Semantic Network Organization Based on Distributed Intelligent Managed Elements

Improving efficiency and resiliency of computational processes

Mark Burgin
UCLA
Los Angeles, USA
mburgin@math.ucla.edu

Rao Mikkilineni
C^3 DNA Inc.
Santa Clara, USA
rao@c3dna.com

***Abstract* — A new network architecture based on increasing intelligence of the computing nodes is suggested for building the semantic grid. In its simplest form, the distributed intelligent managed element (DIME) network architecture extends the conventional computational model of information processing networks, allowing improvement of the efficiency and resiliency of computational processes. This approach is based on organizing the process dynamics under the supervision of intelligent agents. The DIME network architecture utilizes the DIME computing model with non-von Neumann parallel implementation of a managed Turing machine with a signaling network overlay and adds cognitive elements to evolve super recursive information processing, for which it is proved that they improve efficiency and power of computational processes. The main aim of this paper is modeling the DIME network architecture with grid automata. A grid automaton provides a universal model for computer networks, sensor networks and many kinds of other networks.**

Keywords - semantic network; DIME network architecture; grid automaton; structural operation; connectivity; modularity; Turing O-Machine; cloud computing.

I. INTRODUCTION

Information processing networks play more and more important role in society. For instance, close to a billion hosts are connected to the Internet. The rapid rise in popularity of the Internet is due to the World Wide Web (WWW), search engines, e-mail, social networking and instant communication systems, which enable high-speed and resourceful exchanges and transformation of information, as well as provide unlimited access to a huge amount of information [1].

Recently, cloud and grid computing have been regarded as the most promising paradigms to interconnect heterogeneous commodity computing environments. To make it more efficient, the concept of the semantic web or semantic grid was introduced as a new level of the Internet and the World Wide Web. This new level is based on establishing a new form of Web content that is meaningful to computers. The Semantic Web proposes to help computers in obtaining information from the Web and using it for achieving various goals. The first step is to add metadata to Web pages making the existing World Wide Web machine comprehensible and providing machine tools to find, exchange, and to a limited extent, interpret information. Being an extension of, but not a replacement for, the World Wide Web, this approach will unleash a revolution of new possibilities.

In this paper, the distributed intelligent managed element (DIME) network architecture [2, 3, 4, 5, 6] previously discussed at the Turing Centenary Conference [7] in Manchester, is aimed at the development of semantic networks extending the conventional computational model of the network architecture. It is aimed at improving efficiency and resiliency of computational processes by organizing their evolution to model process dynamics under the supervision of intelligent agents. The computing hardware resources are combined with software functions to arrange processes and their dynamics using a network of DIMEs where each end node can be either a DIME unit or a sub-network of DIME units executing a workflow. The hardware resources are characterized by their parameters such as the required CPU, memory, network bandwidth, latency, storage throughput, IOPs and capacity. The efficiency of computation is determined by the required resources, while the expressiveness of the computational process dynamics is established by the structure of the DIME units and connecting hardware units, such as servers or routers, along with its interactions within and with the external world.

The suggested approach to the semantic web lies in provisioning of resource descriptions and ontologies to DIME agents. The agent would search through metadata that clearly identify and define what the agent needs to know. Metadata are machine-readable data that describe other data. In the Semantic Web, metadata are invisible as people read the page, but they are clearly visible to DIME agents. Metadata can also allow more complex, focused Web searches with more accurate results and interpreting these data for controlling DIME basic processors.

To achieve all these goals, it is necessary to base the entire design of the whole network of applications, as well as of the components that build the network, on a system technology with flexibility to interconnect different applications and devices from different vendors. Rigid standards may be suitable to meet a short term requirement, but in the long run, they will limit choices as it will inhibit innovation. System technology, in turn, provides efficient design methods and results in creation of better networks, which satisfy necessary requirements. All these requirements demand a new approach to application and device network design, upgrading, and maintenance.

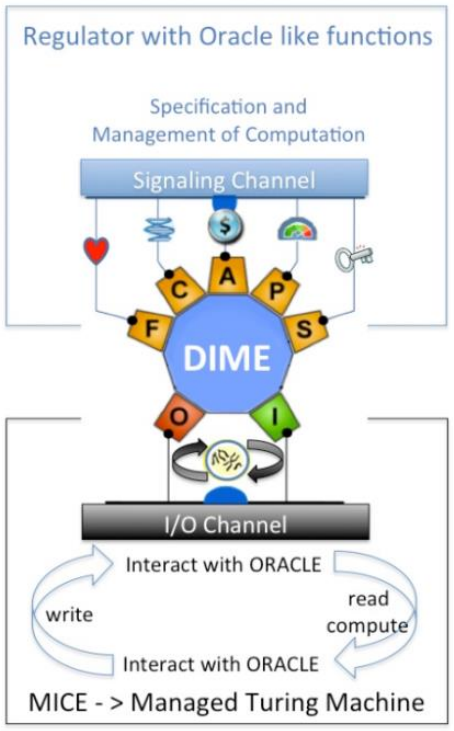
Regulator with Oracle like functions
Specification and Management of Computation
Signaling Channel
C
A
P
F
S
DIME
O
I
I/O Channel
Interact with ORACLE
write
read
compute
Interact with ORACLE
MICE - > Managed Turing Machine

Figure 1: A Distributed Intelligent Managed Element is a managed Turing Oracle Machine endowed with a signaling network overlay for policy based DIME network management

Here, we develop tools for such a systemic network design, upgrading, and maintenance based on three principles: 1) modularity; 2) system representation of each module by grid automata; and 3) utilization of modular operations with networks, which are introduced in this paper. Modular approach means division of a complex system into smaller, manageable ones, making implementation much easier to handle.

Section II, reviews the DIME network architecture and current state of the art. Section III presents a review of the theory of Grid Automata and Grid Arrays. Section IV describes modeling DIME networks with Grid Automata, while in Section V, some conclusions are considered and directions for future work are suggested.

II. DIME Network Architecture

The DIME network architecture introduces three key functional constructs to enable process design, execution and management to improve both resiliency and efficiency of computer networks [2, 3, 5].

1) Machines with an Oracle

Executing an algorithm, the DIME basic processor *P* performs the {read -> compute -> write} instruction cycle or its modified version the {interact with a network agent -> read -> compute -> interact with a network agent -> write} instruction cycle. This allows the different network agents to influence the further evolution of computation, while the computation is still in progress. We consider three types of network agents:

(a) A DIME agent.
(b) A human agent.
(c) An external computing agent.

It is assumed that 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 realize its intent. Thus, every algorithm is associated with a blueprint (analogous to a genetic specification in biology), which provides 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.

2) Blue-print or policy managed fault, configuration, accounting, performance and security monitoring and control

The DIME agent, which uses the blueprint to configure, instantiate, and manage the DIME basic processor executing the algorithm uses concurrent DIME basic processors with their own blueprints specifying their evolution to monitor the vital signs of the DIME basic processor and implements various policies to assure non-functional requirements such as availability, performance, security and cost management while the managed DIME basic processor is executing its intent. Figure 1 shows the DIME basic processor and its DIME agent, which manages it using the knowledge provided by the blueprint [3, 7].

3) DIME network management control overlay over the managed Turing oracle machines

In addition to read/write communication of the DIME basic processor (the data channel), other DIME basic processors communicate with each other using a parallel signaling 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 are DIMEs themselves. As a result, 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 signaling channel and the network of DIME agents can be programmed to execute a process, the intent of which 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 agents specify, configure, and manage the sub-network of DIME units by monitoring and executing policies to optimize the resources while delivering the intent.

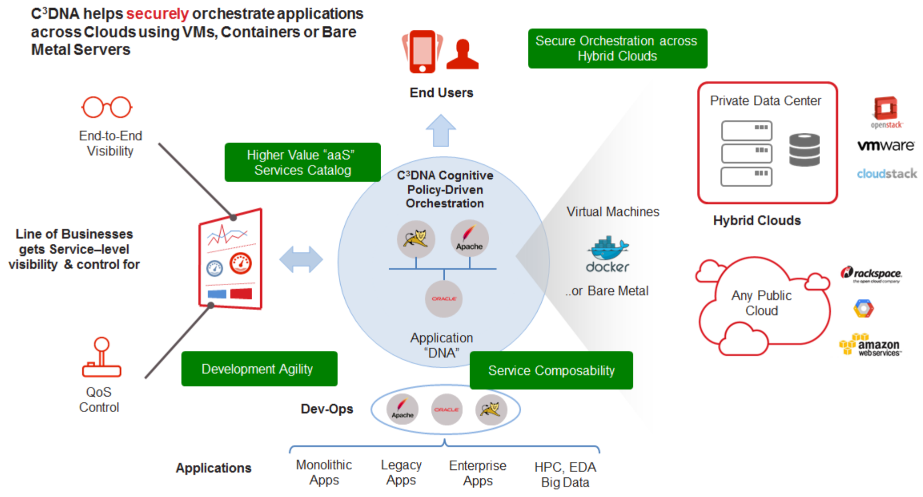
C3DNA helps securely orchestrate applications across Clouds using VMs, Containers or Bare Metal Servers
End Users
Secure Orchestration across Hybrid Clouds
End-to-End Visibility
Higher Value "aaS" Services Catalog
C3DNA Cognitive Policy-Driven Orchestration
Private Data Center
Hybrid Clouds
Line of Businesses gets Service–level visibility & control for
Virtual Machines
docker
..or Bare Metal
Any Public Cloud
Application "DNA"
QoS Control
Development Agility
Service Composability
Dev-Ops
Applications
Monolithic Apps
Legacy Apps
Enterprise Apps
HPC, EDA Big Data

Figure 2: Implementation architecture of a Web application workflow using a physical server network and a cloud using Virtual Machines

Figure 2 shows the DIME network implementation architecture for a process with different hardware, functions and an evolving structure used to attaining the intent of the process.

This architecture has following benefits from current architectures deploying virtual machines to provide cloud services such as self-provisioning, self-repair, auto-scaling and live-migration:

1. Using DNA, same cloud services can be provided at application and workflow group level across physical or virtual servers. The mobility of applications comes from utilization of the policies implemented to manage the intent through the signaling network overlay over the managed computer network. Applications are moved into static Virtual Machines with given service levels provisioned.

2. Scheduling, monitoring, and managing distributed components and groups with policies at various levels de-couple the application/workflow management from underlying distributed infrastructure management systems. The vital signs (cpu, memory, bandwidth, latency, storage IOPs, throughput and capacity) are monitored and managed by DIMEs, which are functioning similar to the Turing o-machines.

While implementing the monitoring and management of the DIME agent, the DIME network monitors and manages its own vital signs and executing various 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. In figure 2, each web component has its own policies and the group has the service level policies that define its availability, performance and security. Based on policies, the elements are replicated or reconfigured to meet the resource requirements based on monitored behavior.

In essence, the DIME computing model infuses sensors and actuators connecting the DIME basic processor with the DIME agent to manage the DIME basic processor and its resources based on the intent, interactions and available resources. Policy managers are used to configure, monitor and manage the basic processor's intent. The DIME network architecture has been successfully implemented using 1) a Linux operating system and 2) a new native operating system called parallax [2, 3]. More recently, a product based on DIME network architecture was used to implement auto-failover, auto-scaling, and live-migration of a web based application deployed on distributed servers with or without virtualization [8]. In this paper we model the DIME network architecture with grid automata. A grid automaton [9] is shown to be more efficient and expressive than the von Neumann implementation of the Turing Machine. In the next section, we review the Grid Automata and Grid Arrays.

III. GRID AUTOMATA AND GRID ARRAYS

All computer and embedded system networks, as well as their software, are grid arrays in the sense of [9, 10, 11]. The Internet is a grid array. The Grid [12, 13] is also a grid array. Computing grid arrays consist of computing devices connected by some ties, e.g., channels. Grid automata provide theoretical models for grid arrays and thus, for computer software, hardware, networks and many other systems. At first, we give an informal definition.

Definition 1. A grid automaton is a system of abstract automata and their networks, which are situated in a grid, are called nodes, are optionally connected and interact with one another.

The difference is that a grid array consists of real (physical) information processing systems and connections between them, while a grid automaton consists of abstract automata as its nodes. Nodes in a grid automaton can be finite automata, Turing machines, vector machines, array machines, random access machines, inductive Turing machines, and so on. Even more, some of the nodes can be also grid automata.

Translating this definition into the mathematical language, two types of grid automata - *basic grid automata* and *grid automata with ports* – are considered.

The basic idea of interacting processes is for a transmitting process to send a message using a port and for the receiving process to get the message from another port. To formalize this structure, we assume, as it is often true in reality, that connections are attached to automata by means of ports. Ports are specific automaton elements through which data come into (*input ports* or *inlets*) and send outside the automaton (*output ports* or *outlets*). Thus, any system P of ports is the union of its two (possibly) disjoint subsets $P = P_{in} \cup P_{out}$ where P_{in} consists of all inlets from P and P_{out} consists of all outlets from P. If in the real system, there are ports that are both inlets and outlets, in the model, we split them, i.e., represented such ports as pairs consisting of an input port and an output port. There are different other types of ports. For instance, contemporary computers have parallel and serial ports. Ports can have inner structure, but in the first approximation, it is possible to consider them as elementary units.

We also assume that each connection is directed, i.e., it has the beginning and end. It is possible to build bidirectional connections from directed connections.

Let us consider a class of automata **B** with ports of types **T** and a class of connections/links **L** that can be connected to automata from **B**.

Definition 2. A (*port*) *grid automaton* $\boldsymbol{G}$ over the collection (**B**, **P**, **L**), which is called *accessible hardware*, is the system

$\boldsymbol{G} = (A_G, P_G, C_G, \boldsymbol{p}_{IG}, \boldsymbol{c}_G, \boldsymbol{p}_{EG})$

that consists of three sets and three mappings:

- A_G is the set of all automata from $\boldsymbol{G}$, assuming $A_G \subseteq$ **B**;
- C_G is the set of all links from $\boldsymbol{G}$, assuming $C_G \subseteq$ **L**;
- $P_G = P_{IG} \cup P_{EG}$ (with $P_{IG} \cap P_{EG} = \varnothing$) is the set of all ports of G, assuming $P_G \subseteq$ **P**, where P_{IG} is the set of all ports (called *internal ports*) of the automata from A_G , and P_{EG} is the set of *external ports* of G, which are used for interaction of G with different external systems;
- $\boldsymbol{p}_{IG} : P_{IG} \to A_G$ is a total function, called the *internal port assignment function*, that assigns ports to automata;
- $\boldsymbol{c}_G : C_G \to (P_{IGout} \times P_{IGin}) \cup P'_{IGin} \cup P''_{IGout}$ is a (eventually, partial) function, called the *port-link adjacency function*, that assigns connections to ports where P'_{IGin} and P''_{IGout} are disjunctive copies of P_{IGin} ;
- $\boldsymbol{p}_{EG} : P_{EG} \to A_G \cup P_{IG} \cup C_G$ is a function, called the *external port assignment function*, that assigns ports to different elements from G.

To have meaningful assignments of ports, the port assignment functions $\boldsymbol{p}_{IG}$ and $\boldsymbol{p}_{EG}$ have to satisfy some additional conditions.

Examples:

1. The screen of a computer monitor is an output port. Such a screen can be also treated as a system of output ports (pixels).

2. The mouse of a computer is an input port. It can be also treated as a system of input ports.

3. The touch screen of a computer is an input port. Such a screen can be also treated as a system of input ports.

Definition 3. A *basic grid automaton* $\boldsymbol{A}$ over the collection (**B**, **L**), which is called *accessible hardware*, is a system $\boldsymbol{A} = (A_A , C_A , \boldsymbol{c}_A)$ that consists of two sets and one mapping:

- the set A_A is the set of all automata from $\boldsymbol{A}$, assuming $A_A \subseteq$ **B**;
- the set C_A is the set of all connections/links from $\boldsymbol{A}$, assuming $C_A \subseteq$ **L**;
- the mapping $\boldsymbol{c}_A$: $C_A \to A_A \times A_A \cup A'_A \cup A''_A$, which is a (variable) function, called the *node-link adjacency function*, which assigns connections to nodes where A'_A and A''_A are disjunctive copies of A_A .

There are different types of connections. For instance, computer networks links or connections are implemented on a variety of different physical media, including twisted pairs, coaxial cable, optical fiber, and space.

It is possible to group connections in grid arrays and grid automata into three main types:

1. *Simple connections* that are not changing deliberately transmitted data and themselves when the automaton or array is functioning.
2. *Transformable connections* that may be changed when the automaton or array is functioning.
3. *Processing connections* that can transform transmitted data.

A grid automaton G is described by three grid characteristics and three node characteristics.

<u>The grid characteristics are:</u>

1. The *space organization* or *structure* of the grid automaton G.

This space structure may be in the physical space, reflecting where the corresponding information processing systems (nodes) are situated, it may be the system structure defined by physical connections between the nodes, or it may be a mathematical structure defined by the geometry of node relations. System structure is so important in grid arrays that in contemporary computers connections between the main components are organized as a specific device, which is called the computer bus. In a computer or on a network, a bus is a transmission path in form of a device or system of devices on which signals are dropped off or picked up at every device attached to the line.

There are three kinds of space organization of a grid automaton: *static structure* that is always, the same; *persistent dynamic structure* that eventually changes between different cycles of computation; and *flexible dynamic structure* that eventually changes at any time of computation. Persistent Turing machines [14] have persistent dynamic structure, while reflexive Turing machines [15] have flexible dynamic structure and perform emergent computations [16].

2. The *topology* of *G* is determined by the type of the node neighborhood and is usually dependent on the system structure of *G*.

A natural way to define a neighborhood of a node is to take the set of those nodes with which this node directly interacts. In a grid, these are often, but not always, the nodes that are physically the closest to the node in question.

For example, if each node has only two neighbors (right and left), it defines linear topology in *G*. When there are four nodes (upper, below, right, and left), the *G* has two-dimensional rectangular topology.

However, it is possible to have other neighborhoods. For instance, consider linear cellular automata in which the neighborhood of each cell has the radius $r > 1$ [9]. It means that *r* cells from each side of a given cell directly influence functioning of this cell.

3. The *dynamics* of *G* determines by what rules its nodes exchange information with each other and with the environment of *G*.

For example, when the interaction of Turing machines in a grid automaton *G* is determined by a Turing machine, then *G* is equivalent to a Turing machine. At the same time, when the interaction of Turing machines in a grid automaton *G* is random, then *G* is much more powerful than any Turing machine.

<u>The node characteristics are:</u>

1. The *structure* of the node. For example, one structure determines a finite automaton, while another structure is a Turing machine.
2. The *external dynamics* of the node determines interactions of this node.

According to this characteristic there are three types of nodes: *accepting nodes* that only accept or reject their input; *generating nodes* that only produce some input; and *transducing nodes* that both accept some input and produce some input. Note that nodes with the same external dynamics can work in grids with various dynamics.

3. The *internal dynamics* of the node determines what processes go inside this node.

For example, the internal dynamics of a finite automaton is defined by its transition function, while the internal dynamics of a Turing machine is defined by its rules. Differences in internal dynamics of nodes are very important because a change in producing the output allows us to go from conventional Turing machines to much more powerful inductive Turing machines of the first order [17].

Representation of grid automata without ports called basic grid automata is the first approximation to a general network model [9, 1], while representation of grid automata with ports is the second (more exact) approximation. In some cases, it is sufficient to use grid automata without ports, while in other situations, to build an adequate, flexible and efficient model of a network, we need automata with ports. Usually, basic grid automata are used when the modeling scale is big, i.e., at the coarse-grain level, while port grid automata are used when the modeling scale is small and we need a fine-grain model.

Neural networks, cellular automata, systolic arrays, and Petri nets are special kinds of grid automata [9]. However, grid automata provide computer science with much more flexibility, expressive power and correlation with real computational and communication systems than any of these models. In comparison with cellular automata, a grid automaton can contain different kinds of automata as its nodes. For example, finite automata, Turing machines and inductive Turing machines can belong to one and the same grid. In comparison with systolic arrays, connections between different nodes in a grid automaton can be arbitrary like connections in neural networks. In comparison with neural networks and Petri nets, a grid automaton contains, as its nodes, more powerful machines than finite automata. An important property of grid automata is a possibility to realize hierarchical structures, that is, a node can be also a grid automaton. In grid automata, interaction and communication becomes as important as computation. This peculiarity results in a variety of types of automata, their functioning modes, and space organization.

Internal ports of a port grid automaton *B* to which no links are attached are called *open*. External ports of a port grid automaton *B* to which no links or automata are attached are called *free*. External ports of a port grid automaton *B*, being always open, are used for connecting *B* to some external systems.

All ports of a grid automaton are divided into three classes: *input ports*, which can only accept information; *output ports*, which can only transmit information; and *mixed ports*, which can accept and transmit information (in the form of signals or symbols).

This typology of ports, as is used in the general case of information processing systems [9], induces the following classification of grid automata:

1. Grid automata without input and output (called *closed grid automata*).
2. Grid automata with input (called *closed from the right* or *open from the left grid automata*).
3. Grid automata with output (called *closed from the left* or *open from the right grid automata*).

Grid automata with both input and output (called *open grid automata*).

IV. MODELING DIME NETWORKS WITH GRID AUTOMATA

In the context of grid automata, a DIME network is represented by a grid automaton with such nodes as DIME units, servers, routers, etc.

Each DIME unit is modeled by a basic automaton A with an Oracle O. The automaton A models the DIME basic processor P, while the Oracle O models the DIME agent DA. Turing machines with Oracles, inductive Turing machines with Oracles, limit Turing machines with Oracles [15], and evolutionary Turing machines with Oracles [19] are examples of such an automaton A with an Oracle O.

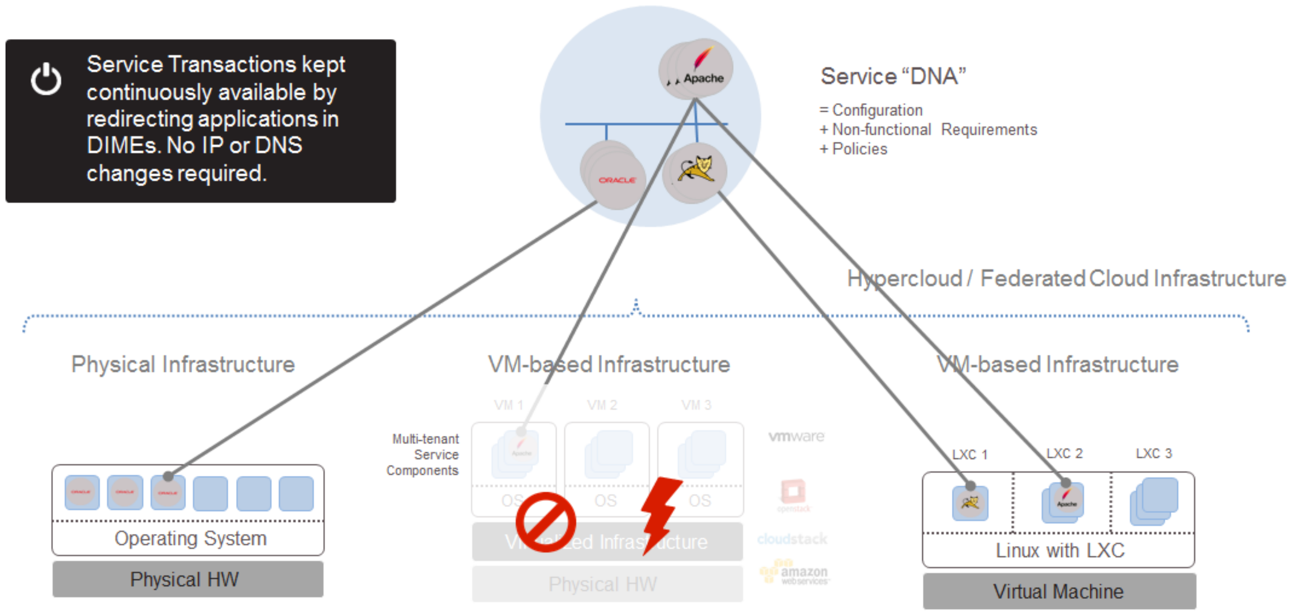

Figure 3: A Web Service Workflow Deployed in a physical server and providing mobility a Virtual server

The Oracle O in a DIME unit knows the intent of the algorithm (along with the context, constraints, communications and control of the algorithm) the basic automaton A is executing under its influence and has the visibility of available resources and the needs of the automaton A as it executes its function. In addition, the Oracle also has the knowledge about alternate courses of action available to facilitate the evolution of the computation to achieve its intent. Thus, every algorithm is associated with a blueprint (analogous to a genetic specification in biology), which can provide the knowledge required by an Oracle to manage its evolution.

In addition to read/write communication of the basic automaton (the data channel), the Oracles manage different basic automata communicating with each other using a parallel signaling channel. This allows the external Oracles to influence the computation of any managed basic automaton in progress based on the context and constraints just as a Turing Oracle is expected to do.

The Oracle uses the blueprint to configure, instantiate, and manage the automaton A executing the algorithm. Utilization of concurrent automata in the network with their own blueprints specifying their evolution to monitor the vital signs of the DIME basic automaton and to implement various policies allows the Oracle to assure non-functional requirements such as availability, performance, security and cost management, while the managed DIME basic automaton is executing its task to achieve its goal and realize its intent.

The external Oracles represent DIME agents, allowing changes in one computing element influence the evolution of another computing element at run time without stopping its basic automaton executing the algorithm. The signaling channel and the network of the Oracles can be programmed to execute a process whose intent itself can be specified in a blueprint. Each basic automaton can have its own Oracle managing its intent, and groups of managed basic automata can have their own domain managers implementing the domain's intent to execute a process. The management Oracles specify, configure and manage the sub-network of DIMEs by monitoring and executing policies to optimize the resources while delivering the intent. The DIME network implementing the Oracles is itself managed by monitoring its own vital signs and executing various FCAPS policies to assure availability, performance and security.

An Oracle is modeled by an abstract automaton that has higher computational power and/or lower computational complexity than the basic automaton it manages. For instance, the Oracle can be an inductive Turing machine, while the basic automaton is a conventional Turing machine. It is proved that inductive Turing machines have much higher computational power and lower complexity than conventional Turing machine [9].

DIME agents possess a possibility to infer new data and knowledge from the given information. Inference is one of the driving principles of the Semantic Web, because it will allow us to create software applications quite easily. For the Semantic Web applications, DIME agents need high expressive power to help users in a wide range of situations. To achieve this, they employ powerful logical tools for making inferences. Inference abilities of DIME agents are developed based on mathematical models of these agents in the form of inductive Turing machines, limit Turing machines [9] and evolutionary Turing machines [18, 19].

Figure 3 shows a workflow DNA of a web application running on a physical infrastructure that has policies to manage auto-failover by moving the components when the vital signs being monitored at various levels are affected. For example if the virtual machine in the middle server fails, the service manager at higher level detects it and replicates the components in another server on the right and synchronizes the states of the components based on consistency policies.

V. Conclusion

Three innovations are introduced, namely, the parallel monitoring of vital signs (cpu, memory, bandwidth, latency, storage IOPs, throughput and capacity) in the DIME, signaling network overlay to provide run-time service management and machines with Oracles in the form of DIME agents. This allows interruption for policy management at read/write in a file/device allow self-repair, auto-scaling, live-migration and end-to-end service transaction security with private key mechanism independent of infrastructure management systems controlling the resources and thus, provide freedom from infrastructure and architecture lock-in. The DIME network architecture puts the safety and survival of applications and groups of applications delivering a service transaction first using secure mobility across physical or virtual servers. It provides information for sectionalizing, isolating, diagnosing and fixing the infrastructure at leisure. The DIME network architecture therefore makes possible reliable services to be delivered on even not-so-reliable infrastructure. Modeling this architecture by grid automata allows researchers to study properties and critical parameters of semantic networks and provides means for optimizing these parameters. Future work will investigate specific predictions that can be made from the theory for a specific DIME network execution and compare the resiliency and efficiency using both recursive and super-recursive implementations.

Acknowledgment

Rao Mikkilineni thanks Giovanni Morana and Ian Seylor for implementing the DIME network architecture and Vijay Sarathy, Nishan Sathyanarayan and Paul Camacho from C^3 DNA Inc., for making it an enterprise-class product.

References

[1] N. Olifer and V. Olifer, Computer networks: Principles, technologies and protocols for network design, New York:Wiley, 2006.

[2] R. Mikkilineni, Designing a new class of distributed systems. New York: Springer, 2011.

[3] R. Mikkilineni, G. Morana and I. Seyler, "Implementing distributed, self-Managing computing services infrastructure using a scalable, parallel and network-centric computing model." In Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice, ed. M. Villari, I. Brandic and F. Tusa, 57-78, 2012.

Accessed September 05, 2014. doi:10.4018/978-1-4666-1631-8.ch004.

[4] R. Mikkilineni, "Architectural resiliency in distributed computing," International Journal of Grid and High Performance Computing (IJGHPC) 4, 2012. Accessed (September 05, 2014), doi:10.4018/jghpc.2012100103.

[5] R. Mikkilineni, G. Morana, D. Zito, and M. Di Sano, “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, 2012. doi:10.1155/2012/604018.

[6] R. Mikkilineni, "Going beyond computation and its limits: Injecting cognition into computing." Applied Mathematics 3, pp. 1826-1835, 2012.

[7] R. Mikkilineni, A. Comparini and G. Morana, “The Turing O-Machine and the DIME network architecture: Injecting the architectural resiliency into distributed computing, In Turing-100. The Alan Turing Centenary, (Ed.) Andrei Voronkov, EasyChair Proceedings in Computing, Volume 10, pp. 239-251, 2012.

http://dx.doi.org/10.1155/2012/604018

[8] R. Mikkilineni and G. Morana, "Infusing cognition into distributed computing: A new approach to distributed datacenters with self-managing services" Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2014 23rd IEEE International Conference, June 2011.

[9] M. Burgin, Super-recursive Algorithms, New York: Springer, 2005.

[10] M. Burgin, From Neural networks to Grid automata, in Proceedings of the IASTED International Conference ”Modeling and Simulation”, Palm Springs, California, 2003

[11] M. Burgin, Cluster computers and Grid automata, in Proceedings of the ISCA 17th International Conference “Computers and their applications”, International Society for Computers and their Applications, Honolulu, Hawaii, pp. 106-109, 2003.

[12] I. Foster, “Computational Grids,” in The Grid: Blueprint for a future computing infrastructure, San Francisco, CA,: Morgan Kauffman, pp. 15-52, 1998.

[13] M. L. Bote-Lorenzo, Y.A.Dimitriadis and E. Gómez-Sánchez, Grid characteristics and uses: a grid definition, in First European Across Grids conference, LNCS 2970, pp. 291–298, 2004.

[14] D. Goldin and P. Wegner, Persistent Turing Machines, Brown University Technical Report, 1988.

[15] M. Burgin, “Reflexive Calculi and Logic of Expert Systems”, in *Creative processes modeling by means of knowledge bases*, Sofia, pp. 139-160, 1992.

[16] J. P. Crutchfield and M. Mitchell, "Evolution of Emergent Computation" Computer Science Faculty Publications and Presentations. Paper 3, 1995.

http://pdxscholar.library.pdx.edu/compsci_fac/3

[17] M. Burgin, "Inductive Turing machines," Notices of the Academy of Sciences of the USSR, 270 N6 pp. 1289-1293, 1983. (translated from Russian).

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

[19] M. Burgin and E. Eberbach, “Evolutionary Automata: Expressiveness and Convergence of Evolutionary Computation,” Computer Journal, v. 55, No. 9 pp. 1023-1029, 2012.