Cloud computing based on agent technology, super-recursive algorithms and DNA

Mark Burgin

UCLA,
Los Angeles, USA
Email: mburgin@math.ucla.edu

Rao Mikkilineni*

C3DNA,
Cupertino, USA
Email: rao@c3dna.com
*Corresponding author

Abstract: Agents and agent systems are becoming more and more important in the development of a variety of fields such as ubiquitous computing, ambient intelligence, autonomous computing, data analytics, machine learning, intelligent systems and intelligent robotics. In this paper, we examine interactions of theoretical computer science with computer and network technologies analysing how agent technology emerged, matured and progressed in mathematical models of computation. We demonstrate how these models are used in the novel distributed intelligent managed element (DIME) network architecture (DNA), which extends the conventional computational model of information processing networks, allowing improvement of the efficiency and resiliency of computational processes. Two implementations of DNA described in the paper illustrate how the application of agent technology radically improves current cloud computing state of the art. First example demonstrates the live migration of a database from a laptop to a cloud without losing transactions and without using containers or moving virtual machine images. The second example exhibits the implementation of cloud agnostic computing over a network of public and private clouds where live computing process workflows are migrated from one cloud to another without losing transactions. Both these implementations demonstrate the power of scientific thought for dramatically extending the current state of the art of cloud and grid computing practice.

Keywords: cloud computing; agent technology; inductive Turing machine; grid computing; DIME network architecture; intelligent systems; super-recursive algorithms.

Reference to this paper should be made as follows: Burgin, M. and Mikkilineni, R. (2018) 'Cloud computing based on agent technology, super-recursive algorithms and DNA', *Int. J. Grid and Utility Computing*, Vol. 9, No. 2, pp.193–204.

Biographical notes: Mark Burgin is currently working at UCLA. He received his MA and PhD in Mathematics from Moscow State University and Doctor of Science in Logic and Philosophy from the National Academy of Sciences of Ukraine. He was a Professor at the Institute of Education; International Solomon University; Kiev State University, Ukraine; and Director of the Assessment Laboratory at the National Academy of Sciences of Ukraine. He is a member of the New York Academy of Sciences and an Honorary Professor of the Aerospace Academy of Ukraine. He has published many papers and books in mathematics, computer science and artificial intelligence.

Rao Mikkilineni received his PhD from University of California, San Diego working under the guidance of Professor 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 co-Founder and Chief Scientist at C3DNA, a Silicon Valley company developing cognitive distributed computing infrastructure. His past experience includes working at AT&T Bell Labs, Bellcore, US West, several start-ups and more recently Hitachi Data Systems. He currently co-chairs the IEEE conference track on the convergence of distributed clouds, grids and their management in WETICE.

This paper is a revised and expanded version of paper entitled 'Agent technology, super-recursive algorithms and DNA as a tool for distributed clouds and grids' presented at the '25th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises', Paris, France, June 2016.

1 Introduction

In a renowned address, Freeman Dyson described missed opportunities in physics and related mathematics due to severed ties between these two areas (Dyson, 1972). Similar situations often emerge in relations between science and technology. Here we describe some missed opportunities in information processing technology demonstrating how these opportunities now are bringing new technological advances due to utilisation of software agents, advanced results from theoretical computer science and a new computing model using a novel distributed intelligent managed element (DIME) network architecture (DNA) (Mikkilineni et al., 2012).

It is possible to divide the history of computing into three periods (Luck et al., 2005):

1 Computation as calculation, or operations undertaken on numbers.

2 Computation as information transformation, or operations on multimedia, such as text, audio or video data.

3 Interactive computation, or computation as interaction.

The first period can be positioned from the time of Charles Babbage in the nineteenth century until the mid-1960s. The second period can be placed from 1960s until 1990s. In 1990s, the third period started when with the growth of the Internet and the World Wide Web, computing has become an inherently social activity, rather than an isolated process, with new ways of conceiving, designing, developing and managing computing systems. This new environment demanded new approaches to information processing and the answer was found in agent technology (Bradshaw, 1997; Dinverno and Luck, 2001; Luck et al., 2005). In this context, an artificial agent is a software system capable of flexible and autonomous action in a dynamic environment, usually an environment containing other agents, while acting on behalf of another system.

Many believe that in computation, computer science only follows achievements of computer technology. However, there is a remarkable example: the great mathematician John von Neumann introduced the so-called, von Neumann architecture of computers taking as a blueprint the structure of a universal Turing machine – a mathematical model of computation created by Alan Turing years before the first electronic computer was built. As we know, almost everything – basic elements, data structures, programming languages, etc. – changes very fast in computer technology but von Neumann architecture existed for several decades as the prevalent architecture for computers.

This is only one example demonstrating that theoretical computer science is a cornerstone for computer science as a whole and computer technology as its practical application (Savage et al., 2001). Theoretical computer science based on mathematical structures and procedures "underlies many aspects of the construction, explanation, and understanding of computers" (Funding a Revolution, 1999).

In this paper, we analyse when and how mathematical models in computer science have started representing agent technology. This analysis allows us to demonstrate (in Section 2) that agents emerged in these models in 1980s, while according to the opinions of experts, agent technology started only in 1990s. Once more science has surpassed technology. In Section 3, we show how agent technology can provide constructive tools for utilisation of such a popular theoretical model of computation as Oracle Turing machine. In Section 4, we demonstrate how the first mathematical model employing agent technology (called an inductive Turing machine) provides for further development of network architecture serving as a theoretical foundation for the innovative distributed intelligent managed element (DIME) network architecture (DNA), which represents a phase transition in the development of global networks, their software and functioning (Mikkilineni et al., 2012). Modelling a network as an extremely large dynamic memory, we see that the DNA agent organises connection in this network in the same way as one inductive Turing machine builds connections in the memory of another one inductive Turing machine.

It is necessary to remark that in 1975, the Actors model related to agent technology was introduced (Hewitt et al., 1973). However, this model is essentially more general than the agent-oriented model because, as Milner wrote (Milner, 1993), Hewitt declared that a value, an operator on values, and a process should all be Actors. This demonstrates that agents are a very special but essentially important case of actors. It is possible to compare relation between actors and agents with the relation between a function and a computable, e.g., recursive, function.

Thus, the concept of a software/hardware agent was not formalised and utilised in technology although John Doyle (1983) and Marvin Minsky (1986) used the word "agent" in an informal setting.

2 Agent constructed hierarchies of Turing machines

Inductive Turing machine, as a rigorous mathematical model of algorithms and computation, was introduced by Mark Burgin in 1983 as an abstract computational device more powerful than Turing machine (Burgin, 1983). It was the first class of automata with this property. Later Burgin proved that inductive Turing machines have many other advantages (Burgin, 1997, 2005).

First, as it has been demonstrated, inductive Turing machines, as an innovative model of computations, algorithms and information processing systems, can compute much more than Turing machines, i.e., they are super-recursive algorithms while Turing machines are only recursive algorithms. In particular, inductive Turing machines can solve problems unmanageable by Turing machines providing means for decreasing complexity of computations and decision-making

(Burgin, 2005). Consequently, in comparison with Turing machines and other recursive algorithms, such as partial recursive functions or Minsky machines, inductive Turing machines represent the next step in the development of computer science as well as in the advancement of network and computational technology. In addition, inductive Turing machines can essentially decrease complexity of various systems making easier their exploration, modelling and construction (Burgin, 2016a).

Second, inductive Turing machines supply algorithms, networks, and information processing systems more adequate than recursive algorithms and automata models of computations, As a result, inductive Turing machines have found diverse applications in algorithmic information theory and complexity studies (Burgin, 2010), software testing (Burgin and Debnath, 2009; Burgin et al., 2009), high performance computing (Burgin, 1999), machine learning (Burgin and Klinger, 2004), software engineering (Burgin and Debnath, 2004, 2005), computer networks (Burgin, 2006; Burgin and Gupta, 2012), evolutionary computations (Burgin and Eberbach, 2009, 2009a, 2012) and in the study of mathematical problem complexity (Calude et al., 2012; Hertel, 2012). For instance, inductive Turing machines can perform all types of machine learning – TxtEx-learning, TxtFin-learning, TxtBC-learning, and TxtEx*-learning with better outcomes than the standard approach to these educational technologies (Beros, 2013). While the standard approach to machine learning models learning processes using different kinds of functions, e.g., limit partial recursive functions (Gold, 1967), inductive Turing machines are automata, which can compute values of the modelling functions and perform other useful operations in comparison with functions, which only describe such operations.

Third, inductive Turing machines also provide efficient tools for algorithmic information theory, which is one of the indispensable areas in information theory and is based on complexity of algorithms and automata (Chaitin, 1977; Burgin, 2010).

Methodological and mathematical analysis of information processing on different levels gives us the following conclusion.

Turing machine formalises the work of an accountant or a human computer. Inductive Turing machine formalises the work of a scientist and functioning of science.

Hence, it is natural that inductive Turing machines can do much more than Turing machines. To understand all these advantages, let us look at the architecture of inductive Turing machines.

The hardware of an inductive Turing machine M consists of three abstract devices:

- The control device A, which is a finite automaton and controls performance of M;
- The operating device H, which can include any number of processors;
- The memory E.

The control device A is a finite automaton. It controls and regulates processes and parameters of the machine M: the state of the whole machine M, the processing of information by the processor H, and the storage of information in the memory E.

The memory E of a general inductive Turing machines consists of cells and is structured by a system of relations that organise memory functioning and provide connections between cells. In particular, input registers, the working memory, and output registers of M are discerned. In a general case, cells may be of different types. Thus, the memory is a network of cells and it is possible to interpret these cells as information processing systems, such as neurons in the brain of a human being or computers in the World Wide Web or abstract computing devices in a grid automaton (Burgin, 2003a).

We will not describe how the processor H works and what the software of an inductive Turing machine is because our goal is to analyse functioning of the memory E as a structured network in the context of the constructive hierarchy of inductive Turing machines (Burgin, 2003). In this section we summarise several basic features of inductive Turing machines:

1 At first, it is necessary to remark that in contrast to many other super recursive models of computation, such as accelerating Turing machines (Shagrir, 2004) or infinite time Turing machines (Hamkins and Lewis, 2000; Welch, 2000; Davies, 2001), inductive Turing machines give their results in finite time, i.e., after finite number of steps.

2 The constructive hierarchy of inductive Turing machines is organised based on the properties of the structured memory. On the first level of the constructive hierarchy, we have inductive Turing machines with recursive memory where the memory E is called recursive if all relations that define its structure are recursive. Here recursive means that it is established by Turing machines or other recursive automata (Burgin, 2005).

3 There are different techniques to organise memory construction. The simplest approach is called the beforehand strategy when given some data, e.g., a description of the structure of E, a Turing machine T builds all connections in the memory E before the machine M starts its computation.

4 According to another methodology called the concurrent strategy, memory construction by the machine T and computations of the machine M go concurrently, while the machine M computes the machine T constructs connections in the memory E.

5 It is also possible to use the mixed strategy when some connections in the memory E are assembled before the machine M starts its computation, while other connections are formed parallel to the computing process of the machine M.

6 These three strategies determine three kinds of a structured memory:

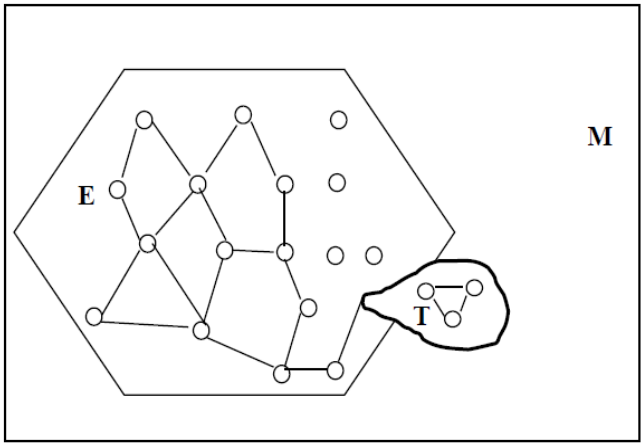
- In the static memory E of the machine M, which is constructed using the beforehand strategy, all connections are constructed before M starts working.
- In the growing memory E of the machine M, which is constructed using the concurrent strategy, connections are constructed while M is working but no connections are deleted.
- In the dynamic memory E of the machine M, which is constructed using the mixed strategy, when it necessary, some connections are constructed and when it necessary, some connections are deleted while M is working.

7 In addition, there are three types of construction agents T:

- A rigid agent T always constructs one and the same memory structure for the machine M.
- An adaptive agent T constructs memory structure for the machine M taking into account the input data.
- A dynamic agent T constructs memory structure for the machine M taking into account the process of computation.

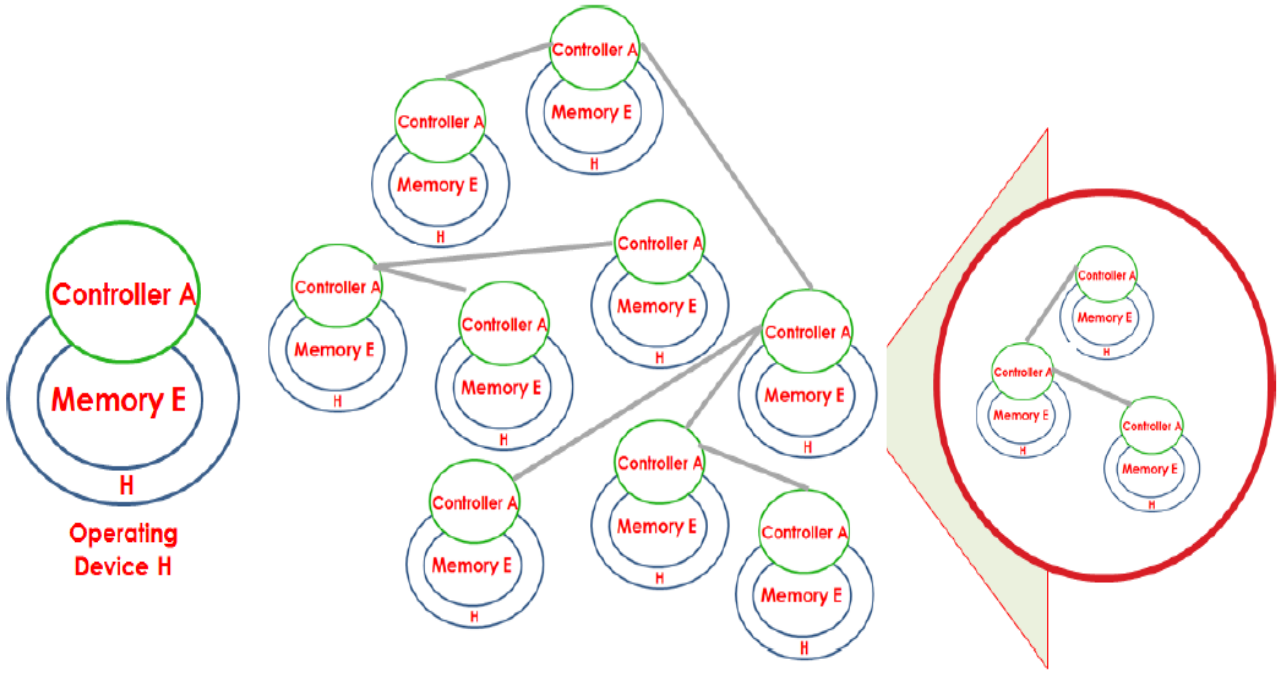
Functioning in the inductive Turing machine, its dynamic agent performs a phase transition with reconfiguration of the structure of the system and architecture transformation bringing in order from chaos (see Figure 1). In a similar way, the agent can build a network of computing devices such as inductive Turing machines or computer clusters (see Figure 2). This is essential when the memory E models a global network such as the World Wide Web or the Grid.

Figure 1 An inductive Turing machine T building connections in the memory E of another inductive Turing machine M



Besides, there is a memory construction schema when the machine (agent) T is separate from the machine M, while another memory construction schema includes the machine T as a part of the machine M. In the first case, the Turing machine T is an autonomous agent for the machine M, while in the second case, T is a controlled agent.

Figure 2 A grid automaton T building a network of inductive Turing machines



(a) Inductive Turing Machine (M)

(b) An agent of an inductive Turing machine building the memory of another Turing machine

It is necessary to remark that when both the agent and the main machine are Turing machines or some recursive automata, such a recursive agent T does not increase the power of the main machine M. However, it is proved that T can essentially amplify the efficiency of the machine M (Burgin, 1999).

Inductive Turing machines with recursive memory are called inductive Turing machines of the first order.

While in inductive Turing machines of the first order, the memory is constructed by Turing machines, it is possible to use inductive Turing machines for memory construction for other inductive Turing machines. This brings us to the concept of inductive Turing machines of higher orders. For instance, in inductive Turing machines of the second order, the memory is constructed by Turing machines of the first order.

In general, we have the following definition.

The memory E is called n-inductive if its structure is constructed by an inductive Turing machine of the order n. The second inductive Turing machine plays the role of an agent for the first inductive Turing machine. Inductive Turing machines with n-inductive memory are called inductive Turing machines of the order $n + 1$. Namely, in inductive Turing machines of order n, the memory is constructed by Turing machines of order $n - 1$.

Thus, if the memory of an inductive Turing machine M is constructed by an inductive Turing machine A, then A is an agent that performs memory structuring for M. Such a super-recursive agent A can essentially advance the power of the machine M because the machine M has higher order than the machine A (Burgin, 2003).

Note that the hierarchy IT of inductive Turing machines induces a hierarchy of agents that function in IT.

At the same time, it is possible to delegate the task of memory building to two or more agents in the form of Turing machines or inductive Turing machines. For instance, given initial data, one of these machines AC computes (designs) a description (schema) of the memory E, while another machine AB constructs connections between cells in E using the schema designed by the machines AC. This approach involves two types of agents:

- Designing agents
- Constructing agents

In a more advanced memory construction schema, memory E of an inductive Turing machine M is constructed by three agents in the form of Turing machines or inductive Turing machines. For instance, the first agent AI collects information, e.g., by data mining, for the schema of the memory E, the second agent AC computes a description (schema) of the memory E, and the third agent AB constructs connections between cells in E. This approach involves three types of agents:

- Mining agents
- Designing agents
- Constructing agents

We see that the schema of inductive Turing machines, which is intrinsically based on agent technology, introduced different types of computational agents before agent technology was usually employed in physical computations and networking.

Note that in some cases, the agents act as oracles in the sense of Burgin (2016), while in other cases, the agents are only assistants and/or facilitators for the main machines

3 Agent technology for Oracle Turing machines

An important mathematical model of computation is Oracle Turing machine. Computer scientists have used Oracle Turing machines for building a sophisticated theory of relative algorithms and computation (cf., for example, Soar, 1987). In this theory, levels of incomputability are constructed and problems are classified according to these levels. Soare (2015) even argues that actually relative computations are in the centre and form the main content of theoretical computer science (compare also Homer and Selman, 2011).

According to the contemporary formalisation, an Oracle Turing machine is a Turing machine enhanced with a tape where the values of the advice function $f(x)$ are stored and with the possibility to access the tape with the advice in constant time and read from it the value of its advice function $f(x)$ also in constant time (Homer and Selman, 2011; Soare, 2015). Thus, in the contemporary representation, an Oracle is a tape with additional information.

However, this model of an Oracle is not constructive because it does not describe how this information in the Oracle is obtained. To build a constructive model, we go beyond an Oracle Turing machine and employ an agent, which by computation or search collects this information for the basic machine or for a more complex information processing device, such as a network or computer cluster, working as an Oracle. For instance, in the theory of relative algorithms and computation, an important type of Oracles contains information whether a given Turing machine halts given definite input or not (Rogers, 1987). As we know in this case, it is possible to use simple inductive Turing machines for bringing this information to Oracles because these machines are able to compute such information (Burgin, 2005). Utilisation of higher-order inductive Turing machines supplies Turing machines and other computing devices with even more powerful Oracles, making the whole schema of Oracle machines, for which Oracle Turing machine is only a particular case, a constructive device.

When the agent that functions as an Oracle collects information for the basic machine by search, it is a *mining agent.* When this agent obtains information for the basic machine by computation, it is a *designing agent.* The hierarchy of computational models determines a hierarchy of mining and designing agents.

There are also rigid agents, which transform only external information using the same program (algorithm), and reflexive agents, which additionally change their own program (algorithm) to achieve better results.

4 DNA as a new application area network organisation

The distributed intelligent managed element (DIME) network architecture (DNA) extends the conventional computational model of information processing networks (IPN), allowing improvement of the efficiency and resiliency of computational processes. This approach is based on organising the process dynamics under the supervision of intelligent agents.

The current network architecture employs three types of functional devices: computers, servers and routers. DNA adds one more type of such devices – service agents, which establish a dynamic structure of the network in the same way as the construction agent in an inductive Turing machine forms the memory structure.

The economics model of an IPN represents it as a graph, nodes of which have three types: supplier (S), consumer (C) and supplier–consumer (SC). Suppliers provide definite resources, consumers use required resources and supplier–consumers use some resources and provide other resources. The goal of service agents is to establish connections between consumers that need some resource and suppliers that provide this resource.

We understand service in comprehensive way as the delivery of some product and consider three forms of the product:

- Physical form where product is a physical object or collection of physical objects
- Informational form where product is an information item or collection of information items
- Process form where product is a process or collection of processes

For instance, a CD of a movie is a product in a physical form, a description of the movie plot is a product in an informational form and demonstration of the movie on the screen of a computer is a product in a process form.

There are also three forms of service connections between consumers and suppliers:

- Building a direct connection, a service agent connects a consumer with a supplier and then the supplier directly provides the needed service (product) to this consumer.
- Forming a product connection, a service agent (1) gets an order from a consumer for the needed product, (2) finds a supplier providing this product, obtains the product from the supplier and then delivers the needed product to the consumer.
- Developing a virtual connection, a service agent connects a consumer not with a supplier but with another system, which connects consumer to the relevant supplier.

Service agents are realised as DIME elements based on the DIME network architecture, which introduces three key modelling constructs to enable process design, execution and its management to improve both expressiveness and efficiency.

4.1 DIME basic process

The first modelling construct is the DIME basic process P, which performs the {read → compute → write} instruction cycle executing an algorithm or its modified version the {interact with external agent → read → compute → interact with external agent → write} instruction cycle. This allows the external agent to influence the further evolution of computation while the computation is still in progress.

DNA employs three types of external agents:

a The DIME agent;

b The human agent;

c The external computing agent.

It is assumed a DIME agent knows the goal and intent of the algorithm (along with the context, constraints, communications and control of the algorithm) the DIME basic processor is executing and has the visibility of available resources and the needs of the basic processor as it executes its tasks. In addition, the DIME agent also has the knowledge about alternate courses of action available to facilitate the evolution of the computation to achieve its goal and realise its intent. Thus, every algorithm is associated with a blueprint (analogous to a genetic specification in biology) that will provide the knowledge required by the DIME agent to manage the process evolution. An external computing agent is any computing node in the network with which the DIME unit interacts. All this makes DIME an intelligent agent in the sense of artificial intelligence (AI).

4.2 DIME agent

The second modelling construct is the DIME agent, which uses the blueprint to configure, instantiate, and manage the DIME basic processors executing the algorithm, and specifies their evolution implementing various policies to assure non-functional requirements such as availability, performance, security, compliance and cost management.

Figure 3 shows the DIME basic processor and its DIME agent, which manages it using the knowledge provided by the blueprint which has:

- Encoded information about the intent,
- Processes that configure the resources to execute the intent,
- The best practice policies and
- Corresponding methods to monitor and manage deviations if they occur during the execution of the intent.

4.3 DIME signalling and control channel overlay

In addition to read/write communication of the DIME basic processor (the data channel), the DIME agents managing different DIME basic processors communicate with each other using a parallel signalling channel. This allows the external DIME agents to influence the computation of any managed DIME basic processor in progress based on the

context and constraints. The external DIME agents themselves are DIMEs in which case, changes in one computing element could influence the evolution of another computing element at run time without halting its Turing machine executing the algorithm. The signalling channel and the network of DIME agents can be programmed to execute a process whose intent itself can be specified in a blueprint. Each DIME basic processor can have its own oracle managing its intent, and groups of managed DIME basic processors can have their own domain managers implementing the domain's intent to execute a process. The management DIME agent specifies, configures and manages the sub-network of DIME units by monitoring and executing policies to optimise the resources while delivering the intent.

Figure 4 shows a DIME network implementing a process using different hardware, functions and an evolving structure to deliver the intent of the process. The DIME network is itself managed by monitoring its own vital signs and executing various fault, configuration, accounting, performance and security (FCAPS) policies to assure availability, performance and security. At each level in the hierarchy, a domain specific task or workflow is executed to implement a distributed process with a specific intent.

Figure 3 The DIME computing model with sensors and actuators connecting the DIME basic processor with the DIME agents

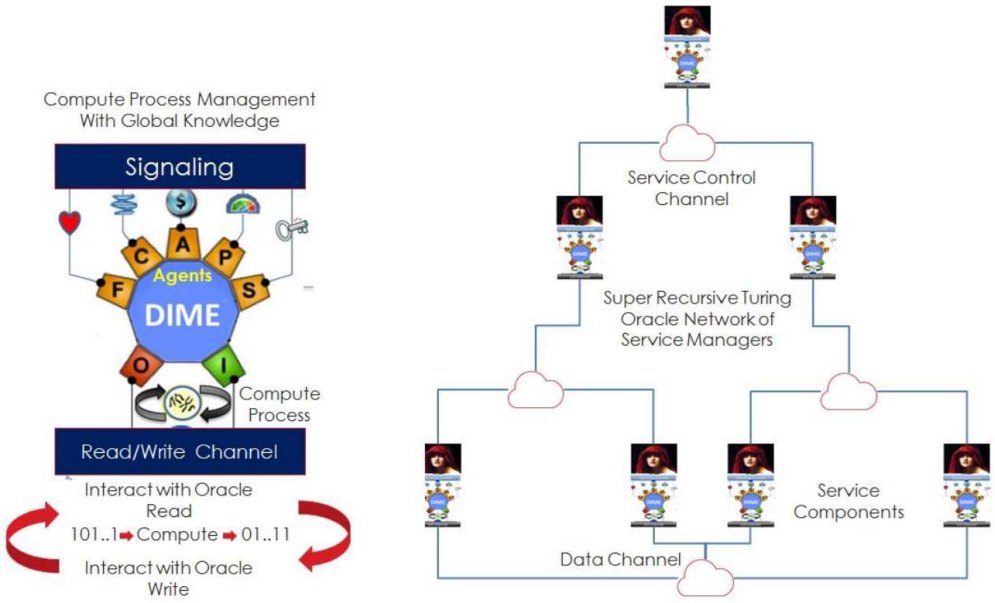
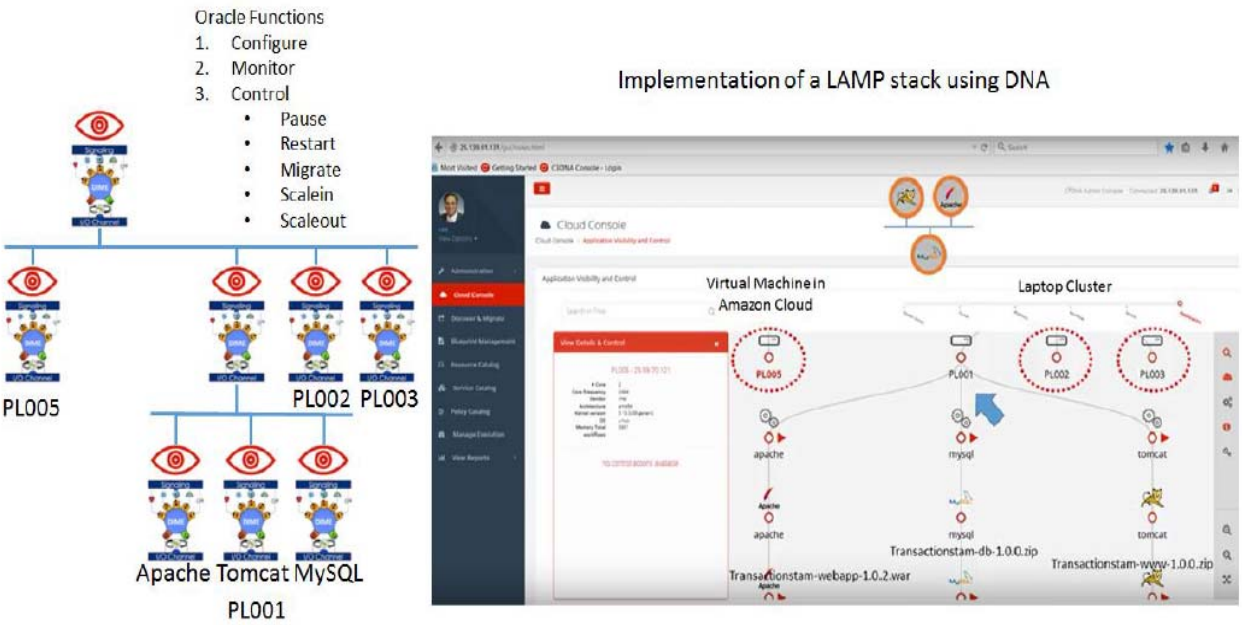


Figure 4 DNA implementation of a managed LAMP stack where individual component or a group of components are configured, monitored and managed to meet the availability, performance and security constraints are fulfilled using auto-scaling, auto-failover and live-migration (see the video demonstrating this application http://www.c3dna.com/cognet.html)



In this implementation a Linux, Apache, MySQL and PHP (LAMP) stack is used to create a transaction intensive application where the Tomcat application writes transactions to the MySQL database and a web application in Apache reads the data and shows the IP address of the database and the transaction number with a time stamp. The service managers are used to implement various oracle functions shown in the figure to configure, monitor and control downstream components.

The LAMP stack is a typical enterprise three tier application to serve web-scale delivery of applications such as e-commerce. The need for large scale and fluctuations in workloads and available resource pools make the resource reconfiguration a real-time requirement which is an ideal requirement for DNA implementation of workload management.

The important aspect of this demonstration is the ability to migrate a live database without losing transactions from one laptop to another or a Virtual Machine in Amazon Web Services cloud without interrupting the transactions being written to the database. This approach is much simpler, less complex using no virtual machine image motion and more efficient because it requires no changes to the application, or Operating System or current infrastructure provisioning processes.

The DIME computing model, in essence, creates a hierarchical managed Universal Turing Machine which executes a process to configure monitor and manage downstream managed Turing machines. At each level, the intent, available resource pools, configuration constraints, self-monitoring methods, and policies and methods to detect and correct any deviations from the intended course of evolution.

According to Mark Burgin "efficiency of an algorithm depends on two parameters: power of the algorithm and the resources that are used in the process of solution. If the algorithm does not have necessary resources, it cannot solve the problem under consideration" (Burgin, 2005).

While the SPC computing model addresses how the computation is executed, it does not address the allocation of resources and their sustenance to complete the computation. The management of the evolution of the computation and the sustenance of the resources is an afterthought.

In the data centre, the computing resources required for the computation depend both on their availability in terms of CPU, memory, network bandwidth, latency, storage capacity, IOPs and throughput characteristics of the hardware and also on the nature of the algorithm (the software). The efficiency of execution of the computation depends upon managing the dynamic relationship of the hardware and the software to monitor the fluctuations and adjusting the resources to execute the computation. Often, based on the function of the computation, the structure of the computing resources may be altered (scaling, migration etc.) to meet the fluctuating demands for resources dictated by the computation. Thus, function, structure and fluctuations dictate the evolution of computation. In order to improve the efficiency of computation, the knowledge of how to manage the dynamics that is outside the purview of the computation is necessary, as well as an external model that integrates the computer (the hardware resources), the computed (the software), and monitoring and management of their evolution in a unified fashion. This challenges the current cloud computing wisdom that promotes cloud native approach and adopting stateless micro-service architecture. Another application of DNA is illustrates interoperability of multiple cloud islands provided by different service providers and enables cloud agnostic computing and workflow management. Figure 5 shows how ten different cloud service islands are integrated to provide a single cloud fabric where applications can be deployed, configured, monitored and managed to auto-failover or auto-scale or live migrate workloads without service interruption based on pre-determine best practice policies.

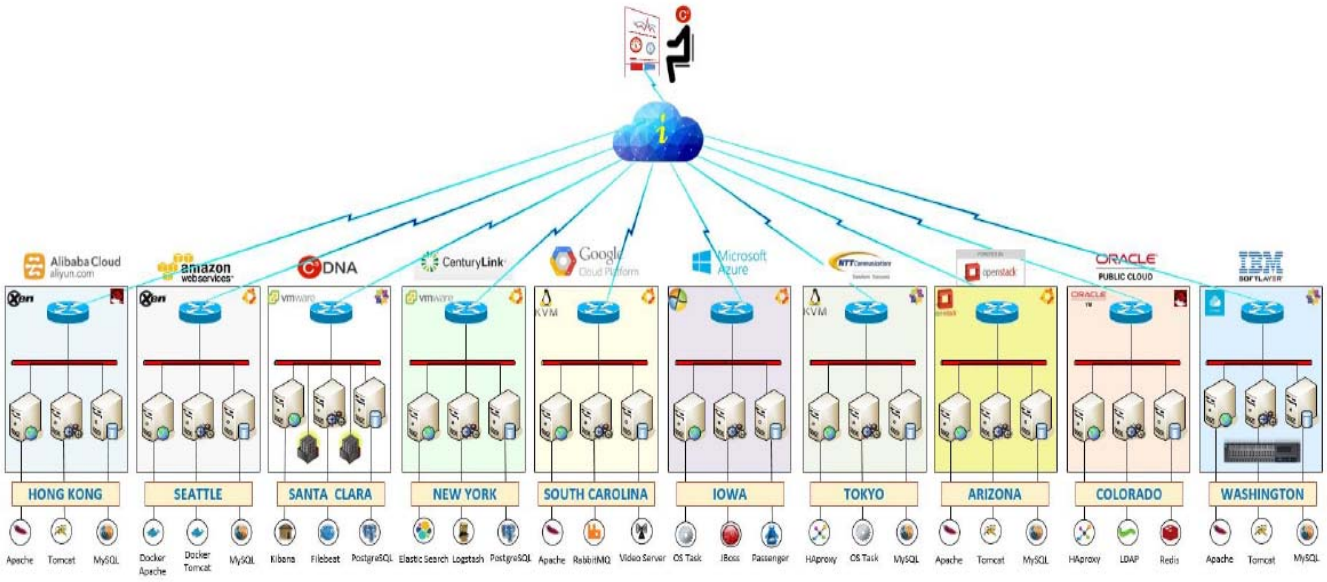
The DNA implementation (Mikkilineni et al., 2015; Mikkilineni and Morana, 2016) uses the specification of the workload and its evolution in the form of policies specifying its composition, structure, configuration methods, monitoring details and best practice controls to manage anticipated deviations from a blueprint. A derived resource blueprint is used to provision the required infrastructure using the knowledge about various resource pools and their provisioning processes. The agent hierarchy at the component, group and workflow management level provide the managed evolution of the workflow. If a failure occurs, the agents detect it and take corrective actions specified including recovery methods. If workloads require scaling of the structure to improve performance, the agents use the specific methods to provide auto-scaling and data synchronisation etc., using the best practice specification. An infrastructure controller provisions the servers, network and storage using the resource blueprint, adds software components and agents that manage the components. A service controller detects the agents and establishes policy based workflow evolution processes.

Applications deployed in Docker containers or virtual machines in an interoperable cloud network (Alibaba, AWS, CenturyLink, Google, Cisco OpenStack based Intercloud, IBM SoftLayer, VMware private cloud, Oracle cloud, Microsoft Azure Cloud and NTT communication cloud in Tokyo) or bare metal servers (in SoftLayer) are made mobile using the workflow management agents with global knowledge and downstream application component monitoring and control.

A video demonstrates this implementation where the application availability and performance are managed using the service managers discussed in this paper.

The local workflow DIME agent assures the availability, performance and security of the node under its control and the global workflow manager assures the availability, performance and security of the sub-process under its impact and control abstractions. The DIME unit at run time provides self-management and supports evolution of computation while computation is still in progress. It is also shown that the DIME computing model supports the genetic transactions of replication, repair, recombination and reconfiguration (Mikkilineni, 2011). This approach eliminates the need for redesigning legacy applications to be cloud aware or stateless in order to scale and adjust its execution quality of service to meet the fluctuating demands from the environment.

Figure 5 Ten different service islands are interconnected using DNA to provide application interoperability, auto-failover, auto-scaling and live migration without service interruption in the face of large fluctuations in workload demands and available resource pools. No virtual image motion or point solutions are used for application mobility



DIME unit is like a stem cell, whose function can be defined along with its context, constraints, and communication.

The result is a new computing model, a management model and a programming model, which using intelligent data, infuse self-awareness into a group of software components deployed on a distributed cluster of hardware devices while enabling the monitoring and control of the dynamics of computation to conform to the intent of the computational process. The DNA based control architecture configures appropriately the software and hardware components to execute the intent. As the computation evolves, the control agents monitor the evolution and makes appropriate adjustments to maintain an equilibrium conforming to the intent. When the fluctuations create conditions for unstable equilibrium, the control agents reconfigure the structure in order to create a new equilibrium state that conforms to the intent based on policies.

In the above implementation, the hardware components are managed dynamically to configure an elastic distributed computing cluster (DCC) to provide the required resources to execute the computations. Modelling a network as an extremely large dynamic memory, we see that the DNA agent organises connection in this network in the same way as one inductive Turing machine builds connections in the memory of another one inductive Turing machine. The software components are organised as managed Turing machines with a control architecture aimed at creation of an "application area network" (AAN) that can be monitored and controlled to execute the intent using the network management abstractions of replication, repair, recombination and reconfiguration. Utilisation of DNA allows the data centres to evolve from being to becoming (Prigogine, 1981). The DIME computing model allows the specification and execution of a recursive composition model where the computing units at any level specifies and executes the workflow at the lower level. Using the DIME computing model, it is possible to deliver higher availability, performance and security to applications than the distributed infrastructure on which they execute. If the requirements are not met at any point of time, the application choses to seek the best execution venue in real time using global knowledge.

5 State-of-the-art IT versus cognitive IT with DNA

Scale and fluctuations are at the heart of IT transformation. Current state of the art cloud computing with virtual infrastructure instead of physical infrastructure provides a way to deal with scale and fluctuations in a timely fashion. However, current cloud technologies, if they offer even 99.99 availability, fall short in providing proactive security, and performance management in real time that mission critical enterprise applications are demanding to meet their business requirements without increasing cost, complexity, tool fatigue and or vendor lock in.

The increase in the complexity of management of services and their availability, performance and security using virtualised and cloud-based resources is a symptom, not of software design or operational execution, but rather a fundamental architectural issue related to Gödel's theorem on self-reflection in Turing machines. Cockshott et al., conclude that, "The key property of general-purpose computers is that they are general purpose. We can use them to deterministically model any physical system, of which they are not themselves a part, to an arbitrary degree of accuracy. Their logical limits arise when we try to get them to model a part of the world that includes themselves" (Cockshott et al., 2012). Therein the problem lies. Automation of dynamic resource administration at run-time makes the computer itself a part of the model and, therefore, the increase in complexity due to increased layers of

management is really a manifestation of an architectural shortfall of current computing models with the usual "who manages the managers conundrum." The limitation becomes pronounced when large scale fluctuations both in workload demand and the availability of resource pools need to be accommodated in real-time. Cognitive computing with global knowledge of both the demand and availability allows local adjustments to be made based on policies and best practices.

The DIME network architecture provides the following functions to implement cognitive processes using distributed computing machines to implement an intent involving the reciprocal influence of "bottom-up" and "top-down" processes (Mikkilineni et al., 2012a; Mikkilineni, 2012b):

1 Managing the "Life" of a Cognitive Process – Policy based resource assurance for system-wide process execution: Each instance provides self-management of resources based on local policies and an ability to load and execute a process using local operating system.

2 Instantiating a distributed process flow with scale-invariant composition and management abstractions: Each instance has a unique identity upon instantiation. A regulator allows provisioning, provides heart-beat, security, performance and account management for each process it executes. A signalling scheme (providing addressing, alerting, mediation and supervision) to facilitate interaction with other instances. The signalling allows same regulation features as one instance for a group of instances (application sub-network and network level).

3 Supporting embodied, embedded, enacted and extended process flows using a DIME network with a system-wide awareness: The DIME network provides a natural framework for implementing reliable managed cognitive processes that are embodied (partly constituted by distributed and specialised structures and processes). Individual processes at the leaf level may be embedded (designed to function only in tandem with system's environment. By interacting with the environment through embedded processes, the DIME network supports cognitive processes that are enacted. The distributed nature and interaction with environment provides the execution of extended cognitive processes using the recursive composition capability of the DIME network.

4 Dynamic process Management: The run-time monitoring of both resource utilisation and process execution along with signalling capabilities allows policy based intervention in each instance while computation is in progress using its agent behaviour of each instance.

6 Conclusion

The DIME network architecture (DNA) introduced in WETICE 2010 has been well described in the literature (cf., for example, Mikkilineni, 2011), while its implementation for providing distributed application management and assuring availability, performance and security is described in a paper in WETICE 2016 (Mikkilineni et al., 2016). It is important to note that DNA implements features that cannot be accomplished today with the current state of art of cloud computing:

- Migrating a workflow executed in a physical server (a web service transaction including a webserver, application server and a database) to another physical server without a reboot or losing transactions while maintaining recovery time and recovery point objectives.
- Provide workflow auto-scaling, auto-failover and live migration using distributed computing clusters with heterogeneous infrastructure (bare metal servers, private and public clouds etc.) without infrastructure orchestration to accomplish them (e.g., moving virtual machine images).
- Create an interoperable private and public cloud network and enable cloud agnostic computing with workflow auto-failover, auto-scaling and live migration across clouds without losing transactions.

Theoretical results demonstrate that DNA extends the current computing model by infusing sensors and actuators into the conventional models of computation, such as Turing machine, as well as in more advanced models, such as inductive Turing machine, supplying cognitive behaviour with super-recursive computing. It is also proved that the DNA model is more efficient, powerful and expressive than the current Turing model based recursive computations (Burgin, 2005; Burgin and Mikkilineni, 2016; Burgin et al., 2016). Consequently, DNA essentially increases the power and possibilities of the contemporary information processing technology.

References

Beros, A.A. (2013) *Learning Theory in the Arithmetical Hierarchy, Preprint in mathematics*, math.LO/1302.7069, 2013 (electronic edition: http://arXiv.org).

Bradshaw, J. (1997) (Ed.) *Software Agents*, AAAI Press/MIT Press, Menlo Park, CA.

Burgin, M. (1983) 'Inductive Turing machines', *Notices of the Academy of Sciences of the USSR*, Vol. 270, No. 6, pp.1289–1293.

Burgin, M. (1999) 'Super-recursive algorithms as a tool for high performance computing', *Proceedings of the High Performance Computing Symposium*, San Diego, pp.224–228.

Burgin, M. (2003) 'Nonlinear phenomena in spaces of algorithms', *International Journal of Computer Mathematics*, Vol. 80, No. 12, pp.1449–1476.

Burgin, M. (2003a) 'Cluster computers and Grid automata', *Proceedings of the ISCA 17th International Conference Computers and their Applications*, International Society for Computers and their Applications, Honolulu, Hawaii, pp.106–109.

Burgin, M. (1997) 'Algorithmic complexity of recursive and inductive algorithms', *Theoretical Computer Science*, Vol. 317, Nos. 1/3, pp.31–60.

Burgin, M. (2005) *Super-recursive Algorithms*, Springer, New York.

Burgin, M. (2006) 'Algorithmic control in concurrent computations', *Proceedings of the 2006 International Conference on Foundations of Computer Science*, CSREA Press, Las Vegas, June, pp.17–23.

Burgin, M. (2010) 'Algorithmic complexity of computational problems', *International Journal of Computing & Information Technology*, Vol. 2, No. 1, pp.149–187.

Burgin, M. (2016) 'On the power of oracles in the context of hierarchical intelligence', *Journal of Artificial Intelligence Research & Advances*, Vol. 3, No. 2, pp.6–17.

Burgin, M. (2016a) 'Decreasing complexity in inductive computations', *Advances in Unconventional Computing, series Emergence, Complexity and Computation*, Springer, Vol. 22, pp.183–203.

Burgin, M., Calude, C.S. and Calude, E. (2013) 'Inductive complexity measures for mathematical problems', *International Journal of Foundations of Computer Science*, Vol. 24, No. 4, pp.487–500.

Burgin, M. and Debnath, N. (2004) 'Measuring software maintenance', *Proceedings of the ISCA 19th International Conference Computers and their Applications*, ISCA, Seattle, Washington, pp.118–121.

Burgin, M. and Debnath, N. (2005) 'Complexity measures for software engineering', *Journal for Computational Methods in Science and Engineering*, Vol. 5, Supplement 1, pp.127–143.

Burgin, M. and Debnath, N. (2009) 'Super-recursive algorithms in testing distributed systems', *Proceedings of the ISCA 24th International Conference Computers and their Applications* (*CATA-2009*), ISCA, New Orleans, Louisiana, USA, April, pp.209–214.

Burgin, M., Debnath, N. and Lee, H.K. (2009) 'Measuring testing as a distributed component of the software life cycle', *Journal for Computational Methods in Science and Engineering*, Vol. 29, Nos. 1/2, pp.211–223.

Burgin, M. and Eberbach, E. (2009) 'Universality for Turing machines, inductive Turing machines and evolutionary algorithms', *Fundamenta Informaticae*, Vol. 91, No. 1, pp.3–77.

Burgin, M. and Eberbach, E. (2009a) 'On foundations of evolutionary computation: an evolutionary automata approach', in Mo, H. (Ed.): *Handbook of Research on Artificial Immune Systems and Natural Computing: Applying Complex Adaptive Technologies*, IGI Global, Hershey, Pennsylvania, pp.342–360.

Burgin, M. and Eberbach, E. (2012) 'Evolutionary automata: expressiveness and convergence of evolutionary computation', *Computer Journal*, Vol. 55, No. 9, pp.1023–1029.

Burgin, M. and Gupta, B. (2012) 'Second-level algorithms, superrecursivity, and recovery problem in distributed systems', *Theory of Computing Systems*, Vol. 50, No. 4, pp.694–705.

Burgin, M. and Klinger, A. (2004) 'Experience, generations, and limits in machine learning', *Theoretical Computer Science*, Vol. 317, Nos. 1/3, pp.71–91.

Burgin, M. and Mikkilineni, R. (2014) 'Semantic network organization based on distributed intelligent managed elements', *Proceeding of the 6th International Conference on Advances in Future Internet*, Lisbon, Portugal, pp.16–20.

Burgin, M., Mikkilineni, R. and Morana, G. (2016) 'Intelligent organization of semantic networks, DIME network architecture and grid automata', *International Journal of Embedded Systems*, Vol. 8, No. 4.

Calude, C.S., Calude, E. and Queen, M.S. (2012) 'Inductive complexity of P versus NP problem, unconventional computation and natural computation', *Lecture Notes in Computer Science*, Vol. 7445, pp.2–9.

Chaitin, G.J. (1977) 'Algorithmic information theory', *IBM Journal of Research and Development*, Vol. 21, No. 4, pp.350–359.

Cockshott, P., MacKenzie, L.M. and Michaelson, G. (2012) *Computation and Its Limits*, Oxford University Press, Oxford.

Dinverno, M. and Luck, M. (Eds) (2001) *Understanding Agent Systems*, Springer Verlag, New York.

Doyle, J. (1983) 'What is rational psychology? Toward a modern mental philosophy', *AI Magazine*, Vol. 4, No. 3, pp.50–53.

Dyson, F.J. (1972) 'Missed opportunities', *Bull. Amer. Math. Soc*., Vol. 78, pp.635–652.

Funding a Revolution (1999) *Government Support for Computing Research*, Committee on Innovations in Computing and Communications: Lessons from, History, National Research Council, p.184.

Gold, E.M. (1967) 'Language identification in the limit', *Information and Control*, Vol. 10, pp.447–474

Hertel, J. (2012) 'Inductive complexity of Goodstein's theorem, unconventional computation and natural computation', *Lecture Notes in Computer Science*, Vol. 7445, pp.141–151.

Hewitt, C., Bishop, P. and Steiger, R. (1973) 'A universal modular actor formalism for artificial intelligence', *Proceedings of the 3rd international joint conference on Artificial intelligence (IJCAI'73)*, pp.235–245.

Homer, S. and Selman, A.L. (2011) *Relative Computability, in Computability and Complexity Theory, series Texts in Computer Science*, Springer, pp.145–179.

Luck, M. et al. (2005) *Agent technology: computing as interaction (a roadmap for agent based computing)*, University of Southampton. Available online at: http://www.inf.kcl.ac.uk/staff/mml/papers/al3roadmap.pdf

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

Mikkilineni, R., Comparini, A. and Morana, G. (2012a) 'The Turing o-machine and the DIME Network Architecture: Injecting the Architectural Resiliency into Distributed Computing', *Turing-100, The Alan Turing Centenary, EasyChair Proceedings in Computing*. Available online at: www.easychair.org/publications/?page=877986046 (accessed on 10 October 2016).

Mikkilineni, R., Morana, G., Zito, D. and Di Sano, M. (2012b) 'Service virtualization using a non-von Neumann parallel, distributed, and scalable computing model', *Journal of Computer Networks and Communications*, Vol. 2012, Article ID 604018, 10 pages.

Mikkilineni, R. (2012) 'Going beyond computation and its limits: injecting cognition into computing', *Applied Mathematics*, Vol. 3, No. 11A, pp.1826–1835.

Mikkilineni, R., Morana, G. and Zito, D. (2015) '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*, pp.51–56.

Mikkilineni, R., Morana, G. and Keshan, S. (2016) 'Demonstration of a New Computing Model to Manage a Distributed Application and Its Resources Using Turing Oracle Design', *2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, Paris, pp.253–254.

Mikkilineni, R. and Morana, G. (2016) 'Cognitive distributed computing: a new approach to distributed datacenters with self-managing services on commodity hardware', *International Journal of Grid and Utility Computing*, Vol. 7, No. 2.

Milner, R. (1993) 'Elements of interaction', *Communications of the ACM*, Vol. 36, No. 1, pp.78–89.

Minsky, M. (1986) *The Society of Mind*, Simon and Schuster, New York.

Prigogine, I. (1981) *From Being to Becoming: Time and Complexity in the Physical Sciences*, W.H. Freeman & Co. New York. Also see https://computingclouds.wordpress. com/2015/02/16/from-being-to-becoming-function-structure-and-fluctuations-incremental-versus-leap-frog-innovation-in-datacenters-2/

Rogers, H. (1987) *Theory of Recursive Functions and Effective Computability*, MIT Press, Cambridge, MA.

Savage, J.E., Selman, A.L. and Smith, C. (2001) 'History and contributions of theoretical computer science', *Advances in Computers*, Vol. 55, pp.171–183.

Soare, R.I. (2015) 'Turing Oracle machines, online computing, and three displacements in computability theory', *Annals of Pure and Applied Logic*, Vol. 160, pp.368–399.