very specific problems in constrained environments has since grown into a frenzied attempt to apply ever grander and more distributed algorithms to an unimaginably vibrant set of industries. However, the performance requirements, size, and architectural demands of our learning machines have grown in tandem, and it is becoming prohibitively expensive to ignore the ensuing rise in complexity, both with respect to time and capital [5, 6, 8].

The field of AI and in particular machine learning, has thus far concerned itself with engineering intelligent software, the maintenance of which is largely outsourced to human beings [9]. We propose that it is time for us to consider applying intelligence to software engineering itself and in this paper, advocate for self-managing software with a nested intelligent agent architecture. Our ideas are organized in three parts. We begin with a first-principles approach to develop the characteristics of a system capable of self-management. We then explore a model that displays these characteristics and finally, discuss a particular instantiation of this model in practice.

2 THE INTELLIGENT NETWORK

Software engineering can be economically said to consist of two related processes, namely, the satisfaction of functional and non-functional requirements. Functional requirements describe the expected behavior of a system, whereas nonfunctional requirements concern themselves with the way in which these behaviors are manifested. Stated differently, the former dictates what needs to be done, while the latter outlines how it will be done. This dichotomy however, is not only prevalent in AI or computer science, and in fact emerges time and time again in the social and natural sciences as well. A particularly salient example comes to us from the study of genetics [3, 5], wherein the human genome is separated into coding and noncoding regulatory sequences, the latter occupying a disproportionately large amount of genomic real estate (here we have chosen not to discuss "junk DNA", the other major contributor to noncoding sequences). Similarly in business, while product development can drive profitability in large companies, the bulk of an organization's resources are employed in managerial maintenance [12]. Since the emphasis of our paper is on the second of this recurring pair, we will focus on the theme of non-functional regulation and the characteristics of a system which enable it to possess this quality.

SE4COG'18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5740-1/18/05…$15.00
https://doi.org/10.1145/3195555.3195560

Based on the presence of this property in most systems with some form of regulation, we arrive at the first principle of our proposed system of self-management, namely, that it exhibit self-awareness. As long as a system consists of only two entities, self-awareness can be the straightforward consequence of a chain of command that flows from the regulator to the regulated. Defined in this way, self-awareness is simply the ability of a system to receive and process information regarding its own functioning. However, system complexity and the subsequent gaps in oversight increase as a function of the number of regulated entities when only one regulator is present. We discover rather quickly then, that while regulation is a necessary condition for self-awareness, it is not a sufficient one. Recursive regulation on the other hand, affords us a much more robust method of ensuring awareness through a nested hierarchy of regulators [3, 6]. An entity is regulated by another, which in turn is controlled by yet another entity, higher up in the hierarchy. In a distributed environment, using recursion allows top-level regulators to accumulate global knowledge and use information about one region to selectively influence behavior in another, providing uninterrupted and fine-tuned monitoring. The updated first principle of our self-managed system is therefore that it exhibit self-awareness through recursive regulation.

Self-management requires that an entity be aware not only of itself, but also of the environment in which it resides. Resisting perturbation, and a tendency to return to equilibrium are the hallmarks of homeostatic systems ranging from the biological to the ecological [3, 11]. The human body resists temperature fluctuations and graciously handles chemical imbalances as readily as the oceans absorb carbon dioxide in response to increased levels of the greenhouse gas in our atmosphere. A self-managed system incapable of displaying such resilience is tantamount to one constantly exposed to the risk of failure. We therefore further modify our first principle to describe an intelligent system that has awareness, both internal and external, with recursive regulation. Internal awareness is the monitoring of processes within an entity, while external awareness refers to the system's ability to understand its environment.

In order to maintain equilibrium, our system must also be able to store and process information, and effectively react to stimuli in its environment. In this way, the next two principles of our self-managed system can be derived rather simply from the first of their kind, all of which can be summarized as follows:

(1) Exhibit internal and external awareness with recursive regulation
(2) Be capable of knowledge representation and information processing
(3) Display goal-driven behavior

We now turn our attention to the tools with which we must equip our self-managed system, if it is to display the behaviors outlined above, namely, a signaling pathway and an organizational hierarchy [3, 9]. As the number of regulated entities grows, so too must the number of regulators, albeit at a smaller rate if we allow a "many to one" relationship. In light of this, it becomes imperative that there exist a means of communication between the regulators, which facilitates information sharing [2]. We subsequently arrive at the first tool necessary to turn our intelligent system into an intelligent network, capable of truly reacting to its internal and external environment: a policy-based, directed, signaling pathway. Once this pathway is in place, our intelligent network can then use the second tool to effectively implement recursive regulation.

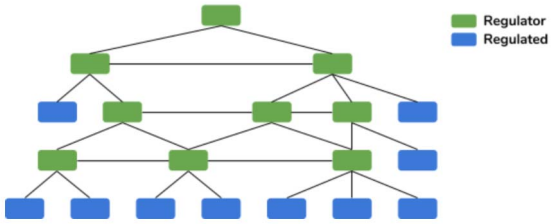

Figure 1: Networked regulators and regulated entities.

Organizational hierarchies appear in entity, management pairs, and occur both naturally, and in man-made institutions. For every city, state, or nation, there exists a system of governance. For every human body, there is nervous, endocrine, and genetic regulation. And for every Turing machine, we can describe Turing's O-machine [1, 11, 13]. Such an organizational structure with nested layers of management agents and managed entity pairs would allow our intelligent network to efficiently and adaptively exhibit all the principles of a self-managed system while maintaining homeostasis. **Figure 2** shows a hypothetical configuration of regulators and regulated entities with a recursive organizational hierarchy.

3 DIME NETWORK ARCHITECTURE

Prior to describing the Distributed Intelligent Computing Element (DIME) computing model, we will first address the shortcomings of current large-scale solutions for distributed software development, which lack an intelligent network architecture with the principles of self-management described in the previous section. The increasingly demanding nature of distributed applications with extremely stringent requirements for system uptime and/or privacy and security concerns has spurred the emergence of virtualization and cloud computing technologies such as VMware, Amazon Web Services, etc [7, 10, 12, 14].

While these new technologies provide automated provisioning, elastic workload balancing, auto-scaling, auto-failover, and migration services, there are several drawbacks. Ensuring that the right resources and availability requirements are met for a computing entity in real-time requires the concerted effort of a whole host of services which, if owned by a single provider, introduces vendor lock-in, stifling innovation and limiting cross-compatibility with other providers for unmet service needs [9, 10]. Contrastingly, if all the required services are provided by competing vendors with bloated compatibility software as overhead or with open-sourced solutions, the ensuring complexity is sure to drive the cost of infrastructure management ever higher, while deteriorating overall system integrity through the numerous points of failure now present [12, 14].

The coupling of functional and non-functional requirements in the software engineering process as in the above scenario can cause unmet non-functional requirements to significantly degrade application performance itself. By dissociating these two classes

of requirements and using a self-managed intelligent system with our desired principles, we can potentially avoid these issues. The DIME computing model provides just this type of self-management through the use of a DIME element, which acts as a regulator, and a Managed Intelligent Computing Element (MICE), which acts as the regulated entity [6]. In addition to extensive information processing capabilities and monitoring the fault, configuration, accounting, performance, and security (FCAPS) requirements of MICE, the DIME computing model also provides support for recursive regulation, and both inter- and intra-DIME communication through signaling pathways [7, 9]. The synchronous updating of internal states across multiple DIMEs in an architecture, coupled with the awareness of even the most distant regions in the network, allows software applications to react to FCAP failures gracefully, and propagate the necessary service modifications to all relevant nodes, providing resilience to fluctuations in system resources and intelligent adaptability [14]. Put differently, distant DIME elements are updated to reflect local changes in the application environment, which allows the network on the whole to exhibit resiliency and adaptability.

In this way, the model decouples functional and non-functional requirements, significantly abstracts away complexity, lowers the cost of infrastructure management, and provides flexibility through modular DIMEs that can work directly with an application while concurrently sharing information with, and modifying the behavior of, DIMEs in other recursively nested layers of the software stack. Using this computing model therefore, software applications can expect automated workload provisioning and mobility, independent of who provides the compute or persistent storage [9]. By applying such a cognitive overlay to a group of MICE or regulated entities, the DIME network architecture therefore exhibits all the desired behaviors of an intelligent network and overcomes the limitations of current distributed software engineering solutions.

An example of this implementation with self-managing workloads in a multi-cloud environment is described in this video: https://youtu.be/tu7EpD_bbyk. We observe how the effective use of the DIME computing model can allow a particular application area network to span multiple cloud providers, a number of different countries or zones, as well as separate operating systems. In an example of the decoupling of functional and non-functional requirements, we see how a three-tiered application built on Apache, Tomcat, and MySQL is able to migrate efficiently from one cloud provider to another, independent of the relative locations of individual layers in the technology stack, without service interruptions. While this is initially done with the help of an active system administrator, the same behavior is then replicated with a policy-based approach, endowing the application area network with self-regulatory capacity, the essential quality of an intelligent network. A policy, such as the one described in the video, allows the application to maintain awareness of system resources and the computing environment, while autonomously responding and migrating the entities under its management in response to increasing workloads. Since the architecture of the application and its goals are decoupled, this behavior is effectively exhibited without having to rely on the proprietary tools offered by each cloud provider.

4 APPLICATION OF INTELLIGENT NETWORK IN AI

The DIME computing model and its principles are also prevalent in the multi-agent architectures employed by many cutting edge AI algorithms. An interesting example of this approach is evident in an application of deep reinforcement learning seen in the recent paper from DeepMind by Espeholt et al. [4]. Drawing inspiration from the study of reward pathways in neuroscience, as well as behavioral psychology, reinforcement learning deals with a class of machine learning algorithms, which train intelligent agents to display optimal behavior in diverse environments as a function of the reward associated with each action. In order to train a single agent on multiple tasks, the researchers at DeepMind developed an Importance Weighted Actor-Learning Architecture (IMPALA), which separated the actors from the learner in an agent attempting to learn parameters for a set of varied challenges in a visually similar environment, including maze navigation, or memory tasks. The distributed agents (acting as MICE) communicated their experiences to a central learner (a DIME) who could in return also be distributed [4].

This separation of actors and learners is also not unlike the decoupling of functional and non-functional requirements of software engineering we observe in the DIME computing model and interestingly this approach improved the throughput of training data being sent from various actors to the central learner, providing greater data efficiency overall. Furthermore, actors performing certain tasks helped improve the performance of actors in related but different tasks in an example of transfer learning, as the IMPALA learner adapted behavior in one region in its network based on information from another [4, 9]. Transfer learning, in this capacity, is the ability of a model in one task to perform well on another.

After compensating for learning differences introduced by the decoupled agent architecture, IMPALA performed better than previous architectures using a new V-trace off-policy actor-critic algorithm. IMPALA was also better at resource management, and is able to scale extremely well to numerous machines without degraded performance on the DMLab-30 suite of tasks [4].

5 CONCLUSION

The Holy Grail [15-16] of software engineering is the ability to create self-managing (autonomous systems) which, manage themselves while also managing their environment to optimally utilize the resources available and execute their intent. Such a system must adopt to fluctuations both in its workload demands and also available resource pools at scale. As John von Neumann pointed out [17] "It is very likely that on the basis of philosophy that every error has to be caught, explained, and corrected, a system of the complexity of the living organism would not last for a millisecond. Such a system is so integrated that it can operate across errors. An error in it does not in general indicate a degenerate tendency. The system is sufficiently flexible and well organized that as soon as an error shows up in any part of it, the system automatically senses whether this error matters or not. If it doesn't matter, the system continues to operate without paying any attention to it. If the error seems to the system to be important, the system blocks that region out, by-passes it, and proceeds along other channels. The system

then analyses the region separately at leisure and corrects what goes on there, and if correction is impossible the system just blocks the region off and by-passes it forever. The duration of operability of the automation is determined by the time it takes until so many incurable errors have occurred, so many alterations and permanent by-passes have been made, that finally the operability is really impaired. This is completely different philosophy from the philosophy which proclaims that the end of the world is at hand as soon as the first error has occurred."

DIME computing model with cognitive control overlay has demonstrated the self-management of a response to the ephemeral nature of distributed computing caused by the non-deterministic impact of environmental interaction with the system. IMPALA has demonstrated that such a cognitive control overlay is also useful in improving scale, resiliency and efficiency of machine learning algorithm implementations.

We propose that the software engineering practices be reexamined to leverage the cognitive control overlay architecture to improve the scale, resiliency and efficiency of next generation software systems without the need for rebooting.

REFERENCES

[1] A. M. Turing, "Systems of logic defined by ordinals," Proc. Lond. Math. Soc., Ser. 2, 45, pp. 161-228, 1939.

[2] C. Frasson, T. Mengelle, E. Aimeur, and G. Gouarderes, "An actor-based architecture for intelligent tutoring systems," In International Conference on Intelligent Tutoring Systems, Springer, Berlin, Heidelberg, 1996, pp. 57-65.

[3] H. Mengistu, J. Huizinga, J. B. Mouret, and J. Clune, "The evolutionary origins of hierarchy," PLoS computational biology, 12(6), e1004829, 2016.

[4] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, ... and S. Legg, "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures." arXiv preprint arXiv:1802.01561, 2018.

[5] M. Burgin, and E. Eberbach, "On foundations of evolutionary computation: an evolutionary automata approach," in Hongwei Mo (Eds.), Handbook of Research on Artificial Immune Systems and Natural Computing: Applying Complex Adaptive Technologies, IGI Global, Hershey, Pennsylvania, pp. 342-360, 2009.

[6] M. Burgin, and R. Mikkilineni, "Semantic Network Organization based on Distributed Intelligent Managed Elements", In Proceeding of the 6th International Conference on Advances in Future Internet, Lisbon, Portugal, 2014, pp. 16-20. Burgin, Super-recursive Algorithms, New York: Springer, 2005.

[7] P. Goyal, R. Mikkilineni and M. Ganti, "FCAPS in the Business Services Fabric Model," Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE '09. 18th IEEE International Workshops on, Groningen, 2009, pp. 45-51.

[8] R. Mikkilineni, "Going beyond Computation and Its Limits: Injecting Cognition into Computing." Applied Mathematics 3, 2012.

[9] R. Mikkilineni and G. Morana, "Infusing Cognition into Distributed Computing: A New Approach to Distributed Datacenters with Self-Managing Services on Commodity Hardware (Virtualized or Not)," WETICE Conference (WETICE), 2014 IEEE 23rd International, Parma, 2014, pp. 131-136.

[10] R. Mikkilineni and V. Sarathy, "Cloud Computing and the Lessons from the Past," Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE '09. 18th IEEE International Workshops on, Groningen, 2009, pp. 57-62.

[11] R. Mikkilineni, A. Comparini and G. Morana, "The Turing O-Machine and the DIME Network Architecture: Injecting the Architectural Resiliency into Distributed Computing," In Turing-100. The Alan Turing Centenary, (Ed.) Andrei Voronkov, EasyChair Proceedings in Computing, Volume 10, 2012.

[12] R. Mikkilineni, G. Morana and D. Zito, "Cognitive Application Area Networks: A New Paradigm for Distributed Computing and Intelligent Service Orchestration," Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2015 IEEE 24th International Conference on, Larnaca, 2015, pp. 51-56.

[13] R. Mikkilineni, G. Morana, and M. Burgin. "Oracles in Software Networks: A New Scientific and Technological Approach to Designing Self-Managing Distributed Computing Processes," In Proceedings of the 2015 European Conference on Software Architecture Workshops (ECSAW '15). ACM, New York, NY, USA, Article 11 , 8 pages, 2015.

[14] V. Sarathy, P. Narayan and R. Mikkilineni, "Next Generation Cloud Computing Architecture: Enabling Real-Time Dynamism for Shared Distributed Physical Infrastructure," Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on, Larissa, 2010, pp. 48-53.

[15] J.O. Kephart and D. Chess, "The Vision of Autonomic Computing," Computer, vol. 36, no. 1, 2003, pp. 41âĂŞ50.

[16] Huebscher, M. C. And Mccann, J. A. "A survey of Autonomic Computing-Degrees, Models, and Applications" ACM Computing Surveys, Vol. 40, No. 3, Article 7, August 2008.

[17] Aspray, W., & Burks, A. (1989). Papers of John von Neumann on Computing and Computer Theory. Cambridge, MA: MIT Press.