The Turing O-Machine and the DIME Network Architecture: Injecting the Architectural Resiliency into Distributed Computing

Rao Mikkilineni[1], Albert Comparini[1] and Giovanni Morana[2]
[1] Kawa Objects Inc., Los Altos, CA, U.S.A.
{rao, albert}@kawaobjects.com
[2]University of Catania, Catania, Italy
giovanni.morana@dieei.unict.it

Abstract

Turing's o-machine discussed in his PhD thesis can perform all of the usual operations of a Turing machine and in addition, when it is in a certain internal state, can also query an oracle for an answer to a specific question that dictates its further evolution. In his thesis, Turing said 'We shall not go any further into the nature of this oracle apart from saying that it cannot be a machine.' There is a host of literature discussing the role of the oracle in AI, modeling brain, computing, and hyper-computing machines. In this paper, we take a broader view of the oracle machine inspired by the genetic computing model of cellular organisms and the self-organizing fractal theory. We describe a specific software architecture implementation that circumvents the halting and un-decidability problems in a process workflow computation to introduce the architectural resiliency found in cellular organisms into distributed computing machines. A DIME (Distributed Intelligent Computing Element), recently introduced as the building block of the DIME computing model, exploits the concepts from Turing's oracle machine and extends them to implement a recursive managed distributed computing network, which can be viewed as an interconnected group of such specialized oracle machines, referred to as a DIME network. The DIME network architecture provides the architectural resiliency through auto-failover; auto-scaling; live-migration; and end-to-end transaction security assurance in a distributed system. We demonstrate these characteristics using prototypes without the complexity introduced by hypervisors, virtual machines and other layers of ad-hoc management software in today's distributed computing environments.

1 Introduction

"Let the whole outside world consist of a long paper tape. (John von Neumann, 1948)" -- Quote from Turing's Cathedral, by George Dyson, NY. Random House: 2012, p 64

An important implication of Gödel's incompleteness theorem (Gödel, 1931) is that it is not possible to have a finite description with the description itself as the proper part. In other words, it is not possible to read yourself or process yourself as a process. However, as Turing (Turing, 1939) put

it beautifully, “the well-known theorem of Gödel (1931) shows that every system of logic is in a certain sense incomplete, but at the same time it indicates means whereby from a system L of logic a more complete system L_ may be obtained. By repeating the process we get a sequence L, L1 = L_, L2 = L_1 … each more complete than the preceding. A logic Lω may then be constructed in which the provable theorems are the totality of theorems provable with the help of the logics L, L1, L2, … Proceeding in this way we can associate a system of logic with any constructive ordinal. It may be asked whether such a sequence of logics of this kind is complete in the sense that to any problem A there corresponds an ordinal α such that A is solvable by means of the logic Lα.”

This observation along with his introduction of the oracle-machine influenced many theoretical advances including the development of generalized recursion theory that extended the concept of an algorithm (Turing, 2004) (Feferman, 2006). “An o-machine is like a Turing machine (TM) except that the machine is endowed with an additional basic operation of a type that no Turing machine can simulate.” Turing called the new operation the ‘oracle’ and said that it works by ‘some unspecified means’. When the Turing machine is in a certain internal state, it can query the oracle for an answer to a specific question and act accordingly depending on the answer. The o-machine provides a generalization of the Turing machines to explore means to address the impact of Gödel’s incompleteness theorems and problems that are not explicitly computable but are limit computable using relative reducibility and relative computability (Soare, 2009).

On the other hand, it seems as if, biology does not pay attention to Gödel’s incompleteness or un-decidability theorems. As George Dyson (Dyson, 1997, p. 123) points out, a recursive computing model in the genome enables the beautiful unfolding of living organisms with self-configuration, self-monitoring, self-protection, and self-healing properties. "The analog of software in the living world is not a self-reproducing organism, but a self-replicating molecule of DNA. Self-replication and self-reproduction have often been confused. Biological organisms, even single-celled organisms, do not replicate themselves; they host the replication of genetic sequences that assist in reproducing an approximate likeness of themselves. For all but the lowest organisms, there is a lengthy, recursive sequence of nested programs to unfold. An elaborate self-extracting process restores entire directories of compressed genetic programs and reconstructs increasingly complicated levels of hardware on which the operating system runs."

In discussing how a single celled egg gives rise to a complex, multi-billion-celled organism, Sean Carroll (Caroll, 2005) put it elegantly - "I will describe how regulatory DNA is organized into fantastic little devices that integrate information about position in the embryo and the time of development. The output of these devices is ultimately transformed into pieces of anatomy that make up animal forms. This regulatory DNA contains the instructions for building anatomy, and evolutionary changes within this regulatory DNA lead to diversity of form.”

In this paper we argue that the DIME network architecture recently introduced (Mikkilineni, 2011) incorporates a “regulatory” function to exert external influence on a Turing machine while computation is still in progress (and has not halted yet), making it act more like an oracle-machine. A network of such “regulated” Turing machines acts like a managed recursive distributed computing engine with nested monitoring and control functions where each level is managed by the oracle-like machine at a higher level. We conclude that the DIME network architecture circumvents both the halting and un-decidability problems by pushing the knowledge about the context, constraints and control of the computation up the hierarchy which regulates the sequence of hierarchical and temporal events required to implement homeostasis and self-management of the computation. At the root level, the process workflow down the chain defines the stable computing patterns that execute the events to accomplish the system’s purpose and the goals specified at each level. The specification of the system’s purpose at the root level (initial conditions at t = 0) is regulated by an external agent in terms of the context and constraints that define the destiny of the process flow. This architecture, resembling the self-organizing fractal structure (Kurakin, 2011), (Kurakin, 2007) is suited to address some of the concerns currently afflicting distributed computing systems such as concurrency, mobility and

synchronization. Further research is currently in progress to provide a way to implement features of π-calculus (Milner, 1999), (Goyal & Mikkilineni, 2012) including mobility using the DIME network architecture.

The goals of the distributed system determine the resource requirements and computational process definition of individual service components based on their priorities, workload characteristics and latency constraints. The overall system resiliency, efficiency and scalability depend upon the individual service component workload and latency characteristics of their interconnections that in turn depend on the placement of these components (configuration) and available resources. The resiliency (fault, configuration, accounting, performance and security often denoted by FCAPS) is measured with respect to a service's tolerance to faults, fluctuations in contention for resources, performance fluctuations, security threats and changing system-wide priorities. Efficiency depicts the optimal resource utilization. Scaling addresses end-to-end resource provisioning and management with respect to increasing the number of computing elements required to meet service needs.

The traditional resource (CPU, memory, network bandwidth, storage capacity, and throughput) management strategies limit the degree of resiliency, efficiency and scaling of distributed systems (where different service components are served by different resource containers such as a computing device with local resource management provided by a local operating system) in three different ways:

1. The operating systems that manage local resources of the containers that host the service components not only do not have the visibility and control of system-wide service component resource requirement fluctuations at run-time, but often, they also do not have the capability to effectively resolve contention for resources among the many service components they serve. In order to compensate for this, various resource management services have been added to monitor and control the overall system resiliency, efficiency, and scaling, at the expense of increased complexity. In self-managing systems, increased complexity often results in phase transitions leading to simpler architectures (Kurakin, 2007), (Kurakin, 2011).

2. The CAP theorem introduced by Eric Brewer (Brewer, 2000) which explores the tradeoffs between consistency, availability and partition tolerance, concludes that a replicated service can possess just two of the three. Here, availability assures that every request received by a non-failing node must result in a response and partition tolerance is defined by the criterion that the network will be allowed to arbitrarily lose many messages sent from one node to another. Every node receiving a request from a client must respond, even though arbitrary messages that are sent may be lost. The CAP theorem constrains the placement of the services components and hence affects the resiliency, efficiency and scaling of the distributed system transactions. Self-managing systems, however, institute system-wide policies to optimize the trade-off with monitoring and control of the system configurations in real-time using signaling and parallelism (Mikkilineni, 2011).

3. The end-to-end transaction security depends on a host of security strategies deployed by the resource management systems managing various containers that host the services, without the awareness of system-wide service component requirements which are distributed across multiple containers and geographies. Self-managing systems on the other hand, implement effective system-wide monitoring and control to propagate required information to assure end-to-end transaction security using signaling mechanisms.

In short, system-wide resilient resource allocation and optimization (which should be treated as a management process of a meta-stable near-equilibrium state) is not possible without end-to-end service FCAPS management that matches system goals and priorities to resource allocations of distributed service components both at service initiation and during service execution stages. The distributed system development challenges and the shortcomings of current approaches using object, component, and service agent orientation are summarized in (Braubach & Pokahr, 2011) and some

new paradigms are introduced such as active component paradigm to address the shortcomings. These authors identify the importance of FCAPS management mentioned above as non-functional properties like robustness and scalability. According to Braubach and Pokahr (Braubach & Pokahr, 2011) "Non-functional characteristics are particularly demanding challenges because they are often cross-cutting concerns affecting various components of a system. Hence, they cannot be built into one central place, but abilities are needed to configure a system according to non-functional criteria." They propose an active component paradigm that "brings together agents, services and components in order to build a world view that is able to naturally map existing distributed system classes into a unified conceptual representation."

While we agree with the need for a new approach, we follow a different route to argue that the recently introduced DIME computing model simulates a specialized oracle machine and the resulting DIME network architecture (we denote it by c-DNA for computing DNA to distinguish it from biological DNA) provides system-wide regulation of the distributed computing system using a network of such machines. This approach brings the architectural resiliency found in cellular organisms to design a new class of self-managing distributed systems.

In section II, we discuss a specialized oracle inside the distributed intelligent computing element (DIME) addressing FCAPS of the computing element. We then propose a DIME network as an extension of the o-machine concept to a distributed computing network with nested regulatory function. In section III, we discuss the resulting managed recursive distributed computing engine nature of the DIME network where each level regulates the process flow in a lower level of the distributed computing network to assure that the system-wide goals are accomplished within the constraints specified. Regulation is carried out by monitoring and modifying, according to certain policies, the individual component goals at the node level, sub-network level and at the network level. In section IV, we present some observations based on our implementation of c-DNA and conclude with suggestions for future research.

2 The Oracle Machine Like Character of DIME

In its simplest form a DIME consists of a computing element called MICE (Managed Intelligent Computing Element); a regulator (or a policy manager determining the FCAPS behavior called an FCAPS manager in our previous work) designed to manage the life-process of the MICE; and two parallel communication channels.

The MICEs are interconnected using a computing (data) channel to communicate and thus execute a networked computing workflow; and the regulators are networked using a signaling channel which overlays the computing channel executing a parallel management workflow (Mikkilineni, 2011). The regulator can be loaded with the specification and the run-time behavior of the computation that will be executed by the MICE under its control. Thus a computation consists of two parts:

1. A regulation part that specifies the resources required, the purpose, run-time behavior and control function (the meta-model of computation) and
2. An execution part that specifies the execution of the computation mixed with the communication with the oracle to accomplish the given purpose.

The signaling channel overlaying the data channel in the c-DNA enables a group of DIMEs to be interconnected to form a managed DIME computing network or just DIME network.

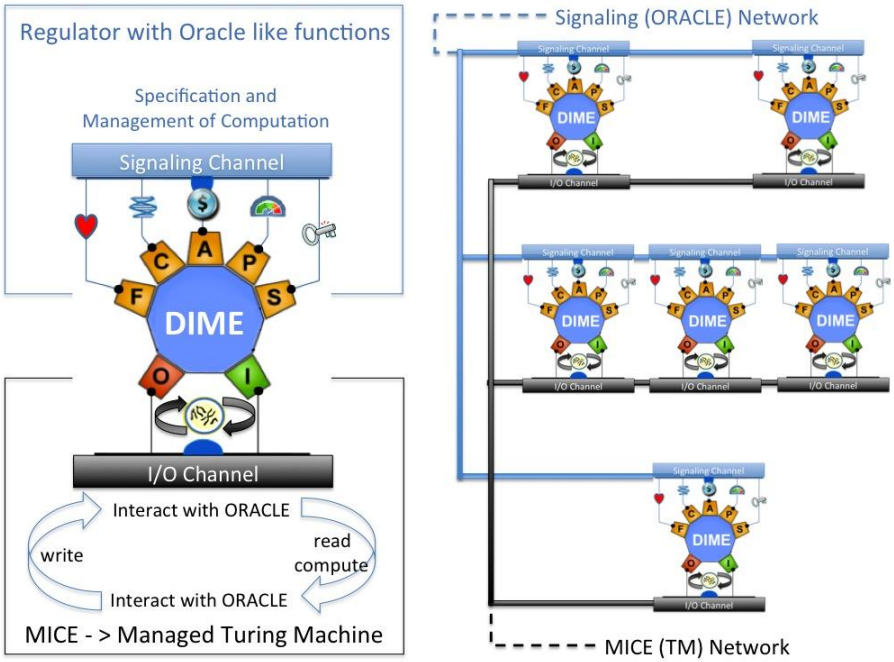
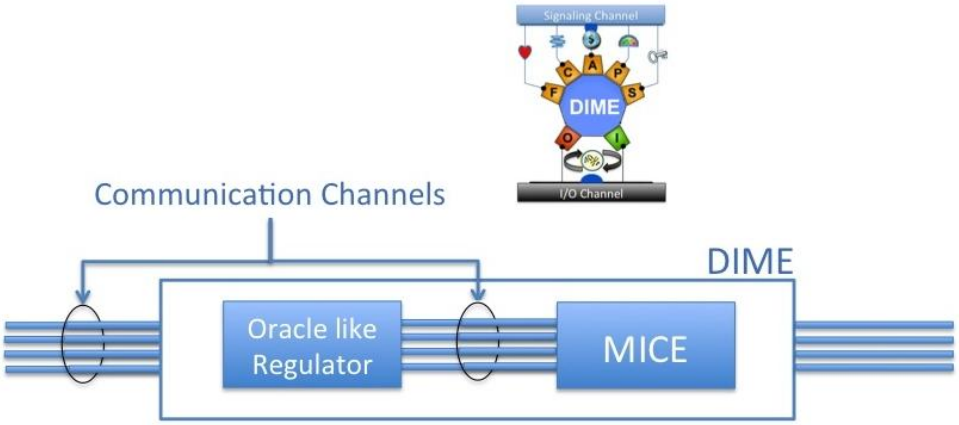


Figure 1 DIME acting like Oracle Machine.

The c-DNA thus enables a network of DIMEs by integrating computing and communication abstractions in a single computing unit. It enables the creation of a network of MICEs executing a distributed computation and a network of managers that provide regulation of the distributed computing events. This allows the composition of hierarchical recursive (each node can itself be a sub-network) managed process implementation. A workflow is implemented as a set of tasks, arranged or organized in a directed acyclic graph (DAG) and executed by a managed network of DIMEs. These tasks, depending on the computation process requirements are programmed and executed as loadable modules in each DIME. The distributed software components along with associated profiles defining their use and management constraints are executed by DIMEs. The profiles are used as blueprints to setup, execute and control the down-stream DAG at each node based on global and local policies that depend on system-wide priorities, process resource utilization fluctuations and communication latency constraints.

Figure 1 shows the anatomy of a DIME and also a network enabled DIME. The information in the regulator about the context and control of the computation in progress in the MICE allows run-time monitoring and control of the computational workflow being performed by the MICE network. The MICE network is the current von Neumann computing network that can be modeled by a Universal Turing Machine. The I/O redirection is under the influence of the FCAPS managers during run time providing the ability to reconfigure the MICE network at run time. The MICE and the regulator exchange policy related information while the computation is in progress to influence each other's behavior and system-wide behavior to implement auto-failover, auto-scaling, live-migration, end-to-end security assurance etc., which improve the architectural resiliency of the system.

Goyal, in addition to showing that the DIME network implements the mobility feature of π-calculus (Goyal & Mikkilineni, 2012) also has developed a generic structure model for the DIME network using the π-calculus recursive operation (Goyal, 2012). We show his model in figure 2.



$$D^{-} \mid (r_i \mid \{c_i\} \mid \{m_i\}) \mid D^{+}$$

Figure 2 A generic DIME where the MICE is a Turing node.

The Regulator possesses a full set of capabilities to manage and control the execution in MICE, generate new DIME nodes and manage communication channels. The management and execution control capabilities, for example, include FCAPS management, capability to provide environmental information, and signaling *termination* or *completion*. *Termination* causes immediate (or halt (or abnormal exit)) of all activities while *completion* signals to the recipient MICE to continue till execution halts normally; additionally, upon post-completion signal receipt, the MICE does not accept any more input.

Generation of a new DIME node accommodates passing of a *Primary Information Packet* (PIP), where the PIP consists of what is traditionally referred to as configuration information but can also include activity modification information. Thus, it is possible to create mutations, clones (identical) or distinct DIME nodes. Each dime node, except the start-up node, when created, by default, is provided with a set of communication channels with its direct ancestors and, thus, any node can also communicate with its direct descendants; The start-up DIME node only consists of the Regulator, r_0, with access to a PIP; thus, with reference to Figure 2, the start-up DIME node consists only of the Regulator. All DIME nodes, except the start-up DIME node, consist of the Regulator, a set of communication channels and MICE. At time of creation a node does not have default direct channels with aunts or n^{th} cousins where n represents natural numbers i.e., n>0). A Regulator can create communication channels among any set of its descendant DIMEs and, thus, creates a network of interconnected nodes. The Regulator can also create or remove communication channels between it and its directly managed MICE or DIMEs; there must be at least one communication channel between two interconnected nodes or components. We distinguish between two types of communication channels – control and data. This use of communication channels is independent of their implementation. During execution, a regulator may pass, as just stated, environmental and control information to its descendants using *Auxiliary Information Packets* (AIP). The PIP is provided at node creation while an AIP is used during execution and lends dynamicity to DIMEs viz., their evolution during operation (for example, akin to "aging" in nature); both PIP and AIP may contain information used by a node to create descendants except that an AIP can dynamically alter the characteristics of the created descendants.

Below we provide some formalism to the DIME network; some notational liberties have been taken. In traditional procedural languages, recursion is implemented by suspending the current iteration while the next iteration executes, while in π-calculus the recursive iterations operate concurrently.

Let C, $\mathcal{D}$, $\mathcal{M}$, $\mathcal{R}$ represent a set of communication channels, DIME, Regulator and MICE nodes respectively.

$d_i = (r_i | \{c_i\} | \{m_i\})$

where a Dime node, d_i, is a set of concurrent processes $r_i \in \mathcal{R}$, $c_i \in C$ and $m_i \in \mathcal{M}$, $\{c_i\}$and $\{m_i\}$ represents a set of channels and Mice; the two set of communication channels of Figure 2 are together represented by the set $\{c_i\}$

$!\mathcal{D} = <r_0> [\ | \mathcal{D} | !\mathcal{D}]$

where '!' is the π-calculus recursion operator, '|' represents concurrency, r_0 represents the initial/root Regulator (at start-up); [⋯] represents option, and {⋯} represents a set.

Thus, from the above we know that a DIME network consists of an initial (start-up) Regulator (the root regulator, r_0) that may be connected through a set of communication channels and operate concurrently with a DIME network . We can visualize the DIME network from some node, d_i , created in the i^{th} iteration as:

$\mathcal{D}^- | (r_i | \{c_i\} | \{m_i\}) | \mathcal{D}^+$ where $\mathcal{D}^-$ represent the ancestors and $\mathcal{D}^+$ the descendants.

3 The Recursive Distributed Computing Engine Nature of c-DNA and Self-Regulation of Process Flow

As discussed above, the DIME network consists of a set of DIMEs endowed with communication channels among the MICEs and among the regulators. The DIME network thus provides a regulatory (or signaling) network overlay over the computing network. Figure 3 shows a simple DIME network. Each configuration in the evolution of the network represents a metastable state with fluctuations (caused by the indeterminism of interaction with the environment) around equilibrium. When the fluctuations exceed certain thresholds, policy management provides self-reconfiguration to create a new metastable equilibrium. Such a hierarchy resembles a single cell unfolding into a complex web of orchestrated workflows that regulate life described by Sean Carroll.

The DIME network architecture (Mikkilineni, 2011) consists of four components:

1. A DIME node which encapsulates the von Neumann computing element with self-management of FCAPS,
2. Signaling capability that enables intra-DIME and Inter-DIME communication and control,
3. 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
4. An infrastructure that assures DIME network management using the signaling network overlay over the computing workflow.

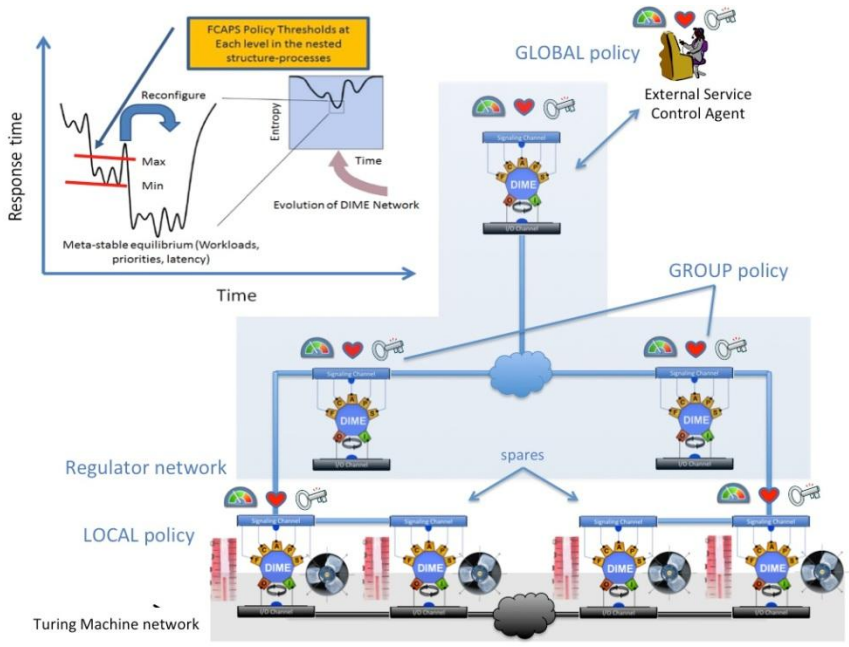


Figure 3 A DIME network where the Turing Machine network is regulated by the Regulator network monitoring the heartbeat, security and other parameters. The metastable configurations and transitions are depicted in the graph.

The self-management and task execution (using the DIME component MICE) are performed in parallel using the stored program control computing devices. 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 DIME FCAPS management allows modeling and representing 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) to form a group that monitors and controls itself while facilitating the implementation of monitoring and control of the observed (environment that is external and influences the observer). Parallelism allows dynamic information flow both in the signaling channel and the external I/O channels of the Turing computing nodes represented by MICE.

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

1. Each MICE is controlled by the FCAPS policies set in each DIME and also those communicated by the regulator network (self-identity and self-management).
2. All reads and writes are dynamically configurable based on the FCAPS policies (system-wide interaction).
3. Each node itself can be a sub-network of DIMEs with goals set by the sub-network policies (hierarchical composition and nested scale-invariant structure- processes (Kurakin, 2007) (Kurakin, 2011))
4. The signaling abstractions (addressing, alerting, supervision and mediation) allow dynamic connection management to reconfigure the DIME network thus changing the policies and behavior at run-time.

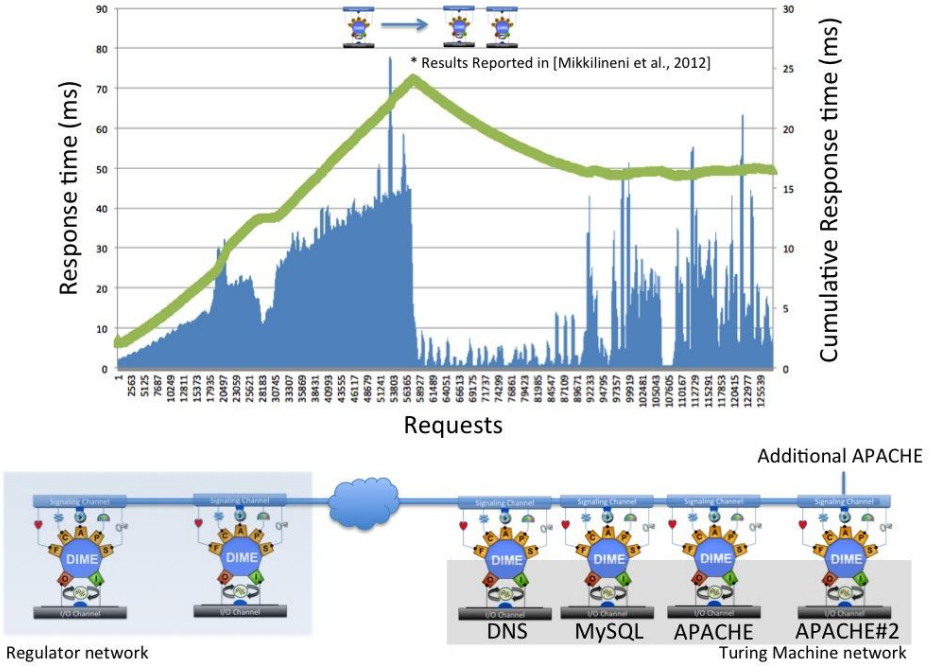


Figure 4 The c-DNA implementing LAMP services with auto-scaling. The response time is monitored periodically to add or remove additional DIMEs to maintain desired service levels.

The DIME network infrastructure allows the instantiation of each DIME, loading of the FCAPS manager with regulation executable, loading of the MICE with the computational workflow executable and starting the computation. It also allows regulation at run-time with the signaling overlay network that allows sub-network and network-level FCAPS managers to implement policies at run-time. Figure 3 depicts a simple workflow controlling a set of fans based on the monitored value of the temperature in a distributed environment. The DIME infrastructure management and the domain workflow management are shown in the same figure.

Currently, the c-DNA has been implemented in two instances (Mikkilineni & Seyler, 2011), (Mikkilineni, Morana, & Seyler, 2012), (Mikkilineni, Morana, Zito, & Di Sano, 2012), (Morana & Mikkilineni, 2011):

1. Using the DIME computing model to provide FCAPS management to a Linux process. This approach allows any Linux executable to be endowed with self-management and signaling capability thus allowing self-repair, auto-scaling, dynamic performance monitoring and management, and end-to-end transaction FCAPS management in a distributed system.

2. A native operating system to run in the next generation multi-core and many-core processor based computing devices to convert each core into a DME and implement managed workflows in a DIME network spanning across multiple computing devices and geographies with network-wide policies based on business priorities, workload fluctuations and latency constraints.

Figures 4 shows the DIME implementation of Linux processes virtualization to implement a distributed Linux, Apache, MySQL, PHP platform with auto-scaling, auto-failover, performance optimization based on response time monitoring and end to end transaction security. In particular, the figure shows the c-DNA auto-scaling ability: when the response time of a web server hosted in APACHE exceeds a given threshold, the c-DNA creates an additional copy of it and configures the DNS server to perform load balancing. More details can be found in (Mikkilineni, Morana, Zito, & Di Sano, 2012). The DIME network architecture allowed Linux process virtualization without using Hypervisors or Virtual Machines. A video showing the auto-failover using DIME network architecture by converting a Linux process into a DIME is available at http://www.youtube.com/kawaobjects .

A demo of the Parallax OS, implementing DIME network architecture in multi-core servers, is available at http://youtu.be/K0AxJPaA_RI . This last video shows state-aware fail-over, auto-scaling, and dynamic I/O reconfiguration.

4 Conclusion

As von Neumann (Aspray & Burks, 1989) put it "It is a theorem of Gödel that the description of an object is one class type higher than the object." Turing's o-machine was designed to provide information that is not available in the computing algorithm executed by the TM. We take a similar approach in the DIME computing model by separating the execution and management of the computation. By implementing parallel management and choosing well specified managed DAGs defining system-wide goals, resources and constraints, the DIME computing model circumvents the thorny problems of decidability and halting by pushing them up the hierarchy to the root level where initial conditions (t = 0) and constraints define the process flow (which are often under the influence of an external agent). Using the same FCAPS management structure at the node, sub-network and at the network level (scale-invariance), the DIME network architecture provides a fractal (recursive) composition model to implement managed DAGs with both hierarchical and temporal event networks.

The DIME computing model supports the genetic transactions of *replication, repair, recombination and reconfiguration* (Stanier & Moore, 2006), (Mikkilineni, 2011). The DIME computing model, we believe is an embodiment of the inspiration from the oracle-machine that Turing envisioned and goes a step further by integrating communication and computing to enable managed distributed computing to implement system-wide homeostasis. Each Turing machine (functioning as the MICE or the parallel FCAPS managers) is implemented using the von Neumann serial computing model, but they communicate using shared memory and operate in parallel. The introduction of signaling (using different communication channels such as shared memory, PCI-Express or socket) allows a network-wide coordination and collaboration of the managers to orchestrate the global policies to implement the managed workflow using the MICE network. The oracle "network effect" provides a synergy that is greater than the sum of its parts by effectively using global knowledge. It is important to note that the oracle behavior in a DIME consists of the context sensitive wisdom of best practices and are not constrained by Turing machine implementations of algorithmic behavior in the spirit of Turing's thesis.

While there is a resurgence in the discussion of computing models (van Leeuwen & Wiedermann, 2000), even a call for a Kuhnian paradigm shift to embrace new computing models (Wegner & Goldin, 2003), (Eberbach & Wegner, 2003), (Wegner & Eberbach, 2004), (Aho, 2010), (Denning, 2011), (Goldin & Wegner, 2008) and rebuttals (Cockshott & Michaelson, 2007), the DIME computing model takes a practical approach by implementing the oracle-like behavior in a Turing machine. It also exploits the parallelism and the integration of computing with communication

networks. Above all, it exploits the signaling abstractions to regulate the computing behavior based on its context that is the norm, not an exception, in living organisms.

The introduction of a signaling network overlay over computing network adds a new dimension in distributed computing by incorporating the architectural resilience of cellular organisms into computing machines. It allows specification of equilibrium patterns in computation process flows, and monitor and control exceptions system-wide. It allows contention resolution based on system-wide view and eliminates race conditions and other common issues found in current ad-hoc distributed computing practices without a concrete computing model (such as a TM or an o-machine) serving as a solid foundation. In systems with strong dynamic coupling between various elements of the system, where each change in one element continually influences the other element's direction of change, signaling in the computation model helps implement system-wide coordination and control based on global priorities, workload fluctuations and latency constraints. Signaling and the separation of specification and execution of a computation provide a mechanism to introduce self-replication, self-repair, recombination and reconfiguration of computing network at run-time.

The DIME network architecture comes close to what von Neumann was searching for in his Hixon lectures in 1948 (Aspray & Burks, 1989) "The basic principle of dealing with malfunctions in nature is to make their effect as unimportant as possible and to apply correctives, if they are necessary at all, at leisure. In our dealings with artificial automata, on the other hand, we require an immediate diagnosis. Therefore, we are trying to arrange the automata in such a manner that errors will become as conspicuous as possible, and intervention and correction follow immediately." Comparing the computing machines and living organisms, he points out that the computing machines are not as fault tolerant as the living organisms. He goes on to say "It's 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 run for a millisecond." The evolutionary history and the genetic transactions supported by the information flow in the DIME network architecture provide the infrastructure for autonomic distributed computing system design.

Our prototypes demonstrate that c-DNA *enables 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*. It has also not escaped our attention that the theoretical landscape of computing models is filled with controversy. It is important to point out that we do not claim that DIME is precisely the o-machine that Turing proposed or that the Universal Turing machine (UTM) can or cannot do what a DIME network does. While communicating Turing machines are modeled by an UTM (Penrose, 1989), can the managed Turing machine networks also be modeled by the UTM? We only point out that the DIME is inspired by the oracle machine and implements at least some aspects of the architectural resiliency of cellular organisms in distributed computing infrastructure by introducing parallel management of both the computing elements and networks. While its feasibility has been demonstrated (Mikkilineni, Morana, & Seyler, 2012), the DIME network architecture is still in its infancy and presents an opportunity on the eve of Turing's centenary celebration to further investigate its usefulness and theoretical soundness. Only time will tell if c-DNA will be useful in mission critical environments.

Acknowledgement

The authors wish to express their gratitude to Daniele Zito and Marco Di Sano for their enthusiasm and effort in implementing the self-regulating Linux, Apache, MySQL, and PHP (LAMP) cloud using c-DNA thus eliminating the complexity of Hypervisors and Virtual Machines. The authors also are grateful to Ian Seyler for implementing the native operating system converting each core in a multi-core processor into a DIME to demonstrate system-wide resiliency in workflow

execution spanning across multiple many-core processors, servers, and geographies. One of the authors (Rao Mikkilineni) is especially grateful to Pankaj Goyal for sharing his research results before publication and many valuable conversations.

References

Aho, A. (2010). Computation and Computational Thinking. *Ubiquity Symposium (http://ubiquity.acm.org/symposia.cfm).* ACM.

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

Braubach, L., & Pokahr, A. (2011). Addressing Challenges of Distributed Systems Using Active Components. In *Intelligent Distributed Computing V*. New York, NY: Springer.

Brewer, E. A. (2000). Towards robust distributed systems. *Proceedings of ACM symosium principles of distributed computing (PODC'00)* (p. 7). New York: ACM Press.

Caroll, S. B. (2005). *The New Science of Evo Devo - Endless Forms Most Beautiful.* New York, NY: W. W. Norton & Co.

Cockshott, P., & Michaelson, G. (2007). Are There New Models of Computation? Reply to Wegner and Eberbach. *Computer Journal, 5*(2), 232-247.

Denning, P. (2011). What Have We Said About Computation? Closing Statement. *http://ubiquity.acm.org/symposia.cfm.* ACM.

Dyson, G. B. (1997). *Darwin among the Machines, the evolution of global intelligence.* Reading MA: Addison Wesley.

Eberbach, E., & Wegner, P. (2003). Beyond Turing Machines. *The Bulletin of the European Association for Theoretical Computer Science (EATCS Bulletin), 81*(10), 279-304.

Feferman, S. (2006). Turing's Thesis. *Notices of the AMS, 53*(10), 2.

Gödel, K. (1931). *Monatshefte für Mathematic und Physik, 38*, 173-198.

Goldin, D., & Wegner, P. (2008). The Interactive Nature of Computing: Refuting the Strong Church-Turing Thesis. *Minds & Machines, 18*, 17-38.

Goyal, P. (2012, June). A Recursive Computing Model for DIME Network Architecture using π-calculus. Private Communication.

Goyal, P., & Mikkilineni, R. (2012). Implementing Managed Loosely-coupled Distributed Business Processes: A New Approach using DIME Networks. *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 21st IEEE International Conference.* Toulouse: IEEE.

Kurakin, A. (2007). Retrieved from The universal principles of self-organization and the unity of Nature and knowledge: http://www.alexeikurakin.org/text/thesoft.pdf

Kurakin, A. (2011). *Theoretical Biology and Medical Modeling.* Retrieved from http://www.tbiomed.com/content/8/1/4

Mikkilineni, R. (2011). *Designing a New Class of Distributed Systems.* New York, NY: Springer.

Mikkilineni, R., & Seyler, I. (2011). A New Operating System for Scalable, Distributed, and Parallel Computing. *Parallel and Distributed Processing Workshops and Ph,d Forum (IPDPSW), 2011 IEEE International Symposium on*, (pp. 976-983).

Mikkilineni, R., Morana, G., & Seyler, I. (2012). Implementing Distributed, Self-managing Computing Services Infrastructure using a Scalable, Parallel and Network-centric Computing Model. In M. Villari, C. I. Brandic, & F. Tusa, *Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice* (pp. 57-78). IGI Global.

Mikkilineni, R., Morana, G., Zito, D., & Di Sano, M. (2012). Service Virtualization Using a Non-von Neumann Parallel, Distributed, and Scalable Computing Model. *Journal of Computer Networks and Communications, 2012*.

Milner, R. (1999). *Communicating and mobile systems: The pi-calculus.* Cambridge, UK: Cambridge University Press.

Morana, G., & Mikkilineni, R. (2011). Scaling and Self-repair of Linux Based Services Using a Novel Distributed Computing Model Exploiting Parallelism. *20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)* (pp. 98-103). IEEE.

Penrose, R. (1989). *The Emperor's New Mind: Concerning Computers, Minds, And The Laws of Physics.* Oxford, UK: Oxford University Press.

Soare, R. (2009). Turing oracle machines, online computing, and three displacements in computability theory. *Annals of Pure and Applied Logic, 160*(3), 368-399.

Stanier, P., & Moore, G. (2006). In P. Ferretti, A. Copp, C. Tickle, & G. (. Moore, *Embryos, Genes and Birth Defects. (2nd Edition)* (p. 5). London: John Wiley & Sons.

Turing, A. M. (1939). Systems of logic defined by ordinals. *Proc. Lond. Math. Soc., Ser. 2, 45*, 161-228.

Turing, A. M. (2004). In B. J. Copeland (Ed.), *The Essential Turing.* Oxford, UK: Oxford University Press.

van Leeuwen, J., & Wiedermann, J. (2000). The Turing machine paradigm in contemporary computing. In B. Enquist, & W. Schmidt, *Mathematics Unlimited—2001 and Beyond.LNCS.* New York, NY: Springer-Verlag.

Wegner, P., & Eberbach, E. (2004). New Models of Computation. *The Computer Journal, 47*(1), 4-9.

Wegner, P., & Goldin, D. (2003). Computation beyond Turing Machines: Seeking appropriate methods to model computing and human thought. *Communications of the ACM, 46*(4), 100.

Dr. Rao Mikkilineni received his PhD from University of California, San Diego in 1972 working under the guidance of prof. Walter Kohn (Nobel Laureate 1998). He later worked as a research associate at the University of Paris, Orsay, Courant Institute of Mathematical Sciences, New York and Columbia University, New York.

He is currently the Founder and CTO of Kawa Objects Inc., California, a Silicon Valley startup developing next generation computing infrastructure. His past experience includes working at AT&T Bell Labs, Bellcore, U S West, several startups and more recently at Hitachi Data Systems.

Dr. Giovanni Morana received his PhD from University of Catania, Italy and is currently at the University of Catania. He has contributed many papers in grid computing, cloud computing and distributed systems management.

Dr. Albert Comparini received his PhD from Yale University and is currently with Kawa Objects Inc. His past experience includes working at AT&T Bell Labs, Siemens and Infinion.

Dr. Mikkilineni and **Dr. Giovanni Morana** co-chair the 2nd track on Convergence of Distributed Clouds, Grids and their Management in IEEE International WETICE Conference