Injecting the Architectural Resiliency into Distributed Autonomic Systems

using DIME Network Architecture

Rao Mikkilineni
KawaObjects Inc
Los Altos, CA (USA)
rao@kawaobjects.com

Giovanni Morana
University of Catania
Catani, Italy
giovanni.morana@dieei.unict.it

Abstract—By introducing signaling and self-management in a Turing node and a signaling network as an overlay over the computing network, the current von-Neumann computing model is evolved to bring the architectural resiliency of cellular organisms to computing infrastructure. The DIME computing model introduces the genetic transactions of replication, repair, recombination and reconfiguration to program self-resiliency in distributed computing systems executing a managed workflow. The injection of parallelism and network based composition of “Self” identity are the first steps in introducing the elements of homeostasis and self-management in the computing infrastructure. DIMEs inject the architectural resiliency of cellular organisms to create a new class of distributed autonomic computing systems using managed Turing machine networks.

Keywords-component; DIME; Distributed Autonomic Systems; Turing Machine; Bio-inspired Computing

I. INTRODUCTION

Autonomic computing by definition implies two components in the system: the observer (or the Self) and the observed (or the environment) with which the observer interacts by monitoring and controlling various aspects that are of importance. It also implies that the observer is aware of systemic goals to measure and control its interaction with the observed. In living organisms, the autonomic behavior is attributed to the self and to the consciousness that contribute to defining one’s multiple tasks to reach specific goals within a dynamic environment and adapting the behavior accordingly. The concept of self and self-reflection which models the observer, the observed and their interactions is quite different from a third party observation of a set of actors or agents interacting with each other.

This paper examines recent advances in biology and neuro-science that attempt to explain consciousness and apply some of the lessons learned to model distributed computing systems in incorporating the features that contribute to self-aware interactions of computing elements.

II. CONSCIOUSNESS IN DISTRIBUTED SYSTEMS

As recent advances in neuroscience throw new light on the process of evolution of the cellular computing models, it is becoming clear that communication and collaboration mechanisms of distributed computing elements played a crucial role in the development of self-resiliency, efficiency and scaling which are exhibited by diverse forms of life from the cellular organisms to highly evolved human beings.

According to Damasio [3,4], managing and safe keeping life is the fundamental premise of biological entities. At the base of these features there is the homeostasis process. Homeostasis is the property of a system that regulates its internal environment and tends to maintain a stable, constant condition of properties like temperature or chemical parameters that are essential to its survival. System-wide homeostasis goals are accomplished through a representation of current state, desired state, a comparison process and control mechanisms.

Consciousness plays an important role in all these processes: the knowledge about the internal (self) and external (environment) conditions is the key element for understanding which action (or sequence of actions) has to be performed to reach a specific goal given the current situations.

This brings to light the cellular computing model that:

1. spells out the computational workflow components as a stable sequence of patterns that accomplishes a specific purpose,
2. implements a parallel management workflow with another sequence of patterns that assures the successful execution of the system’s purpose (the computing network to assure biological value with management and safekeeping),
3. uses a signaling mechanism that controls the execution of the workflow for gene expression (the regulatory, i.e. management, network) and
4. assures real-time monitoring and control (homeostasis) to execute genetic transactions of replication, repair, recombination and reconfiguration [13].

The connection between consciousness and computing models is succinctly summarized by Samad and Cofer in [12]. They say that autonomous systems designed for emulating the humanlike characteristics needs elements providing consciousness. In that work, authors point to two theoretical limitations of formal systems that may inhibit the implementation of computational consciousness and hence limit our ability to design human-like autonomous systems.

The first one is directly related to the halting problem in the current computational model based on Turing Machines. The second limitation is due to the Gödel’s incompleteness theorem. An important implication of this theorem is that it is not possible to have a finite description with the description itself as the proper part. In short, Gödel’s theorems prohibit “self-reflection” in Turing machines.

In [1], Barrett highlights the difference between Turing Machines implemented using von Neumann architecture and biological systems. She argues that the Turing machines based on algorithmic symbolic manipulation using von Neumann architecture, gravitate toward those aspects of cognition, like natural language, formal reasoning, planning, mathematics and playing chess, in which the processing of abstract symbols is in a logical fashion but leaves out other aspects of cognition that deal with producing adaptive behavior in a changeable environment. Unlike the approach where perception, cognition and action are clearly separated, she suggests that the dynamic coupling between various elements of the system, where each change in one element continually influences every other element’s direction of change has to be accounted for in any computational model that includes system’s sensory and motor functions along with analysis. To be fair, such couplings in the observed can be modeled and managed using a Turing machine network and the Turing network itself can be managed and controlled by another serial Turing network. What is not possible is the tight integration of the models of the observer and the observed with a description of the “self” using parallelism and signaling that are the norm and not an exception in biology.

A more interesting controversy that has erupted regarding the need for new computing models [2,7,17] throws some new light on the need for re-examining the Turing machines, Gödel’s prohibition of self-reflection and von Neumann’s conjecture. As we describe later, those authors are attempting to address how to model computational problems that cannot be solved by a single Turing machine but can be solved using a set of Turing machines interacting with each other. In particular, the property of being aware of one’s multiple tasks and goals within a dynamic environment and of adapting behavior accordingly which is related to consciousness is one such problem that a single Turing machine cannot solve.

The insights into biology suggest that in order to model temporal dynamics of the observer and the observed while also assuring the safe-keeping of the observer (with a “self” identity) requires modifications to the Turing machine to accommodate changes to the behavior while computation is still in progress.

III. SELF, SELF-REFLECTION AND SELF-MANAGEMENT

Self-reflection, setting expectations, monitoring the deviations and taking corrective action are essential for managing the business of life through homeostasis and evolution has figured out how to encapsulate the right descriptions to execute the life’s processes using the genetic transaction of replication, repair, recombination and reconfiguration by exploiting parallelism and signaling.

Self-reflection is another key component in living organisms. Homeostasis is not possible without a dynamic and active representation of the observer and the observed.

A cellular organism is the simplest form of life that maintains an internal environment that supports its essential biochemical reactions, despite changes in the external environment. The cell adapts to its environment by recognition and transduction of a broad range of environmental signals, which in turn activate response mechanisms by regulating the expression of proteins that take part in the corresponding processes.

The regulatory gene network forms a cellular control circuitry defining the overall behavior of the various cells. The complex network of neural connections and signaling mechanisms collaborate to create a dynamic, active and temporal representation of both the observer and the observed with myriad patterns, associations and constraints among their components.

It seems that the business of managing life is more than mere book-keeping that is possible with a Turing machine. It involves the orchestration of an ensemble with a self-identity both at the group and the component level contributing to the system’s biological value. It is a hierarchy of individual components where each node itself is a sub-network with its own identity and purpose that is consistent with the system-wide purpose. To be sure, each component is capable of book-keeping and algorithmic manipulation of symbols. In addition, identity and representations of the observer and the observed at both the component and group level make system-wide self-reflection possible.

In short, the business of managing life is implemented by a system consisting of a network of networks with multiple parallel links that transmit both control information and the mission critical data required to sense and to control the observed by the observer. The data and control networks provide the capabilities to develop an internal representation of both the observer and the observed along with the processes required to implement the business of managing life. The organism is made up of autonomic components making up an ensemble collaborating and coordinating a complex set of life’s processes that are executed to sense and control both the observer and the observed.
The system:

1. consists of components with a purpose within a larger system (specialization); all of a component parts must be present for the system to carry out its purpose optimally,
2. parts of it must be arranged in a specific way for the system to carry out its purpose (separation of concerns),
3. changes in response to feedback (collect information, analyze information and control environment using specialized resources), and
4. maintains its stability (in accomplishing its purpose) by making adjustments based on feedback (homeostasis).

Literature is filled with discussion about Gödel’s prohibition of self-reflection in Turing machines and why consciousness cannot emerge from the brain models that depend on Turing machines.

There are many theories on how the human brain is unique and may even involve quantum phenomena or gravity waves [14,15]. However Damasio [4] takes the evolutionary approach to discuss genomic unconsciousness, the feeling of conscious will, educating the cognitive conscious, the reflective self and its consequences. He affirms that the cultural developments and the automated homeostasis manifest the same goal in that they react to an imbalance detection seeking to correct it within the constraints of “human biology and of the physical and social environment”.

Instead of adding to the already existing controversy [14] on consciousness, we take a different route using Damasio’s emphasis on homeostasis along with the dynamic representation of the observer and the observed. We apply them to extend the Turing machine and its von Neumann Serial computing implementation. We ask how we can utilize the abstractions that assist in the business of managing life in cellular organisms, discussed above, to enhance the resiliency of distributed computing systems. In the next section we analyze the current implementation of Turing machines and suggest adding some of the abstractions that have proven useful in managing life’s processes to develop a computing model that addresses the problem of being aware of one’s multiple tasks and goals within a dynamic environment and of adapting behavior accordingly.

IV. TURING MACHINES, SUPER TURING MACHINES AND DIME NETWORKS

While a single stored program control (SPC) node lacks self-reflection prohibited by Gödel’s theorems, a network of Turing machines have been successfully used to implement business workflows that observe and manage the external world. This is accomplished by modeling the observed (external to the computing infrastructure) and orchestrating the temporal dynamics of the observed. This has helped us develop complex control systems that, in turn, can be monitored and controlled with the resiliency of cellular organisms.

However, what is missing is the same resiliency in the infrastructure (or the observer) that implements the control of the observed. In order to introduce consciousness, we must also introduce the “self” identity of the observer and its multiple tasks and goals within a dynamic environment and of adapting behavior accordingly.

The evolution of computing seems to follow a similar path to cellular organisms in the sense that it emerged as an individual computing element (von Neumann stored program control (SPC) implementation of the Turing machine) and evolved into today’s networks of managed computing elements executing complex workflows that monitor and control external environment.

The Turing machine originally started as a static closed system [7] analogous to a single cell. It was designed for computing algorithms that correspond to mathematical world view. As soon as an operating system is introduced, the Turing machine SPC implementation immediately becomes a workflow of computations implementing a process, where each process now behaves as a new Turing machine with SPC implementation. It is as if the operating system is a manager (implementing a management workflow using a group of management Turing machines dedicated for this purpose) controlling a series of computing Turing machines based on policies set in the operating system. The operating system instructions and the computational flow dependent instructions are mixed to serially execute the process and a sequence of processes. This is analogous to the evolution of multi-cellular organisms where individual cells establish a common management protocol to execute their goals with shared resources. The individual processes may or may not have a common goal but they share the same resources. The operating system communicates with the processes to exert its role using shared memory. While the individual processes do not have fault, configuration, accounting, performance and security management of self, the operating system provides these functions using the signaling abstractions of addressing, alerting, mediation and supervision.

Since then, multi-threading in a single processor, networked and interactive computing have influenced the computations. In a network, concurrency and influence of one node on another (impact of the environment on the computation) are the new elements that have to be addressed. The Pi calculus and super Turing models [6] are an attempt to address these aspects. While these attempts are embroiled in controversy [2], what is not in dispute is that a network of computers represents a network of organized Turing machines where each node is a group of Turing machines managed locally.

In such a network, the local operating systems cannot provide Fault, Configuration, Accounting, Performance and Security (FCAPS) management of the system as whole. The disciplines of distributed computing and distributed systems management evolved to address the FCAPS management of the system in an ad-hoc manner without a formal computing
model for the system as a whole. This is even more complicated when the system as a whole now acts in unison with a system-wide purpose where one element can influence other elements as pointed out in [1].

In this case, the description of functions performed and the influence of one computation on another have to be encoded at compile time and each computing element does not have the ability to change the behavior at run time.

In addition, operating system function is to allocate the resources appropriately to the consumers (processes running applications) and the applications themselves do not have any influence on the resources during run time. For example, if the workload fluctuates, the application has no way of monitoring and controlling the resources.

If multiple applications are contending for resources, external policies have to be implemented as other Turing machines and the applications themselves are not aware of these external influences.

In order to manage distributed set of Turing machines, another set of Turing machines are introduced to provide service management to improve fault, configuration, accounting, performance and security characteristics of the distributed system. See figure 1.

Taking the cue from cellular biology, we can introduce self-management into the Turing machine that assures resiliency of the Turing node. This requires a parallel monitoring and control mechanism to observe and control the Turing node and a control or signaling channel to collaborate with other Turing nodes to participate in a system-wide FCAPS management that assures all the Turing nodes participating in a workflow management of the observed are also managing themselves to assure the resiliency of the observer network. We stipulate that “The DIME computing model allows the specification and execution of a recursive composition model where each computing unit at any level specifies and executes the workflow at the lower level.

The specification at a higher level eliminates the self-reflection prohibition of Gödel’s theorems on computational units.

The parallel implementation of the management workflow and the computational workflow at each level allows the influence of one component in the workflow to influence another component at the lower level.

At any level, the computational unit specifies and assures the execution of the lower level workflow thus it becomes the observer observing and controlling the workflow execution at lower level (which is the observed).”

This stipulation eliminates the problem of separation of communication among the computing system components in a system and the communication between the computing system and its environment.

Figure 2 shows the new computing model we call Distributed Intelligent Managed Element (DIME) network computing model and the resulting computing infrastructure is designed with DIME network architecture.

The DIME Network Architecture (DNA) [5,8,9,10,11] consists of four components:

- a DIME node which encapsulates the von Neumann computing element with self-management of FCAPS (see Figure 2).
- signaling capability that allows intra-DIME and Inter-DIME communication and control,
- an infrastructure that allows implementing distributed service workflows as a set of tasks, arranged or organized in a DAG and executed by a managed network of DIMEs and
- an infrastructure that assures DIME network management using the signaling network overlay over the computing workflow.

The self-management and task execution (using the DIME component called MICE, the Managed Intelligent Computing Element) are performed in parallel using the stored program control computing devices.

Figure 1: A network of TMs implementing a service workflow that manages the external environment (the observed). The management of the observer is implemented using the same serial TM.

Figure 2: A Distributed Intelligent Managed Element (DIME)
The DIME encapsulates the “dispositional know-how.” Each DIME is programmable to control the MICE and provide continuous supervision of the execution of the programs executed by the MICE. The FCAPS management allows to model and to represent dynamic behavior of each DIME, the state of the MICE and its evolution as a function of time based on both internal and external stimuli. The parallel management architecture allows the observer (a network or sub-network) that forms a group to monitor and control itself while facilitating the implementation of monitoring and control of the observed in external environment. Parallelism allows dynamic information flow both in the signaling channel and the external I/O channels of the Turing computing nodes.

There are three special features of DNA that contribute to self-resiliency:

1. each Turing computing node is controlled by the FCAPS policies set in each DIME. Each read and write are dynamically configurable based on the FCAPS policies.
2. each node itself can be a sub-network of DIMES with goals set by the sub-network policies.
3. the signaling allows dynamic connection management to reconfigure the DIME network thus changing the policies and behavior.

A single node of a DIME that can execute a workflow by itself or by instantiating a sub-network provides a way to implement a managed DAG (Directed Acyclic Graph) executing a workflow.

It is easy to show that the DIME network architecture supports the genetic transactions of replication, repair, recombination and rearrangement. Replication is implemented by executing the same service in two (or more) different DIMEs as shown in figure 3.

By defining service S2 to execute itself, we replicate S2 DIME. Note that S2 is a service that can be programmed to terminate instantiating itself further when resources are not available.

In addition, FCAPS management at runtime (parallel service monitoring and control) allows changing overall service behavior by executing the control commands and replacing desired service components. We can also redirect I/O dynamically during run time. Any DIME can also allow a sub-network instantiation and control as shown in figure 4, where the service S2 (and its management policies) is changed, at run time, with service S1.

The workflow orchestrator instantiates the worker nodes, monitors heartbeat and performance of workers and implement fault tolerance, recovery, and performance management policies. It can also implement accounting and security monitoring and management using the signaling channel. Redirection of I/O allows dynamic reconfiguration of worker input and output thus providing computational network control.

In summary, the dynamic configuration at DIME node level and the ability to implement at each node, a managed directed acyclic graph using a DIME sub-network provides a powerful paradigm for designing and deploying managed services that are decoupled from the hardware infrastructure management.

An example of practical implementation of DIME network is provided in [10], where the authors demonstrate self-repair, auto-scaling to control the response time of a web server in a LAMP architecture.

DIME network related videos can be found at http://www.youtube.com/kawaobjects.
V. CONCLUSION

Since the first appearance of the term autonomic computing coined by Paul Horn, many papers and books have been published to bring self-* (self-configuration, self-monitoring, self-protection, self-optimization and self-repair) properties to distributed systems design.

Different architectures and many implementations have contributed to the progress of the autonomic computing discipline. Object, component, service and agent orientation paradigms have emerged as major paradigms for implementing autonomous distributed systems.

However, as argued in [16], each paradigm addresses only some aspects of distributed computing and none of them are capable of supporting concurrency, distribution, and non-functional aspects such as availability, reliability, performance, scalability and end-to-end transaction security at the system level including the observer and the observed.

Their approaches to distributed computing depend on their implementations on an SPC based on the Turing machine which is constrained by its serial implementation of computations and the prohibition of precise specification, self-reflection, and execution together of a workflow by the SPC implementation of the Turing machine by itself because of Gödel’s theorems as we discussed.

Therefore we assert that the lack of resiliency in distributed systems that affects the availability, reliability, performance and security of end-to-end distributed transactions where changes in one component influences other components, traces back to the fundamental computing model, i.e. the von Neumann Stored Program Control (SPC) implementation of the Turing machine.

In order to overcome these limitations, we introduce DIME computing model. The DIME computing model allows the specification and execution of a recursive composition model where the computing units at any level (observer, network of Turing Machines) specifies and executes the workflow (observed, network of Turing Machines) at the lower level. The specification at a higher level eliminates the self-reflection prohibition of Gödel’s theorems on computational units.

Using the DIME computing model, it is possible to provide to the observer the same resiliency in the infrastructure that implements the control of the observed, improving the ability of the whole system to adapt itself to the dynamic changes of environment.

REFERENCES


