Anomaly Detection and Intelligent Notification

Tammy R. Fuller
Echo Messaging Systems, Inc.
Lincoln, Rhode Island US
tammy@echomessaging.com

Gerald E. Deane
Echo Messaging Systems, Inc.
Lincoln, Rhode Island US
gerald@echomessaging.com

***Abstract*—Applications and Operating Systems are programs responding to stimuli to perform tasks on computers, and these programs can be constructed in ways to minimize overall complexity of the software lifecycle, by organizing tasks into units that bring about a response and adaptation to stimuli. The response is the task's main function, the stimuli are user interaction, data source conditions, or time-based events, and adaptation is an opportunity for the task to adapt to its environment. Developing software using autonomous adaptive agents in an intelligent system minimizes overall complexity by organizing computing into reusable, scalable logical units that are by nature parallelizable. Each autonomous agent has a common core of virus resistant outer virtual casing by continually comparing its core attributes against master copies distributed via block-chain technology. Building upon research and implementation of these concepts of using intelligent systems using autonomous block-chain protected agents to rapidly create commercial-grade software over a wide variety of domains, we discuss in this paper how these concepts not only facilitate application development but can also be used for operating systems.**

Keywords—intelligent systems; autonomous agents; artificial intelligence; adaptive processing; self-organizing systems; self-adapting systems; block chain ;software lifecycle; software engineering;

I. INTRODUCTION

Creating software is a complex process of gathering requirements, designing the components, implementing the design and testing against the original specifications, followed by maintaining for a long and fruitful software lifecycle. Each step in this process requires hard won technical skill sets, coordinated by management processes to orchestrate and dovetail the myriad tasks to create software and hardware systems that solve everything from remote surgery, to mars travel, to smart weaponry, to ordering a pizza that a driverless vehicle delivers fresh and hot upon demand.

Reducing the complexity to go from *'we need a system to* ...' (fill in the blank) to having the software, hardware, user interfaces, legacy data source connections, compliance with regulations, among other things, to a functioning safe reliable system that does what it needs to do, can be achieved through design choices as we will show in this paper. However, to do this in a timely fashion where reusability promotes productivity, and expands to future capabilities of scaling based on the environment is not only expected, but required as Artificial Intelligence becomes ubiquitous, and Ambient Intelligence will be pervasive and foundational to modern living.

Through real-world experience, we tackle the challenges of creating complex applications by using autonomous adaptive agents in an intelligent system [1]. We propose that software created using this approach of adaptive agents responding to stimuli in a common framework to solve specific complex tasks minimizes complexity over other design approaches in a number of ways.

A. Waterfall

Waterfall design approach is comprised of the following steps: requirements gathering, design, implementation and test. Each step completes before the next is started. In practice, the Waterfall approach is not common due to hidden requirements revealing themselves during design, or worse during implementation. This causes the entire waterfall process to start over, resulting in delays from the original timeline.

B. Agile

To address the problems of waterfall, Agile cycles freely and often through the four basic steps in a cyclic fashion. The problem with this approach is sight of the overall process can be lost due to frequent, small-scale cycles. Hidden assumptions are revealed sooner, but larger-scale goals can be sidetracked at each cycle, causing delays or worse missing requirements.

C. Hybrid

A hybrid approach [2] where agile-type cycles are done between the first two steps of requirements gather and design, followed by a onetime waterfall-type step to the second two steps of implementation and test, has shown to improve development timelines and eliminate late-revealed requirements. The problem with Hybrid is there are no viewable results until 40-60% of the way through the entire timeline causing stress to key stakeholders.

D. Autonomous Agents/Intelligent System

Combining autonomous agents that solve key components of a design, and imbuing them with the ability to adapt to the specifics of their environment, such as the nature and amount of data they are handling, by being inherently adaptive, as well as scalable and parallelizable, results in applications that are quickly implemented, which addressed the main problem of the Hybrid design approach.

Designing software using reusable autonomous agents to implement specific applications takes a different approach were requirements are organized around triggering criteria and response actions. The process of determining how best to organize processing units is typically a challenge [3], but our approach of working in actual commercial applications provides direction on how to best segment and organize the agents.

Agents, being adaptive, will conform to their environment, such as increasing or decreasing computing capacity in response to data bandwidths. Agents, being inherently parallelizable, will scale in response to their environment, such as automatically replicating and distributing agents across a network computing facility as they become available for load balancing.

Over many years of creating commercial applications in a wide variety of settings, platforms and domains, along with a background in AI, we realized the benefits of creating software using intelligent systems. We expanded the core concepts in agents to include built-in virus protection and authenticity checks using Block-Chain technology, as well as creating agents specifically to monitor and mitigating for anomalies, which could come from spikes in data, changes in underlying hardware or networking, or any conditions and criteria where human intervention should be considered for automated systems.

Anomaly detection and intelligent notification form the core tenets of ADIN 2.0, a platform for application development that reduces overall complexity of the software lifecycle.

II. Autonomous Agents in an Intelligent Systems

Autonomous agents in an intelligent system are built from one or more agents running in a common environment. This means the agents are connected to one or more of the same data sources and/or user interfaces. Agents can be of the same type, configured differently or identically.

For example, in a notification system where the data source is a web-hosted E-commerce system, an agent can detect a new order. Configuration on the agent can determine if an agent responds to orders of a particular type, such as a corporate order vs. an individual order. At any point, more agents with expanded/refined criteria can be added to the ADIN Cell, as shown in Figure 1, which eases lifecycle maintenance.

A. ADIN Cell

An ADIN Cell organizes a set of one or more agents together by common data sources and user interfaces to define an application. All agents have 3 basic components:

- Triggering criteria - time, date, event, or any detectable digital entity from tiny to complex can define the triggering criteria
- Action - what happens when the triggering criteria is detected. A notification is issued, a record is replicated, business logic is performed, intermediate data is created. Any response is possible
- Adaptation - logging is done here for statistical analysis and any type of adaption such as adjustment of agent configuration, frequency of checking the criteria. An action about the agent itself or ADIN cell rather than the specific criteria

Agents are grouped into basic types based on the data sources they connect to and their responses and lend themselves to reusability. Some example groups are:

- Time Based - agents that check every hour, every minute, once a day or any function of time. Any data source can be used. For example, an agent whose data source is a back-office accounting system can check every day at 8am for invoices overdue by 30 days and send a friendly reminder. The same agent can also run except it's triggered on 60 days or 90 days where the reminders are more urgent.
- Geo-Location Based - agents can check for GPS locations within a geo-fenced area. One agent can monitor many areas or individual agents can monitor their own specific area. This example shows the how this approach lends itself to scalability and parallelization.
- IoT Device Based - agents can monitor a data channel where IoT sensed data is accepted. For example, an IoT enabled washing machine can send periodic status updates to a known data channel, where an agent can monitor and notify when a near-term-failure status is detected.

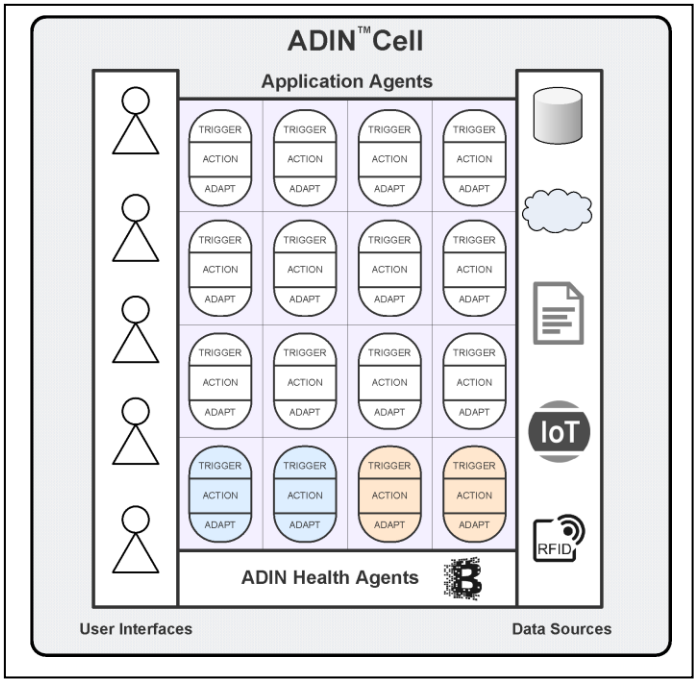

Fig. 1. ADIN 2.0 Cell components

Figure 1 shows a graphic representation of an ADIN Cell with multiple agents and how it can interface with many different types of data sources and user interfaces. The

components shown in Figure 3 are described further in this section.

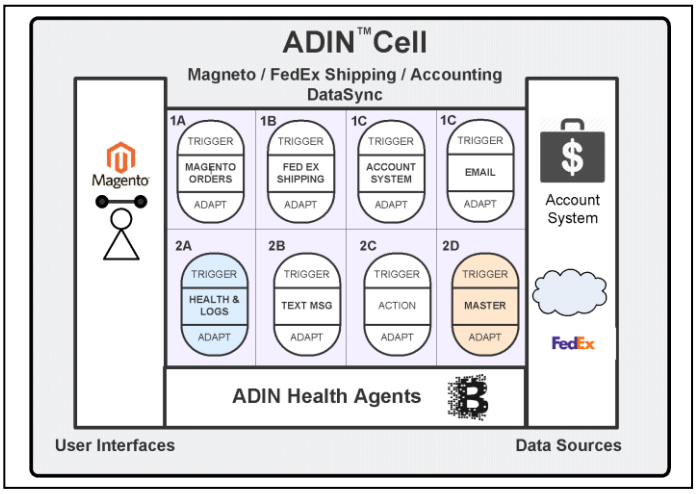

Fig. 2. Magento-FedEx-Accounting ADIN Cell

Figure 2 shows a graphic representation of an actual ADIN Cell with agents, each uniquely identified by ID number, that detects new Magento orders and replicate them into a back-office accounting system. Additionally, another agent monitors a FedEx data source and will update the orders in Magento with specific shipping information when the orders ship. Components and Agent Types

The various types of components in an ADIN Cell are represented by different icons as shown in Figure 3. There is no limit to the number of agents.

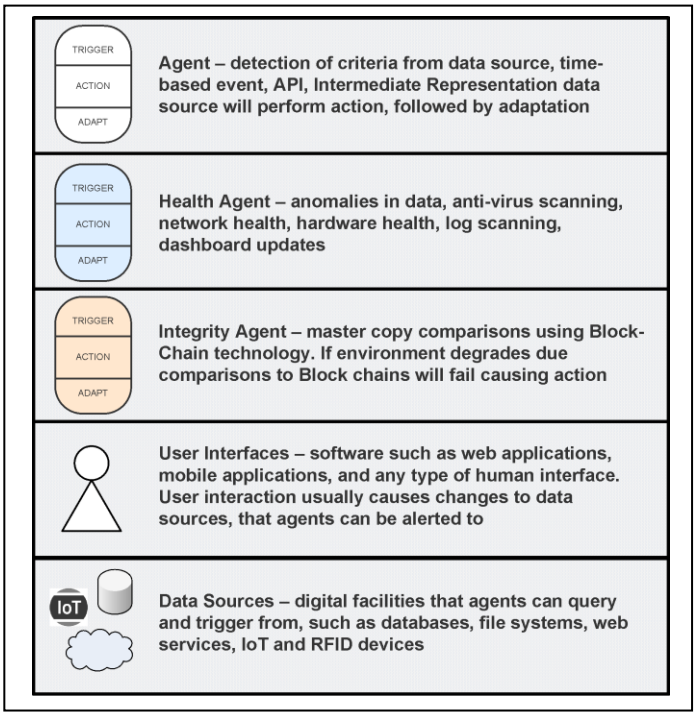

Fig. 3. ADIN Cell High level components

An ADIN cell can be installed on one computer/server, virtual server, or the agents can be distributed across many servers based on the needs of the application and the processing environment available.

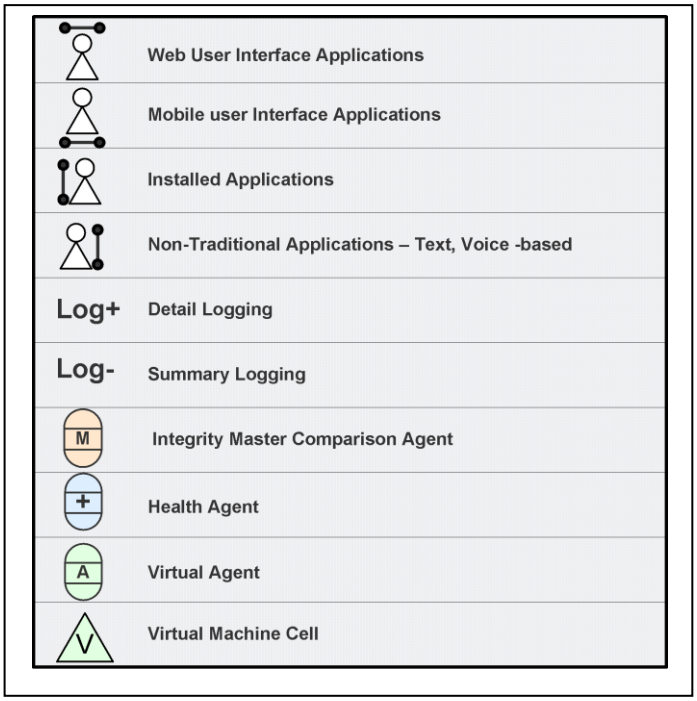

Fig. 4. Additonal ADIN Cell compoennts

Figure 4 shows there are other types of agents and components that are components to this design approaching. The user interfaces are designated into four types: web, mobile installed and non-traditional, such as, text-based or voice-based user interfaces. Figure 6 (next section) shows an example where multiple user interfaces are used as part of as greater application using common agents and data sources.

Applications can reside on single server environments with RAID 1-10, or without RAID, as well as in virtualized and non-virtualized environments. Agents and cells in a virtual environment are designated as Virtual Agents or Virtual machine cells.

B. Health and Integrity

Automated and autonomous systems work quietly and efficiently until something unexpected happens and then full insight into all processes is often required. For example, an automated datasync can for years be running where the triggering criteria is a new order in an E-commerce system, the action is to create an identical record in the back-office accounting system and the adaptation is to log for statistical processing such as timestamp, and amount of data processed.

Health agents monitor the resulting logs and its triggering criteria are detecting spikes and valleys in timestamp or data amounts in the logs (for example by looking for peaks in the second derivative or mean/mode/std deviations crossing a predetermined boundary) A valley could indicate a hardware or network failure, where a peak could indicate a denial of service attack, or a sudden increase in sales due to some external event.

In either case, health agents will detect the occurrence and can respond a number of ways, such as:

- Notify administrators via text/email/push immediately
- Shut down the ADIN cell, one or more particular agents, data sources and/or user interfaces
- Increase processing bandwidth by replicating the ADIN cell, one or more particular agents, and/or user interfaces.
- Start or stop a particular set of agents in response to the peak or valley.

Health agents can have standard predefined configurations, to particular triggering criteria, such as notify the administrator, update a heartbeat dashboard and increase bandwidth when a spike in orders is detected up to additional 25%. An additional heath agent in the same ADIN cell can trigger on the same criteria but only after 25% increase as been detected, as which an escalated notification is issued. This allows for predetermined responses to be added, and modified as a system's environment is better understood, which happens after deployment.

When hardware failure is the cause of the anomaly such as the Health and/or Integrity agents are returning a failure for one or more particular applications agents, these agents can quickly be replicated and their original copies set to dormant.

III. Agent Integrity using Blockchain

Agents can be replicated many times with configuration variations or identical copies. To maintain integrity of the data in the face of hardware failures, network failures, software bugs, failed upgrades, failed deployments or outright attacks on data and systems, ADIN Cells include at a foundational level, Integrity agents, that store and compare master copies of the agent metadata, such as configuration settings, DLL signatures, timestamps, and any data that lends authenticity and integrity that each agent is working as expected. Figure 5 shows this process.

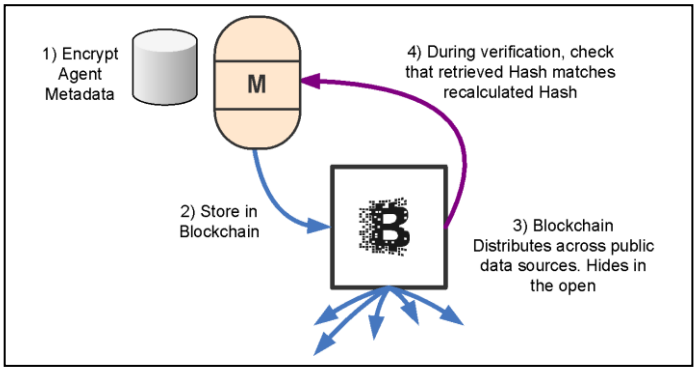

Fig. 5. Blockchain to store encrypted Agent metadata for verification

Blockchain technology is used to hold the original master copy of agent metadata. Blockchains [4] are publically available ledgers that are resistant to tampering due to the nature of being both decentralized and publically available. Blockchains were first used to represent the digital currency Bitcoin. In addition to digital currently, blockchain technology is being for holding any information that has intrinsic value such as contracts, trading records, public records such as voter IDs and birth/death certificates, private records such as wills and trusts, digital keys, and many more. The decentralized nature means that many copies of transactional data is stored in a widely distributed fashion.

To tamper with blockchain data is considered an extremely difficult task, since every copy at every location would have be tampered with exactly the same way at exactly the same time. Integrity agents continue to compare agents against the master copy to ensure that the code has not been modified by virus or any other external nefarious attack. The triggering criteria of Integrity agents are when Blockchain master copy checks don't match, at which point the master copy could reinstall based on the Blockchain metadata. In addition, notifications can be issued via text message, push alert and email, as well as health-related dashboards can be updated.

Integrity Agents are a foundational component to guard against unexpected environmental failures, as well as to act as a shield against explicit hacking-type attacks, which sadly is now commonplace. Having a built-in immune system, with predefined responses, provides insight into the anomalies that occur in automated systems, when that insight is needed.

Furthermore, by making the integrity and health agents based on the same technology as the application business logic means that the response to anomalies will happen in the same time-scale. If an application is running normally, it will be quietly processing the background. If something goes wrong, for example, a data source goes offline, at the speed at which an order is processed, the system knows an anomaly occurred and will respond immediately, verses, a human watching logs or a dashboard, or responding to an alert notification manually. Humans aren't able to respond in the same time-scale as computers, so the anomaly response must match the overall system processing speed to be most effective.

IV. Scalability

An advantage to designing and implementing complex software using autonomous agents in an intelligent system is that agent scalability is built-in by design.

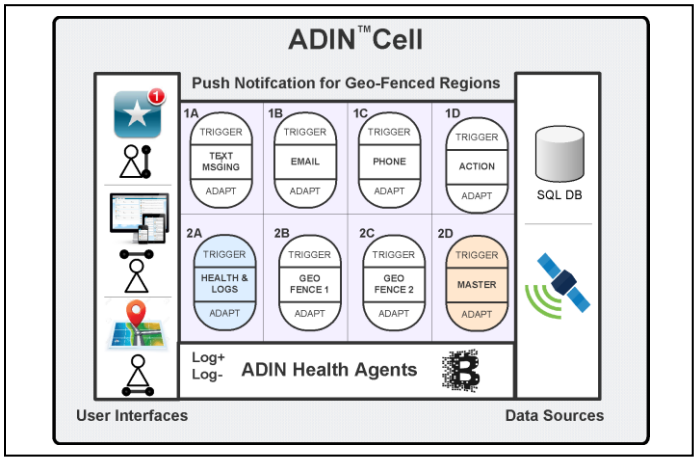

Fig. 6. Push Notification based on Geo-Fenced Regions

For example, an ADIN Cell, as shown in Figure 6, is comprised of web and mobile applications to define Push Notification Channels based on users crossing into predefined Geo-Fenced regions.

Agents can be configured to monitor all defined geo-fenced regions. Once the number of channels crosses a boundary, the agent will replicate and refine its search criteria. As usage increases, the agents' adaptation will cause them to replicate. Likewise, if usage of the push channels decrease, the agents' adaptation can cause them to go dormant until needed again.

By organizing business logic into agents based on their triggering criteria, specific response and adaptation, this approach to designing software is less complex by design. Replicating agents to response to an increase in data processing needs or to refine the search space is something that can be coded into the agents' adaptation or into a health agent and this flexibility means that the approach can be a policy level decision.

Resources are used in a scalable way as well. For example, an application can start with fewer developer resources where feature agents are developed. Even after running against live data sources, refinement of the agents or more complicated agents can be developed and added to an existing ADIN cell.

V. PARALLELIZATION

Like scalability, parallelism is built-in by nature and parallelization is achieved by the agents' adaptation or via health agents depending on requirements or preference. Hardware is better used when the underlying software components are individual processing units. One of the key functions of an operating system it to allocation the system resources as efficiently as possible. A multi-core system will be better used by agents that are already naturally segmented along processing units.

In a highly parallelized computing environment, the performance is even better, again because the work of parallelizing an application is already started at the design level by organizing processing units based on the triggering criteria and associated response and adaptation.

Fig. 7. Health Agents

Health agents (Figure 7) again can take up the task of seeking ways to improve overall performance. For example, a health agent can monitor the environment for opportunities to where adjusting the time intervals agents are checked for triggering criteria could be improved. If multiple agents are connecting to the same data sources, health agents can look for patterns and modify agent timing criteria.

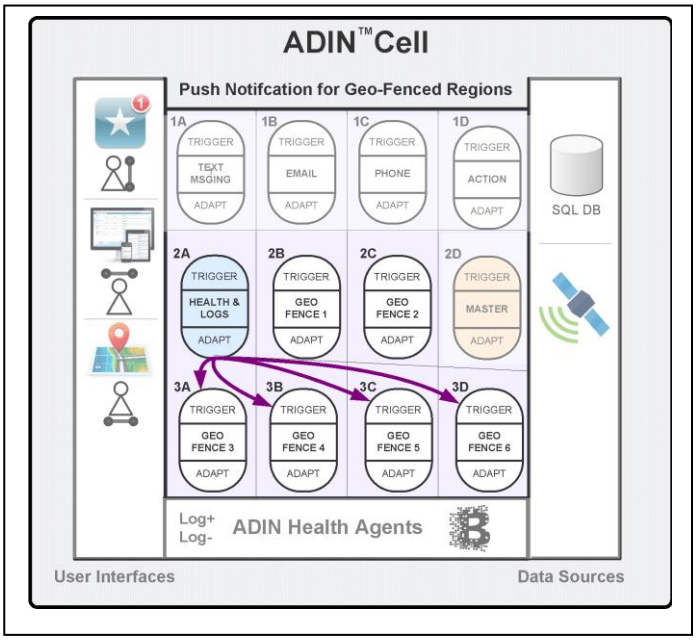

Fig. 8. Health Agents clone additional Alerts on demand

Using the Geo-Fence Push Notification application from Figure 6, parallelization can occur if expanded capacity is needed or if resources became limited due to a hardware or network failure. In both case, the Heath Agent is triggered. For example, a Health alert can monitor the number of active geo-fenced regions and can be triggered every time a new one is detected. It will replicate the basic Geo-Fenced alert from 2B in Figure 8, and will clone to alerts 3A, 3B, 3C, and 3D representing 4 new geo-fenced regions. Each alert will track and communicate with recipients detected within their region. Regions can be static in a fixed location, dynamic such as tethered to a vehicle or train, as well has may have specific altitude, such as floors 10-20 of a high rise.

A. Power Consumption Profile

A power consumption profile can be generated where for each point in time (every millisecond, second, minute, depending on preference) the power consumption is calculated that reflects how hard the Cell environment is working. As shown in Figure 9, when the power consumption crosses a threshold, this can be the triggering criteria for a Health agent that causes the cell to be replicated along with all its agents into another server or virtualized environment. The opposite can occur, when the power consumption drops below a threshold, the agents can be set to a dormant status since they are no longer needed.

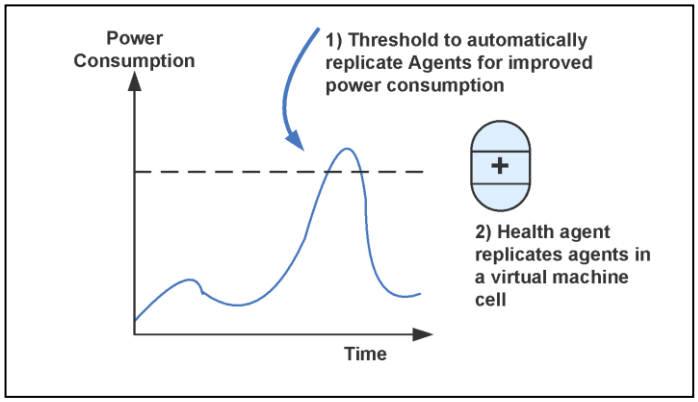

Fig. 9. Power consumption profile used by Health Agent

VI. AI Operating System

After numerous positive experiences with ADIN Cells for creating complex applications over many years, the next logical step is to consider this approach for operating system level functions. We posed the question, this can approach improve OS level performance. Over the years, hardware consistently made steady improvements to processing speed, where software, on the other hand, at all levels from application down to OS, became more and more complex. Increased processing power still resulted nonetheless, but if software was to be improved and streamlined the performance leap could be enormous.

We have dedicated efforts to investigating what an AI-based OS would entail and using this approach has a basis has a lot of advantages as outline in the previous sections. If applications were instead OS-level functions, and components were organized around the OS-level triggering criteria and its associated response and adaptation, an AI-OS would gain the same benefits in scalability, parallelization, reusability, improved resource allocation and overall reduced complexity.

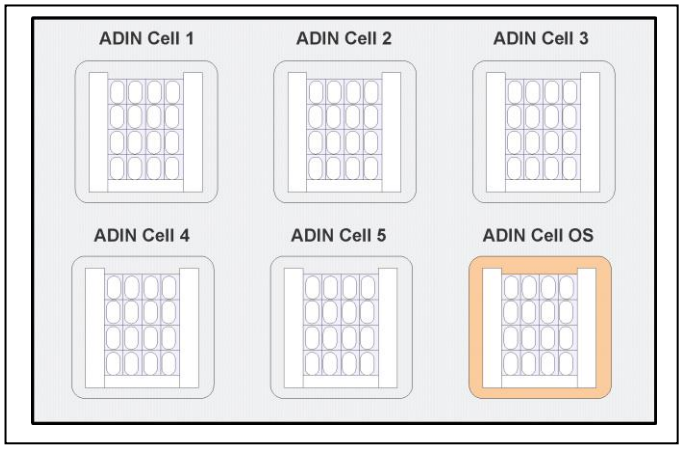

Fig. 10. Operating Sytsem level Cells and Agents

Figure 10 shows how multiple ADIN Cells could run within the same server environment where each cell in maintaining its own integrity, as well as computing / data health. An ADIN cell could (highlighted in orange) could take on additional monitoring tasks that worked across the individual ADIN applications and OS elements to provide OS level coordination. As shown at the application level, cells and agents by design can flexibly work in a single or distributed environment, OS level tasks can likewise benefit from work across hardware and networking boundaries.

VII. Impact and Future Directions

Artificial intelligence mimics human activity and historically involves huge search spaces and use of heuristics, from playing chess to understand speech to finding the defective part on an assembly line. The term 'artificial' is waning, as it should and instead both technical and non-technical people alike are realizing AI applications are highly automated, often invisible and extremely pervasive. As humans no longer do tasks that automated systems are doing instead, people nonetheless need to stay involved and assist in decision points particularly during anomalous events.

Certain events handled by automated systems comes with problems that some posit are existential to humanity. The processing rate and reaction time when something goes wrong without being detected by humans, has the potential to cause harm to life. This is the main motivation for creating an autonomous agent-based platform. Where most of the time it's used to process a tremendous amount of mundane but necessary work that humans no longer need to perform, but more and more processes are automated that have greater and greater impact on human life.

As is typical, information systems are developed faster than regulatory systems can keep up. Every designer wants to avoid unintended harm, but the rate at which automated systems are being created is extremely fast, and many fear that safety systems are not in place at a foundational level [5]. To create a system without a self-healing aspect or without the ability for mitigation in the face of the unexpected is why AI-based system can potentially cause harm. Automated systems will continue to do more but it's imperative that the health and integrity of these systems be a foundational component.

A. Applications

Future direction of the work describe here, is to continue to implement new applications where Health and Integrity checks are integral, including new systems such as Text4Tow where users send text messages to be automatically matched with nearby auto services. The agents to coordinate workflow and to do all the back-and-forth notification were already developed and in the basic Agent Library. A few key business logic relationships were created along with application specific templates. This example using this design approach took under two weeks to create a working test system that is fully functioning. The concepts for automated resource matching are transferable to other domains.

The Text4Tow Application as shown in Figure 11, uses a text-messaging based user interface where the user texts 'tow me' or 'need gas' or some other natural language command. An agent monitors for new commands and sets up the request after getting the current GPS location. Another agent is continually monitoring for new requests and matching them against open service truck vendors who are pre-registered. When a match is

found, the service truck is emailed, texted and if desired, phoned with a message saying where the person in need is and their requested service. A quick text reply by the service truck ('a' to accept, 'no' to decline) causes another agent to send out the notifications to the customer that a match has been found, and a link to track the service truck in real type.

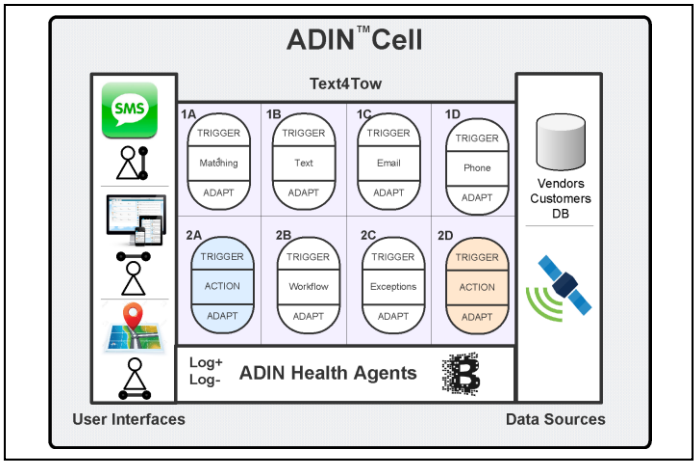

Fig. 11. Text4Tow

This entire application is based on agents monitoring triggering criteria and performing their associated actions, such as determining GPS-to-GPS distance to find a nearby match or sending out notifications. These agents are quickly configurable if more notifications or different criteria are a better fit for the application. The entire process of gathering requirements, designing implementing and testing is a much quicker process when everything is oriented around triggering criteria, and their associated actions. For this application, no new agents were required, as all were already in existence in the agent library.

B. Developing OS-level components

Developing the OS level concepts and eventually creating an entire operating system using ADIN cells is a high priority. Based on how agents can dynamically respond to the environment based on hardware, network, data and other processes, OS level components will be inherently scalable and parallelizable, and we expect that increased performance will result from this effort. Software has long depended on increases in hardware performance to allow for more and more complex computation. By creating software components for both application- and OS-level processing, using ADIN cells increases overall computational performance with no other changes is something we anticipate.

C. Parallel Architectures

Experimenting with how parallel architectures run ADIN cells is an exciting area since the types of applications developed using this approach aren't traditional parallel algorithms such as image processing where data is can be locally processed. However, parallel architectures should reveal the true power of the ADIN cell approach to designing complex, scalable software since by design the components are organized around localize data stimuli.

VIII. Conclusion

This paper talks about how we have lowered overall complexity of application development and resource usage by taking this design approach. Using a software engineering methodology based on a Hybrid method, this approach takes the best aspects from Agile, but implements in an intelligent system using autonomous adaptive agents as the core processing units. Requirements and design tasks become centered around triggering criteria and response actions, followed by an adaptation phase where criteria can be modified to best work in the current data processing environment. The result of this work is a reproducible process where highly complex reliable applications are created very quickly, taking days or weeks instead of months and years. The resources to create sophisticated software, human resources in particular, are very limited, and yet software automation is increasing at exponential rates.

Automated systems are being created so fast that there is a real danger to human safety in the event when something goes wrong, due to unknown environmental conditions, such as unanticipated data, or suddenly losing network connectivity, or incorrect implementation or deployment. These real-world scenarios happen frequently. Where automated software runs silently and often invisibly in the background, when something goes wrong, damage to data or human safety may occur before it's detected and corrected. This paper outlines how the intelligent system has at a foundational level Health and Integrity checking agents consistently monitoring itself and check for data inconsistencies to verify all authentic versions of agents are running. If any sort of hacking is detected, copying back from a known-good master copy stored using difficult to tamper blockchain technology is done immediately.

Automated systems with built-in anomaly detection and intelligent notification using an AI-based intelligent system and autonomous adaptive agents continues to show that robust, maintainable, sophisticated software can be created quickly using minimal resources, thus minimizing overall complexity over the software lifecycle.

References

[1] T. R. Fuller and G. E. Deane, “Creating Complex Applications via Self-Adapting Autonomous Agents in an Intelligent System Framework,” *2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems*, pp. 164–165, Sep. 2015.

[2] T. Fuller, “Design-Analysis Centric Method for Creating Sustainable, Stable, Complex Systems". *Proceedings of the IEEE ITSC 7th International Conference on Intelligent Transportation Systems*, Washington, D.C, Oct 3, 2004

[3] Li, B., Yu, H., Shen, Z., Cui, L., & Lesser, V. R. "An Evolutionary Framework for Multi-Agent Organizations", *2015 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'15)*, Singapore, Dec 6-9, 2015

[4] Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton and Oxford: Princeton University Press.

[5] Musk, E. (n.d.). An Open Letter to the United Nations Convention on Certain Conventional Weapons. Retrieved September 01, 2017, from https://futureoflife.org/autonomous-weapons-open-let