Use Cases for Evaluation of Machine Based Situation Awareness

K. Baclawski*, A. Chystiakova†, K. C. Gross†, D. Gawlick†
A. Ghoneimy† and Z. H. Liu†
*Northeastern University, Boston, MA USA
†Oracle Corporation, Redwood City, CA USA

***Abstract*—Situation awareness (SA) is important both for human decision making and for complex automated decision making. The presumption is that improving the accuracy of SA will lead to better decisions. While there has been significant research on measuring the accuracy of human SA, there has not been as much work on machine-based SA. Unlike humans, complex systems have many levels of decision making that may operate independently and may run at very different timescales. The accuracy of SA for each decision making process, determined in isolation, need not contribute to overall system performance. Moreover, achieving more accurate SA may require devoting resources that are disproportionate to the benefits. We propose that one should focus on the net value of SA to the system rather than simply on the accuracy. In this article, we present some use cases for determining the value of machine-based SA. The purpose is to illustrate how one can quantitatively evaluate SA so that one can optimize important issues for automated decision making processes such as system performance and stability.**

***Keywords*—situation awareness; evaluation; use cases; control theory; stability**

I. Introduction

Situation awareness (SA) the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future [1]. Humans are constantly acquiring and using SA in everyday activities. For example, they have to judge and react to social situations, traffic situations, and a variety of situations in their workplace. SA is especially important for complex decision making in high stress situations. Loss of SA was the root cause of several high-profile disasters.

The oldest and most famous model of situation awareness and decision making is Boyd's observe-orient-decide-act loop (OODA loop) [2]. The Boyd loop models human behavior that is mostly performed unconsciously: humans observe and interpret their environment using their senses and based on their focus, they try to understand the meaning of their observation and interpretation, they decide what should be done, and they execute the decision. The observations are biased by their interpretation and classifications. The decisions are mostly based on experience and are often intuitive and less reflective. For example, in traffic situations, and in many industrial applications, there is often no time for a detailed analysis.

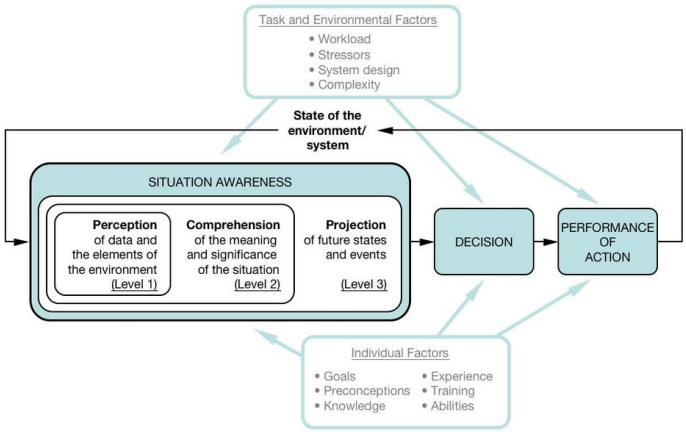

Fig. 1. Endsley's model of SA based on [1], [3]. Drawn by Dr. Peter Lankton, May 2007

Endsley elaborated on the OODA loop model with the loop shown in Figure 1. However, neither the OODA loop nor Endley's loop specify either the processes or the data involved in SA and decision making [4, p. xiii].

Cognitive architectures are formal models of cognition that are formalized sufficiently to be the basis for a computer program. Example of cognitive architectures include Adaptive Control of Thought - Rational (ACT-R) [5], [6], [7], [8] and Soar [9]. These cognitive architectures have process and data models, explicitly expressed in their implementations. While cognitive architectures are machine based and could be used for decision making tasks, their primary purpose is refine theories of cognition. In ACT-R, for example, cognition unfolds as a succession of production firings intended to match the behavior of human cognition. In other words, it models cognition (at least at the lowest level) as kind of feedback loop. However, it does not appear that ACT-R is concerned with whether their decision loop is either efficient or stable. Indeed, doing so would detract from the purpose of modeling human cognition as accurately as possible.

If sensors and computers are used we propose a different model: KIDS with its ingest-classify-assess-resolve-enact

(ICARE) loop [10]. Sensors are used to measure raw data; this data are ingested into a computer system. The computer system stores the data for long term documentation and applies models using the machine learning (ML) technology of choice. The ML technology will improve and compact the data and also identify abnormal conditions; this includes a distinction between sensor and asset issues. While the data so far are usable for computers, they are not usable for humans; there are typically just far too many raw data elements. Therefore KIDS creates a compact representation using domain specific languages. This transformation results in the most likely classification(s), assessment(s), and resolutions(s) based on the most similar cases (representing experience) and/or reasoning using rules and/or models. In effect, KIDS provides the processes and data structures that are missing from the Endsley model, although only for machine SA, not human SA.

KIDS associates to each instance of an abnormal conditions an object called KIDS instance; the underlying data triggering the abnormal conditions are part of the instance. KIDS instances can be related to each other using hierarchical and/or graph models; e.g., a hospital stay is a collection of many nested KIDS instances. These KIDS instances can be accessed using domain specific languages (and dialects); they represent the shared experience of a group of users. The access can focus of an individual instance, on similar instances, or on (complex) analytics. Accessing and adding new KIDS instances - especially when new approaches are used - leads to the new methodology of the evolution of the shared knowledge and experience with minimal additional work.

Whether to use the Endsley model or KIDS depends on the particular use case. The Endsley model is preferable when there are few or no measurements and/or reliable process and data models do not exist. KIDS is preferable in data rich environments with complex correlations and the need for provenance and explainability. In many cases, a combination of both methods is beneficial. KIDS refines significantly data gathering, data analysis, provenance as well as abnormal condition detection. Once an abnormal condition has been detected it very much follows the Endsley model. This approach enables KIDS to allow automated as well as human decision making; since automated decisions can be explained using domain specific languages.

While human SA is very similar to machine SA, there are significant differences. When measuring human SA performance, one would not normally use a financial performance measure. Although such a measure is meaningful for human SA, it is not socially acceptable when human lives are at stake. The infamous "Pinto Memo" is an example [11]. In this memo, it was computed that the cost of correcting a defect of the Ford Pinto was 3 times the social cost of the burns suffered by individuals as a result of the gas tank exploding. Unfortunately, the memo failed to consider the cost to Ford of the negative publicity that the memo caused when it was made public.

By contrast, for machine SA, a cost-benefit analysis is usually the main measure of performance. If the cost of achieving a better quality of SA is greater than the benefit so that the net benefit is negative, then it would be more beneficial to employ a lower quality SA whose cost and benefit are both lower than the better quality SA but whose net benefit is maximized. There are other differences as well. Machine decision making processes can, in principle, manage far more information than even a team of humans; such processes can operate at a far more rapid rate, with much better response time; and machines do not make the same kinds of mistakes that humans do. On the other hand, machines are not as flexible as humans, especially when confronted with new situations, goals and kinds of data. For this reason, one would expect that humans will continue to be "in the loop" at least at some level, and the system must provide for this kind of interaction [12]. However, machines are improving in this regard, and we proposed an architecture for self-adaptation using flexible data structures and processes [13].

Measurement of SA is a well studied area of research. The book by Endsley and Garland [3], especially the articles [14], [15] by Endsley, give an excellent survey of the state of the art and current research issues. However, this work differs from our own in several ways:

1) Existing SA measurement research deals almost exclusively with human SA. To the extent that machine SA is mentioned, it is subsidiary to human SA.
2) More importantly, measurement is not the same as evaluation. SA measurement is concerned with the accuracy of SA. Evaluation of machine-based SA, by contrast, is concerned with the net value of SA.

The purpose of this article is to explore some of the ways that can be used for evaluating machine-based SA in various contexts so that one can better understand the tradeoffs involved in efforts to use SA to improve system performance and stability.

We cover three use cases. In Section II, we consider an example of a commonly occurring failure of SA that has substantial costs for manufacturers of components. The problem is that components can be incorrectly determined to be faulty by a customer when, in fact, they function correctly. In Section III, we discuss how independent decisions in a chain of processes can result in instabilities and loss of performance even when the individual decision making processes are optimum and, in principle, have accurate SA when each is considered on its own. The last use case, in Section IV, discusses a scenario in which the overall system has hierarchically nested decision making processes that occur at different timescales. As with chains of processes, optimizing decision making at one level may conflict with other levels. We end the article with some conclusions and propose some

future work.

II. No Trouble Found

The first use case is a problem known as No-Trouble-Found (NTF). In the literature, it is also called No-Faults-Found [16]. The NTF problem is that components used in application areas, such as automobiles, electric utilities, and manufacturing, have mechanisms for indicating component failure. The failure is typically advertised with an alarm. When an alarm is raised, the component may be replaced at little or no cost under the terms of a warranty or service contract. The component that raised the alarm is returned to the supplier and tested in their laboratory. Remarkably, as much as 25% to 70% of the time, the returned component operates correctly when tested. The NTF rate depends on the particular industry segment and product line. The cost of replacing NTF components has been estimated as about $2B per year in the industrial sectors of transportation, utilities and manufacturing. Needless to say, there is a financial advantage for reducing the NTF problem. Furthermore, the NTF problem may be regarded as an example of a loss of SA, since the true status of the component and the perception of its status are inconsistent.

Sun Microsystems performed in-depth root-cause analyses of NTFs in electronic systems and found that the leading causes of NTFs in electronic components are the following [17]:

- Transient/Intermittent Faults
- Threshold Limits on Noisy Physical Variables
- Sensor Degradation Events
- Human Errors During Testing and Diagnosis

The first three of these causes occur at the customer side, while the last cause is at the supplier side. This use case will focus on the supplier side testing to avoid discarding a component that is not faulty, but similar techniques could be applied to the problem of evaluating SA for any of the causes.

On the supplier side, a returned component is tested in the laboratory to determine whether the component is an NTF. Such testing can be costly, so it is only useful if the savings outweigh the costs. To make this more precise, suppose that there are several laboratory tests of a component, $T_1, T_2, \ldots, T_n$, with increasing costs $C_1 < C_2 < \ldots < C_n$, For each test T_i, there are three possible outcomes:

- the component is defective;
- the component is working properly; and
- the test cannot determine the status of the component.

Assume that we have empirically determined that the probabilities of each of the three outcomes of the test T_i are p_i, q_i, and r_i where $p_i + q_i + r_i = 1$ and that the tests are statistically independent. If the value of the component is V, then the average benefit of performing T_i by itself is $b_i = q_i V$ because there will be a gain of V with probability q_i. The net benefit of performing T_i is then the difference $f(i) = b_i - C_i$. So if only one test is to be performed, then one should select the test T_i with the highest net benefit $f(i)$.

If multiple tests can be performed then one may be able to achieve a higher net benefit. An example of three tests is shown in Figure 2, and the test sequence is described in more detail below. Suppose that we perform the tests in the order $\{i_1, i_2, \ldots, i_n\}$, stopping when a test determines that the component is either defective or working properly. Then the first test is always performed, the second test is performed only if the first test is indeterminate, and so on. The total net benefit is then $f(i_1) + r_{i_1}(f(i_2) + r_{i_2}(f(i_3) + \ldots + f(i_n)))$ or

$$f(i_1, i_2, \ldots, i_n) = \sum_{j=1}^{j=n} (\prod_{k=1}^{j-1} r_{i_k}) f(i_j)$$

One can also write this formula recursively as follows:

$$f(i_1, i_2, \ldots, i_n) = f(i_1) + r_{i-1} f(i_2, \ldots, i_n)$$

One can maximize the net benefit of performing a sequence of tests by computing the maximum value of $f(i_1, i_2, \ldots, i_n)$ over all possible sequences. If a single test can be performed more than once, and the results are statistically independent, then the sequences can include duplicate tests, and there will be infinitely many sequences to consider.[1] Another complication is that tests may not be statistically independent.

A sequence of three tests is illustrated in Figure 2. The individual tests are performed as follows:

- Exercise the component using archived time series data.
- Collect and store the results of exercising the component.
- Analyze the results by classifying any anomalies in the component output using statistical machine learning techniques.
- Perform a statistical test to decide whether one of the following has occurred for the component:
  - True Failure
  - True No Trouble Found
  - Inconclusive
- The action to be performed is determined by the result of the statistical test as follows:
  - If True Failure, then send to scrap/recycling.
  - If True NTF, then return the component to the stock in the warehouse.
  - If Inconclusive, then perform the next test.

An example of a specific test sequence for detecting NTF components that is used in industry is given in [10].

[1] Indeed, the sequence that maximizes the net benefit can be infinite.

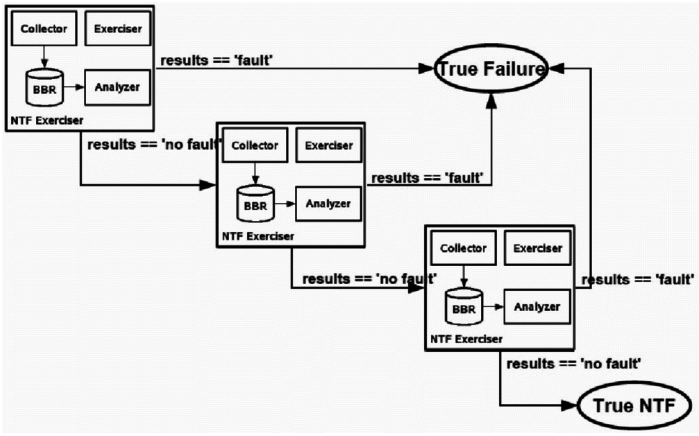

Fig. 2. Sequential Hypothesis Flowchart for Electronic Systems and Components Evaluated as "Suspect NTFs" from [18]

III. Bullwhip Effect

The *bullwhip* or *Forrester effect* is the phenomenon in business economics in which forecasts overcompensate in response to shifts in demand, resulting in increasing swings in supply [19]. While this phenomenon can occur in a simple supply and demand situation, it becomes much more pronounced when there are multiple linked supply/demand links forming a supply chain. The reasons for this phenomenon are complex, and it is often attributed to imperfect information by the participants. However, it is known that even when people have perfect information, they have difficulty achieving the theoretical optimum performance of the supply chain.

For automated decision making of a linked chain of services, the bullwhip effect can be mitigated, in principle, by ensuring that all of the steps in the service chain are fully informed. Unfortunately, the components of a service chain may not allow for sharing of information and control. Modern software engineering emphasizes separation of concerns, low coupling and modularity to achieve correct operation, which is exactly the kind of situation in which bullwhip effects may occur. For example, if a file server detects what appears to be a trend, it may retrieve data in anticipation of future requests. If these future requests do not occur, the effort was wasted, and the file server may change to a mode where it does not anticipate future requests, which causes a loss of responsiveness. Eventually the file server returns to the previous mode, and the situation can repeat. If the file server and the program using the file server both had full knowledge and control over each other, then one could, in theory, mitigate this phenomenon. However, this would be a serious violation of modern software engineering practices.

Linked chains of services are becoming much more popular. A common style for such linked chains is the service-oriented architecture (SOA). This is a style of software design in which application components communicate over a network using standard protocols. The Open Group (www.opengroup.org) is a standards body that develops and encourages standards for SOA. When the SOA services are very small and the protocols lightweight, the style is called a microservices architecture [20]. As these new architectural styles become more popular, bullwhip effects may also become more common. Since the services are, by design, independently developed, SA for the many services can become inconsistent with one another. Good SA in some of the services could easily be overwhelmed by poor SA in others.

Another issue is the need for ensuring that the service chain is stable in the sense of control theory.[2] A system is stable if the effect of a small perturbation will eventually dissipate. A system is unstable if a small perturbation will cause the system to oscillate forever without dissipating or to exhibit increasingly large deviations from normal behavior. One could have stability even if a small perturbation causes large transient effects as long as the effects eventually disappear. Accordingly, in practice, it is also necessary to control transient effects. Unfortunately, modern software engineering practices seldom consider issues such as stability or transient effects. The SOA architecture could make this even worse because of the large number of services that must coordinate with each other.

A commonly suggested approach for preventing bullwhip effects in supply chains is to share information between the services in order to synchronize them [21]. This is an effective technique for supply chains, although this can be a substantial effort as typically the supply chain organizations will need to be restructured and individuals will be re-skilled [22].

Using this same approach for SOA process chains would generally require substantial amounts of reprogramming because the services would not only need to provide internal information about their decision making processes but would also need to make use of the new shared information from other services. In addition, the services would have to give up some of their autonomy. So while the synchronization approach is likely to be effective at reducing bullwhip and other coordination problems, it is not very realistic, in practice. A more realistic approach is to introduce controllers to ensure stability.

To show how this can be done, we analyze a chain of services. We presume that time is discrete, with each unit of time representing the time for a service to fulfill each request. Because time is discrete, the z-transform can be used to analyze the system [23]. We model each service approximately as a Proportional-Integral (PI) controller, where the input and output are changes to service levels[3]. The transfer function for

[2]To be more precise, one must ensure that the most relevant notions of stability hold for the system, since there are many stability concepts in control theory.

[3]The most general controller architecture in current use is the PID controller. For simplicity, we have omitted the Derivative term.

a PI controller is $(K_P + \frac{K_I z}{z-1})(\frac{1}{z-1})$, where K_P is the coefficient of proportional control and K_I is the coefficient of integral control. One can regard K_P as the amount of direct response to the requested change in service level. The coefficient K_I is a response to the history of changes in the past. If a service responds exactly to the requested change, then $K_P = 1$. If a service does not "remember" the past, then $K_I = 0$.

Of course, an actual service will not be exactly equivalent to a PI controller. To deal with the behavior of a service that is not simply a controller, we introduce a factor G which bounds the additional behavior. The resulting model is then a form of *worst case analysis* which will be valid so long as the additional behavior remains bounded within the specified factor.

The transfer function for a chain of n services is the product

$$F(z) = \prod_{i=1}^{n} (K_P^{(i)} + \frac{K_I^{(i)} z}{z-1})(\frac{1}{z-1})G_i$$

The closed-loop transfer function is then

$$H(z) = \frac{F(z)}{1+F(z)} = \frac{R(z)}{(z-1)^{2n} + R(z)}$$

where

$$R(z) = \prod_{i=1}^{n} ((K_P^{(i)} + K_I^{(i)})z - K_P^{(i)})G_i$$

The system will be stable if the (complex) poles of $H(z)$ are inside the unit circle in the complex plane. However, even if all poles are inside the unit circle, as poles get closer to the unit circle, the systems will take longer to "settle down" after a disturbance. If a pole is on or outside the unit circle, then the system is unstable, and bullwhip effects can occur.

If each service simply responds independently to its requests, then it is likely that the system will be unstable. To prevent instability, one must "tune" the control coefficients so that the poles are inside the unit circle. For example, one can "throttle" the response of a service to reduce K_P for that service. The service itself might have a parameter setting for this purpose; but if it does not, then the service could be wrapped in a scheduler module that modifies G for the service.

The constant G is only an estimated bound. If the actual service exceeds the bound, then instability could occur. The cost analysis would estimate the probability $P(G)$ of the bound G being exceeded, possibly from historical records; as well as the cost CI associated with instability; and the cost $CT(G)$ of throttling the service as a function of G. The total cost is then $P(G)CI + CT(G)$, and one can choose G so as to minimize the total cost.

The bullwhip effect for supply chains has been analyzed quantitatively using proportional controllers, although not with z-transforms, in [21], where it was mentioned that several studies show that order rate stability tends to improve for proper tuning of the proportional controllers. So it appears that even in economics it would be useful to use control theory techniques such as the ones we developed above.

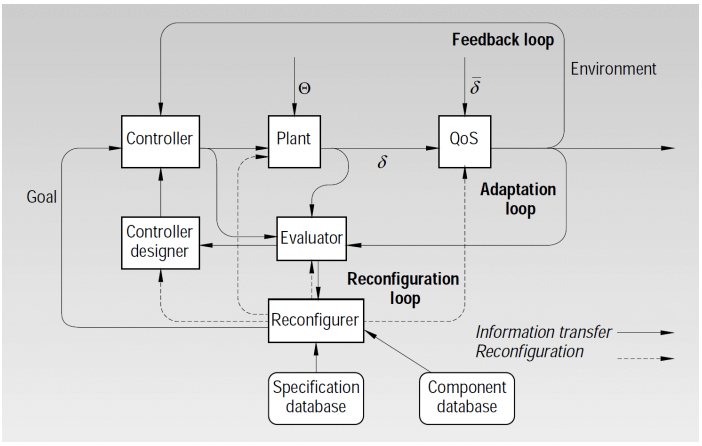

Fig. 3. Self-Controlling Software Model from [24]

The Self-Controlling Software Model (SCSM) organizes the various modules of a system so that it can automatically tune control coefficients to ensure desirable characteristics such as stability. SCSM can also restructure the system to adapt to changing circumstances [24]. The architecture of SCSM consists of three loops as shown in Figure 3. These loops operate at different rates.

1) The *feedback loop* is the fastest and usually simplest loop. It focuses on rapid responses. As a result, it generally has relatively simple control algorithms that apply control parameters to feedback from the Quality-of-Service subsystem. This would most likely be the level at which the process loop of a cognitive architecture such as ACT-R would run.
2) The *adaptation loop* is concerned with ensuring that the feedback loop is running at an appropriate performance level. It modifies the process parameters of the feedback loop when the Evaluator determines that the performance is not adequate or that the performance is deteriorating. However, this loop would not change the structure of the system. This loop is especially important in the case of the bullwhip effect as discussed above.
3) The *reconfiguration loop* is available for more drastic and costlier adaptations. This generally involves making structural changes to the system. Modifying the individual modules in a linked chain of services may be necessary for dealing with this effect. For example, it might be necessary to add scheduler modules to throttle services to help ensure stability.

A combination of SCSM and KIDS for self-adaptive situation management has been developed in [13].

IV. Cloud Services

Cloud computing is a popular technology for sharing computing resources that is reminiscent of a public utility. When a company provides its own computing resources, it must not only purchase the resources but it must also provide for the personnel and infrastructure necessary for running the computing resources. To ensure that the company has adequate resources it must provision sufficient resources to handle the worst case demands. The worst case might not even be known to the company which can result in inadequate resources at peak times. On the other hand, if peak demand is known, the computing resources will be underutilized nearly all of the time.

To mitigate these problems, large technology-oriented companies organize their resources using cloud computing. Smaller companies and companies that are not concerned with computing technology will use the services of a cloud company. A cloud company will generally have a large number of clients around the world. As a result, fluctuations in demand by the various clients will tend to cancel one another statistically. In addition, the cloud company can afford to have trained staff available continuously, which may not be cost effective for the client companies.

When a client company contracts for services with a cloud company, it signs a service level agreement (SLA) that specifies the minimum level of computing service that the cloud company is required to provide to the client. An SLA will generally specify penalties that the cloud company must pay the client if an SLA is violated. Consequently, cloud companies have a financial incentive to ensure conformance to all SLAs. For this reason, cloud companies continuously monitor performance metrics to detect SLA violations as soon as they occur so that their impact can be minimized and to predict the possibility of an impending SLA violation so that it can be avoided. In other words, cloud companies must maintain situation awareness. While this kind of SA could be done by humans, it is far more efficient and responsive to automate it as much as possible.

While client requirements are, in principle, specified by their SLA, in practice the actual resource usage is much lower. There are strong financial incentives for a cloud service company to plan the allocation of resources so as to minimize the difference between provisioned and used resource capacity. In other words, the company wants to ensure that the available resources satisfies the needs of the clients with no SLA violations, while also minimizing the cost of hardware installation and support. Unfortunately, capacity planning is difficult because the behavior of client programs are constantly changing as a result of business needs, such as introducing or phasing out products, sales campaigns, software upgrades, and process migration. As a result, the cloud becomes a stochastic system with unpredictable changes.

Because of the unpredictable nature of client behavior, instabilities such at the bullwhip effect in Section III have been observed for cloud services. Moreover, the instabilities for cloud services are much more complicated because there are many classes of resources to be managed, including: processor time, memory, network bandwidth, storage, and standard services such as database servers. The techniques of Section III can be generalized to multiple resources using linear algebra (matrix) methods [25]. This is known as "multivariable feedback control." The transfer function $T(z)$ of the closed-loop system is now a matrix function. If there is a pole of the determinant $det(T(z))$ that is on or outside the unit circle, then the system is unstable. However, the absence of such a pole does not guarantee stability due to cancellation effects that cannot occur in the one-variable case. Nevertheless, there are more sophisticated criteria that can guarantee stability [25].

Another complication of cloud services is that the resources that must be allocated are often arranged in a hierarchy with different levels being allocated at different timescales. For example, compute services consist of physical machines, each of which runs a collection of virtual machines; and each virtual machine runs a collection of threads. In addition, the physical machines may be organized into several levels of groups of machines.

A further complication for dealing with issues such as bullwhip effects in cloud services is that the providers have no legal right to examine or to control any of the internals of client software. The key to the predictive diagnosis solutions lies in the representation of a relatively high-level state model of the system that is amenable to estimation and update of the states from low-level events and measurements [26]. The state models characterize the normal operation of the client software and forms the baseline for detecting anomalies. In effect, this is what scientists do when they employ the scientific method. A model of a system is constructed by observing the system. The model is then used to predict future behavior. If subsequent behavior does not fall within the range of normal behavior for the model, then it has been falsified, and a new model must be constructed, usually by modifying the old one.

Having found a model for the client software, one can then apply the techniques of Section III, generalized to the multivariable case, as discussed above. A machine learning technique such as neural networks or MSET [27] can be employed to monitor the client software to determine whether the model is still valid. If the model is no longer valid, then either a new model can be developed or, in more extreme cases, the system can be restructured.

V. Conclusion

In this article we examined the evaluation of machine based SA, contrasting it with the more traditional human SA. We did this by presenting three use cases. Section II dealt with the NTF problem, although the formalism developed in this section would apply to any collection of independent tests of a component.

In Section III we considered a number of effects that can arise when independent software components interact with one another. Because the components cannot directly access or control each other, the system can lose SA. The interaction can be between two or more components operating concurrently or between a component at one time and the same component at a later time, as in a loop. One can deal with these problems by employing methods from control theory.

We ended in Section IV with the example of cloud services. This is a more extreme example of the situation in Section III. In this case, the cloud service must provide a variety of resources to the clients, even though it not only has no access to or control of a client program, but it also does not even know what purpose the client program might have. To deal with this, the cloud server should construct a high-level empirical model of the client program by observing its behavior. Then use this model to predict its future behavior.

VI. Acknowledgments

We wish to acknowledge the continuing support of Oracle.

References

[1] M. Endsley, "Measurement of situation awareness in dynamic systems," *Human Factors*, vol. 37, no. 1, pp. 65–84, 1995.

[2] J. Boyd, "Destruction and creation," U.S. Army Command and General Staff College, Tech. Rep., September 3 1976, Available at `http://bit.ly/1aAje2`.

[3] M. Endsley and D. Garland, *Situation Awareness, Analysis and Measurement*. Mahwah, NJ: Lawrence Erlbaum Associates, 2000.

[4] S. Banbury and S. Tremblay, *A cognitive approach to situation awareness: Theory and application*. Aldershot, UK: Ashgate Publishing, 2004.

[5] J. Anderson and C. Lebiere, "A connectionist implementation of the ACT-R production system," in *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*. Mahwah, NJ: Lawrence Erlbaum Associates, 1993, pp. 635–640.

[6] ——, *The Atomic Components of Thought*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1998.

[7] J. Anderson, D. Bothell, M. Byrne, S. Douglass, C. Lebiere, and Y. Qin, "An integrated theory of the mind," *Psychological Review*, vol. 111, no. 4, pp. 1036–1060, 2004.

[8] J. Anderson, *How can the human mind occur in the physical universe?* New York, NY: Oxford University Press, 2007.

[9] J. Laird, *The Soar Cognitive Architecture*. MIT Press, 2012.

[10] K. Baclawski, E. Chan, D. Gawlick, A. Ghoneimy, K. Gross, Z. Liu, and X. Zhang, "Framework for ontology-driven decision making," *Applied Ontology*, vol. 12, no. 3-4, pp. 245–273, 2017.

[11] E. Grush and C. Saundy, "Fatalities associated with crash induced fuel leakage and fires (report)," Ford Environmental and Safety Engineering, Dearborn, Michigan USA, Tech. Rep., 1973, Available at `http://bit.ly/2JLYIVE`.

[12] K. Gross, K. Baclawski, E. Chan, D. Gawlick, A. Ghoneimy, and Z. Liu, "A supervisory control loop with prognostics for human-in-the-loop decision support and control applications," in *IEEE Conference on Cognitive and Computational Aspects of Situation Management*, 2017, Available at `http://bit.ly/2fPlG45`.

[13] K. Baclawski, K. Gross, E. Chan, D. Gawlick, A. Ghoneimy, and Z. Liu, "Self-adaptive dynamic decision making processes," in *IEEE Conference on Cognitive and Computational Aspects of Situation Management*, 2017, Available at `http://bit.ly/2fOG9G2`.

[14] M. Endsley, "Theoretical underpinnings of situation awareness: A critical review," in *Situation Awareness Analysis and Measurement*, M. Endsley and D. Garland, Eds. Mahwah, NJ: Lawrence Erlbaum Associates, 2000, Available at `http://bit.ly/2IRRe24`.

[15] ——, "Direct measurement of situation awareness: Validity and use of SAGAT," in *Situation Awareness Analysis and Measurement*, M. Endsley and D. Garland, Eds. Mahwah, NJ: Lawrence Erlbaum Associates, 2000, Available at `http://bit.ly/2UCtl5k`.

[16] "Big Trouble with "No Trouble Found" Returns: Confronting the High Cost of Customer Returns," 2016, Available at `http://bit.ly/2vO5QZD`.

[17] K. Gross and K. Whisnant, "Process for resolving "no trouble found" server products," October 27 2009, united States Patent No. 7,610,173 Assigned to Sun Microsystems, Santa Clara, CA.

[18] E. Chan and K. Baclawski, "JVM seasonal load trending," Oracle, Tech. Rep., 2016.

[19] J. Forrester, *Industrial Dynamics*. Cambridge, MA: MIT Press, 1961.

[20] L. Chen, "Microservices: Architecting for continuous delivery and devops," in *The IEEE International Conference on Software Architecture (ICSA 2018)*, 2018.

[21] E. Ciancimino, S. Cannella, M. Bruccoleri, and J. Framinan, "On the bullwhip avoidance phase: The synchronised supply chain," *European Journal of Operational Research*, vol. 221, pp. 49–63, 2012.

[22] D. Anderson and H. Lee, "Synchronized supply chains: the new frontier," in *Achieving supply chain excellence through technology*, D. Anderson, Ed. San Francisco: Montgomery Research, 1999, pp. 112–121.

[23] M. Hazewinkel, "Z-transform," in *Encyclopedia of Mathematics*. Springer Science+Business Media B.V. / Kluwer Academic Publishers, 2001, Available at `http://bit.ly/2Xe5BWd`.

[24] M. Kokar, K. Baclawski, and Y. Eracar, "Control theory-based foundations of self-controlling software," *IEEE Intelligent Systems and their Applications*, vol. 14, no. 3, pp. 37–45, 1999.

[25] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and design*. John Wiley and Sons, August 29 2001.

[26] E. Chan, D. Gawlick, A. Ghoneimy, and Z. Liu, "Situation aware computing for Big Data," in *Workshop on Semantics for Big Data on the Internet of Things (SemBIoT 2014), 2014 IEEE International Conference on Big Data*, Washington DC, October 27–30 2014.

[27] K. Gross, K. Whisnant, and A. Urmanov, "Prognostics of electronic components: Health monitoring, failure prediction, time to failure," in *Proc. New Challenges in Aerospace Technology and Maintenance Conf. 2006*, Suntec City, Singapore, Feb 2006.