Transactions of the SDPS:
Journal of Integrated Design and Process Science
25 (X), 2021-XX
DOI 10.3233/JID-210013
http://www.sdpsnet.org

Integrating AI Microservices into Hard-Real-Time SoS to Ensure Trustworthiness of Digital Enterprise Using Mission Engineering

Alvin C. Murphy[a*] and James D. Moreland Jr[b]

[a] *Department of Engineering Management and Systems Engineering, Old Dominion University, Norfolk, VA, USA*
[b] *Department of Engineering Management and Systems Engineering, Old Dominion University, Norfolk, VA, USA*

Abstract Due to the increased complexities and operating speeds of today's and tomorrow's system-of-systems (SoS) architectural configurations for digital enterprises, the design of new domain architecture management systems is required. A key element of these new designs will be the incorporation of Artificial Intelligence (AI) microservices to provide dynamically containerized and orchestrated service capabilities within a lightweight interoperability fabric with the ability to operate in hard-real-time environments. Each containerized AI microservice exposes an independent, programmable function, which enables it to be easily reused, evolved, or replaced without compromising interoperability across critical mission essential functions to execute mission threads. In addition, embedded in this design needs to be a trust management layer to enforce reliable messaging and trust amongst the actors. This paper provides a framework for planned research and demonstrates the feasibility of microservices using a representative simple problem to demonstrate the application of the framework. Early positive analysis results using AI microservices within an SoS environment shows that the 500 milli-second (ms) threshold for latency can be met.

Keywords: Microservices, hard-real-time, artificial intelligence, combat management system, mission threads

1. Introduction

The Department of Defense (DOD) is pursuing an aggressive software development program, called the DOD Enterprise DevSecOps (software development, security, and operations) Initiative. This effort is focused on bringing automated software tools, services, and standards to DOD programs so that software applications can be created, deployed, and operated in a secure, flexible, and interoperable manner. The DevSecOps effort harnesses so-called software containers, a dedicated repository of code, and solutions that are secure and compliant with the Federal Risk and Authorization Management Program, or FedRAMP, and National Institute of Standards and Technology (NIST) criteria. The DOD initiative also utilizes Kubernetes, the Google-designed open-source container orchestration tool for automatically deploying and managing software containers (AFCEA, 2019). While these technologies are well known and understood

* Corresponding author. Email: amurp003@odu.edu. Tel: (+1)540-373-0686

 Published by IOS Press

in the commercial industry, they have not been used to date within surface combat systems. Prototype research is needed to verify and validate that it is possible to synthesize cloud computing and open source technologies to realize a microservices architecture for a hard-real-time deterministic Combat Management System (CMS). An example of past surface combat system research to assess commercial technologies is provided in (Moreland Jr. *et al.*, 2014).

A CMS supports the team in the Combat Information Center (CIC) of a naval vessel with its tactical work. Basically, this means the continuous execution of the stages of information processing in the naval tactical domain and involves the building of a situational picture of the surroundings of the ship, an understanding of the situation (including an extrapolation into the future), and the possible undertaking of offensive and defensive actions (Arciszewski et al., 2009). In military applications, this is known as the Observe, Orient, Decide, and Act (OODA) loop. The loop is similar to the domain agnostic information processing model of (Endsley, 1987) and of (Parasuraman *et al.*, 2000). The loop can be further subdivided into distinct tasks like correlation, classification, identification, threat assessment, and engagement. Correlation is the process whereby different sensor readings are integrated over time to generate a track. Classification is the process of determining the type of platform of a track (e.g., an F15 and F16 fighter aircraft, a U.K. type 45 air defense destroyer, or a U.S. Arleigh Burke class destroyer), and identification is the attempt to determine its identity or allegiance in terms of it being friendly, neutral, or hostile. The threat assessment task assesses the danger a track (or a combination of tracks) represents to valued assets. The engagement task includes the decision to apply various levels of force to neutralize a threat and the execution of the decision (Arciszewski, *et al.*, 2009). Today, the tasks are primarily done by the crew and based upon trained tactics, techniques, and procedures. Artificial Intelligence (AI) microservice provide an opportunity to aid operators with automated algorithms by incorporating AI composable microservices into the OODA process to rapidly update parts of the OODA loop while mitigating impacts to the whole. However, orchestration of numerous microservices may impact the ability to provide hard-real-time trustworthy performance.

The current definition of microservices was formally introduced in 2014. (Lewis & Fowler, 2014) described microservices as an architectural style characterized around organization and business capability, automated deployment intelligence in the endpoints, and decentralized control of languages and data. The choice to adopt a microservices architecture as noted by industry giants Netflix, Twitter, Amazon, eBay and Uber has been to overcome scalability challenges, gain efficiency, increase developer velocity, and overcome difficulties in adopting new technologies. A successful, scalable microservices ecosystem requires that a stable and mature infrastructure is in place. Additionally, the organizational structure of a company adopting microservices should be aligned with the microservices architecture to gain efficiencies through alignment of microservice domain allocations. The largest challenges that microservices bring are a need for standardization of the architecture or services themselves, along with requirements for each microservice in order to ensure trust and availability (Fowler, 2016). Table 1 contains a rating and analysis of the common architecture characteristics for the microservices architecture pattern. The rating for each characteristic is based on the natural tendency for that characteristic as a capability based on a typical implementation of the pattern, as well as what the pattern is generally known from (Richards, 2015). The "low" assessment of "performance" is a motivator for this research.

Table 1. Microservice architecture pattern assessment (Richards, 2015).

Category	Rating	Analysis
Overall agility	High	Overall agility is the ability to respond quickly to a constantly changing environment. Due to the notion of separately deployed units, change is generally isolated to individual service components, which allows for fast and easy deployment. Also, applications build using this pattern tend to be very loosely coupled, which also helps facilitate change.
Ease of deployment	High	The deployment characteristics of the microservices pattern rate very high due to the fine-grained and independent nature of the remote services. Services are generally deployed as separate units of software, resulting in the ability to do "hot deployments" any time during the day or night. Overall deployment risk is also significantly reduced, in that failed deployments are able to be restored

		more quickly and only impact the operations on the service being deployed, resulting in continued operations for all other operations.
Testability	High	Due to the separation and isolation of business functionality into independent applications, testing can be scoped, allowing for more targeted testing efforts. Regression testing for a particular service component is much easier and more feasible than regression testing for an entire monolithic application. Also, since the service components in this pattern are loosely coupled, there is much less of a chance from a development perspective of making a change that breaks another part of the application, easing the testing burden of having to test the entire application for one small change.
Performance	**Low**	**While you can create applications implemented from this pattern that performs very well, overall this pattern does not naturally lend itself to high-performance applications due to the distributed nature of the microservices architecture pattern.**
Scalability	High	Because the application is split into separately deployed units, each service component can be individually scaled, allowing for fine-tuned scaling of the application. For example, the admin area of a stock-trading application may not need to scale due to the low user volumes for that functionality, but the trade-placement service component may need to scale due to the high throughput needed by most trading applications for this functionality.
Ease of development	High	Because functionality is isolated into separate and distinct service components, development becomes easier due to the smaller and isolated scope. There is much less chance a developer will make a change in one service component that would affect other service components, thereby reducing the coordination needed among developers or development teams.

AI techniques are required in the form of embedded algorithms within microservice applications to execute critical mission essential functions with reasoning about complex data and problems with a lot of interacting constraints and where there is a need to integrate heterogeneous knowledge. AI microservices will also play a critical role in the development of an integrated development environment (IDE) for ontologies, rules, and reasoning (i.e., Semantic Technology) that is built on top of the CMS. These AI microservices can be deployed as stand-alone application services or along with existing application services as part of enterprise-level applications or client/server architectures via its Web API. The use of semantic ontologies and rules as a basis for implementing microservices within the CMS and decision support systems for the DOD provides flexibility and adaptability, because domains can be captured and modeled in an incremental manner as new knowledge is gained or new integrations occur by adding or modifying the underlying schema without affecting the information that already exists in the system. This adaptability is a source of economic benefits to the enterprise as well as enabling experimentation and lowering risk. The expressiveness of the combination of ontologies and rules is an effective means to not only represent extant data sources or schemas in a canonical way through ontologies, but also capture detailed policies and behavior of decision support systems as rules. In addition, these AI microservices can incorporate checks that programs, e.g. databases, queries, analytics, etc., are sound. Since all functions are annotated with the input type and return type, a compiler can reason over an expression to determine if the expression is "well-typed." As you refine the types of a program to be more precise about which sets of values are distinguished at the type level, you get stronger guarantees that programs that type check are trustworthy. In functional programming, careful work needs to be done upfront to establish the types and the signatures of the functions in order to capture the "business logic" into the type system. Once this work to capture the expert knowledge is done, computational systems can automatically validate statements about these systems, including statements about the existence of particular mission threads. This computational validation yields higher trust in the resulting execution of complex system-of-systems (SoS).

The connection between operators and things is complex, and creates a set of trust concerns. Trust should be considered at two levels: (1) whether a thing trusts the data it receives or trusts the other things it interacts with (machine to machine (M2M)) and (2) whether humans trusts the things, services, data, or SoS offerings that it uses (human to machine (H2M) or M2H). This leads to the idea that trust is multi-dimensional. (Huang *et al.*, 2020) emphasize the importance on advanced research around digital artifacts and trustworthy systems in the following statement, "To address the above research issues, digital systems engineering needs to integrate and leverage digital technologies such as Big Data technologies (including

cloud computing), Data Science, ML, AI, semantics technologies, as well as digital mechanisms of security and trust developed in cybersecurity, Blockchain, and computational trust communities."

The rest of this paper is organized as follows. Section 2 provides an overview of literature related to the research topic. Section 3 presents the proposed research and hypothesis statements. Section 4 presents the research methodology and framework. Section 5 presents anticipated "toy problem" results prior to the completion of the research. Section 6 presents conclusions within the context of this JIDPS journal edition.

2. Literature Review

The following provides an overview of what hard-real-time means in terms of developing an integrated combat system-of-systems (CSoS) that is managed by a CMS. In addition, a literature review is presented on the key technology areas and techniques to include microservice, container, and orchestration technologies which are all being pursued by DoD to support the DevSecOps vision.

2.1. Defining hard-real-time

(Jamshidi, 2009) discusses the need to revisit all aspects of systems engineering to address the key aspects of sensing and control within a system of systems. Several patterns are presented to provide options for addressing design concerns. Modern combat systems are evolving to use of a hierarchical control pattern to address not only coordination and control of organic sensors and weapons on own ship, but the pattern supports extensibility to offboard systems connected by communication within a task force. Figure 1 is adapted to show the relationship between the pattern presented in (Jamshidi, 2009) to patterns currently found in modern combat systems (Pollard, 1991). The sensors and weapons each can provide standalone capabilities to the ship; however, the coordinator components enable allocation of tasking across multiple resources. The Tactical Decision Support Software (TDSS) within a Combat Management System (CMS) enables a CSoS capability where sensors and weapons can be paired to provide increased combat capability. TDSS may rely on multiple algorithms based upon known tactics, techniques, and procedures to provide automated sensor and weapon employment recommendations to shipboard operators; e.g. AI. To be useful, these recommendations must be timely, and response times must be well understood, consistent and predictable. (Jamshidi, 2009) points out that "real-time control – which is required in almost all application domains – of interdependent systems poses an especially difficult problem."

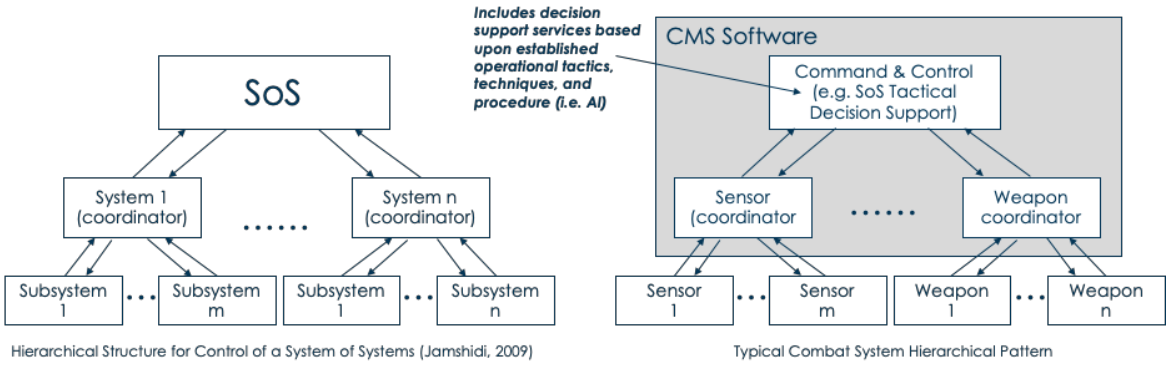

Fig. 1. Application of a Hierarchical Resource Pattern Within Combat Systems (Jamshidi, 2009).

There's a common misconception that real-time means fast. Real-time is really focused on getting the expected result given input "in time" for the response to be useful; e.g. meeting a deadline. (Abbott, 2017) provides a good definition of this condition:

"Real-time is even harder to explain. The basic idea behind real-time is that we expect the computer to respond to its environment in time. But what does "in time" mean? Many people assume that real-time means really fast. Not true. Real-time simply means fast enough in the context in which the system is operating. …. The essence of real-time computing is not only that the computer responds to its environment fast enough, but that it responds reliably fast enough."

The severity of missing a deadline is further defined as hard, soft, and firm real-time. Missing a hard-real-time deadline can result in catastrophic mission failure. Figure 2 from (Wang, 2011) illustrates the impact of missing a deadline in a real-time system. V(t) defines the value of making or missing a deadline. In soft real-time subfigure (a) if a deadline is missed the value of the data provided gracefully degrades over time. In firm real-time subfigure (b) if a deadline is missed, the data after the deadline is of no value. In hard essential real-time subfigure (c) if the deadline is missed, there is a known defined penalty (n). In hard critical real-time subfigure (d) if the deadline is missed, the realized disaster is of exponential unknown impact.

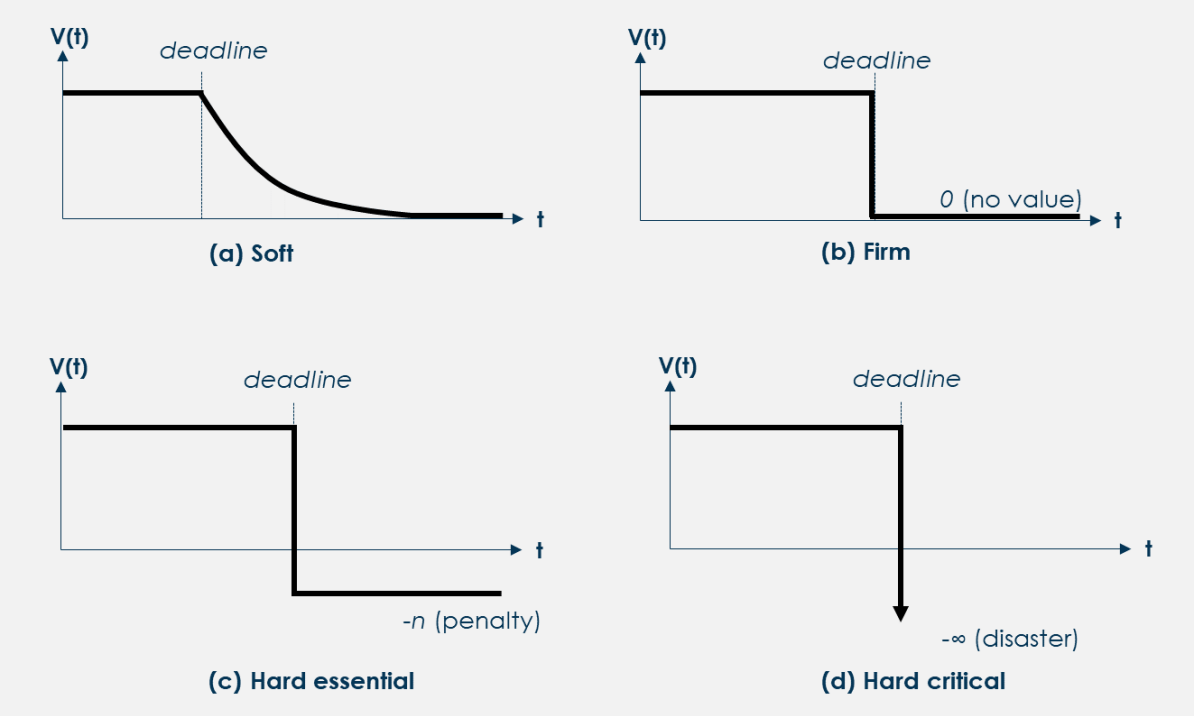

Fig. 2. Deadline Represented with Value Functions (Wang, 2011).

In Fig. 3 we define a combat system use case to illustrate the impact of missing a hard-real-time deadline. In the scenario depicted, the Weapon Coordinator software component requests effectiveness options from the integrated USV, Missile, and UAV subsystems via their controllers interfaces. Controller interfaces provide for normalization of interface data into a common data model. Let's assume that the hard-real-time deadline for controller responses is set to 75 milliseconds (ms). The Weapon Coordinator would need to make a weapon selection decision prior to getting the USV Controller response in 100 ms. The negative impact or penalty is a less effective, higher cost weapon would be selected and potentially waste a weapon that could be more effective in a future engagement.

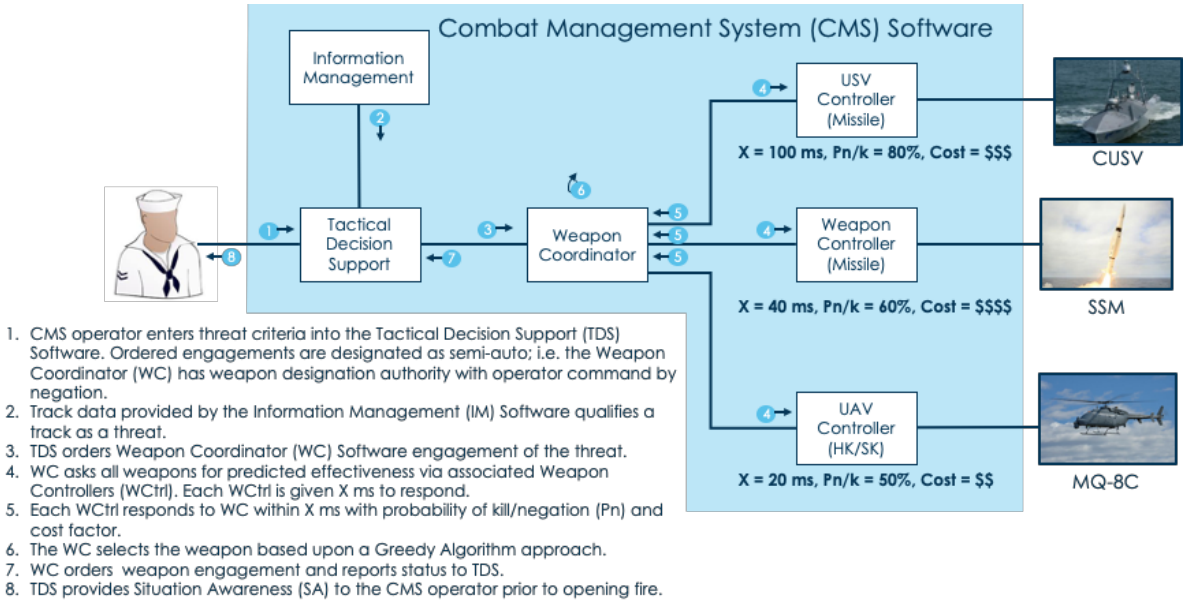

Fig. 3. System-of-Systems (SoS) Scheduling Example.

2.2. DevSecOps technologies

The following is an overview of DevSecOps technologies proposed for use within a future CMS.

2.2.1. Microservices in industry with web-based interfaces

During 17 interviews with software professionals from 10 companies, (Bogner *et al.*, 2019) analyzed 14 service-based systems. The interviews focused on applied technologies, Microservices characteristics, and the perceived influence on software quality. The authors found that companies generally rely on well-established technologies for service implementation, communication, and deployment. Most systems, however, did not exhibit a high degree of technological diversity as commonly expected with microservices.

In (Bogner, *et al.*, 2019), the de facto standard for microservice communications was RESTful HTTP. Representative state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating web services that may be accessed using the web-based hypertext transfer protocol (HTTP). Even though it was not the primary protocol in each of the 14 cases, it existed in all of them, sometimes for minor interfaces. Participants named interoperability, technology independence, and loose coupling as advantages, even though most participants that used REST felt no need to justify this decision. Some participants saw direct synchronous RESTful communication between services as harmful and relied more on messaging to decouple services further. Kafka was the preferred messaging solution followed by the Advanced Message Queuing Protocol (AMQP).

For the deployment of services, (Bogner, *et al.*, 2019) most often relied on Docker containers. Overall, the operability and portability of Docker are valued very highly. Two participants also made positive experiences with Cloud Foundry that was described as more developer-friendly as Kubernetes which was generally seen as powerful but complex. Even though there is the option for diverse languages with microservices, Java is used in each of our 14 cases, in several exclusively or in combination with other Java Virtual Machine (JVM) languages. Reasons are the availability of skilled developers plus mature frameworks like Spring and a solid tool ecosystem.

The following negative feedback on the use of a microservices architecture was provided by the interviews (Bogner, *et al.*, 2019):

- In many of the migration cases, the "killer attribute," maintainability was either the main migration driver or reported as the most obvious improvement over the monolith.
- Reusability was seen as rather ambiguous. While testability of a single service would be greatly improved, the increased outer complexity could prevent the same on system level.
- In general, reliability was always a challenge as something was always down when you consider many microservices working together. So, the consensus was that you need a lot of tools and knowledge to be more reliable than your old monolith due to the increased complexity.
- No participant explicitly mentioned that microservices would ease securing a system due to increased attack surface. To tackle this challenge, it would be beneficial to use the API gateway pattern.

2.2.2. Container overhead

(Wei *et al.*, 2018) notes that surprisingly, however, there are very few studies revealing the over-heads, such as starting new containers in orchestration systems, such as Kubernetes. Though traditional Virtual Machines (VMs) can take on the order of minutes to launch, containers are much faster and the launch times can be on the order of seconds. These overheads are typically considered to be negligible compared with the benefits of container-based systems, however, are they predictable?

(Wei, *et al.*, 2018) investigated costs in a systematic study within a private cloud platform. Their results confirm that launch times of VMs are in the range of minutes, whereas containers typically only take seconds. However, these results also show that launch times for new containers do not always scale linearly. Specifically, the discussion of a system organized by Minikube, a tool that eases local deployment of Kubernetes, introduces a penalty on launch times once the number of containers exceeds 80% of the maximum number of pods available for the cluster. This work demonstrates the presence of unexpected overheads and the need for our proposed systematic infrastructure for testing deployments of containerized services at scale.

2.2.3. Auto-scaling technology

(Kho Lin *et al.*, 2018) explores the use of Kubernetes technology for an Australian Defense Forces (ADF) ATHENA platform. ATHENA is a strategic simulation and analysis platform focused on manpower planning. ATHENA leverages container-based technologies for auto-scaling. As the market leader, Docker was used as the core container-based solution and Kubernetes was used as the container orchestration. In Kubernetes, the concept of a Pod is used to encapsulate containers. A Kubernetes Pod object holds one or more containers and introduces an IP-per-Pod network model. Therefore, containers within a Pod share their network namespaces including their IP address. In Kubernetes, Pods are ephemeral. That is, a Kubernetes cluster can replicate Pods (destroy and re-create new ones) for dynamically scaling up and down, for self-healing purposes and/or for self-managing purposes. This is challenging for application developers to track.

To experiment with the auto-scaling setup and benchmarking, (Kho Lin, *et al.*, 2018) configured the ATHENA Worker Horizontal Pod Autoscaler (APA), to use 80% target CPU utilization with 1 CPU resource request for each Worker Pod instance. The HPA replication factor was set to a minimum of one Pod to a maximum of 6 Pods. The trend lines showed that the rate of increase in runtime decreases as more resources were added; this is a clear indication that auto-scaling was successful. However, during the experimental runs, it was noted that the auto-scaler doesn't react immediately to usage spikes. The authors concluded that an auto-scaling system cannot meet the user performance demands by simply relying on CPU utilization and memory usage metrics. Most web and mobile applications require auto-scaling based upon requests per second to handle burst traffic and stochastic user load.

2.3. Trustworthiness in systems

(Ahn *et al.*, 2007) described the concept of multi-dimensional trust by different agent characteristics, such as quality, reliability, and availability. For (Matei *et al.*, 2009), trust refers to the trustworthiness of a sensor, whether it has been compromised, the quality of data from the sensor, and the network connection. (Grandison & Sloman, 2000) define trust as the belief in the competence of an entity to act dependably, securely, and reliably within a specified context. Lastly, recognizing trust is multi-dimensional, NIST defines it as "… the demonstrable likelihood that the system performs according to designed behavior under any set of conditions as evidenced by characteristics including, … security, privacy, reliability, safety and resilience" (Griffor *et al.*, 2017).

2.4. Useable AI

For AI to be useable, a common language must be defined to ensure that the service and service user correctly interpret the information provided. In early AI research, (Gruber, 1993) stated:

"Several technical problems stand in the way of shared, reusable knowledge-based software. Like conventional applications, knowledge-based systems are based on heterogeneous hardware platforms, programming languages, and network protocols. However, knowledge-based systems pose special requirements for interoperability. Such systems operate on and communicate using statements in a formal knowledge representation. They ask queries and give answers. They take 'background knowledge' as an input. And as agents in a distributed AI environment, they negotiate and exchange knowledge. For such knowledge-level communication, we need conventions at three levels: representation language format, agent communication protocol, and specification of the content of shared knowledge."

Web Ontology Language (OWL) and Resource Description Framework Schema (RDFS) help to facilitate shared knowledge and common understanding. RDFS was designed to create a common English based subject-predicate-object model that enables connecting of data within the semantic web (W3C, 2010). OWL based ontologies help different domains (e.g. business units) understand and share concepts using a common language to avoid confusion. This common understanding is an enabler for the application of AI by making the data machine usable. Ontologies also support reasoning.

(Huang, 2018) established an ontology to assess Scientific Computing Integrity (SCI). This ontology integrates an Open Provence Model (OPM), temporal logic, and trust reasoning to enable SCI validation. A failure of SCI may be caused by malicious attacks, natural environmental changes, faults of scientists, operations mistakes, faults of supporting systems, faults of processes, and errors in the data or theories on which a research relies. Factoring AI into modular microservices requires the decomposition of data-driven algorithmic functions into reusable primitives. For AI, the core primitives consist of algorithms that perform regression, classification, clustering, predictive analysis, feature reduction, pattern recognition, and natural language processing.

3. Research Areas Being Investigated

Knowledge gaps identified in the literature review are the basis for the research problem statements presented in this section. The primary research relates to the deterministic nature of DevSecOps technologies and can be formed in the following questions:

Q1: Is it possible to synthesize cloud computing and open-source technologies to realize a Microservices architecture for a hard-real-time deterministic CMS?

The additional research questions proposed are related to the application of Mission Engineering with respect to implementing specific microservice technologies and techniques as part of executing critical mission threads:

Q2: Does end-to-end (E2E) mission thread analysis increase SoS interoperability and reduce integration issues?
Q3: What benefits are gained from a microservice based DevSecOps approach?
Q4: Is the resultant architecture hard-real-time deterministic?

3.1. Generation of hypothesis statements

Hypothesis is "innocent until proven guilty." We'll assume that SpaceX and others have proven that DevSecOps tech can meet hard-real-time requirements but nothing available in the body of knowledge documents this.

Hypothesis: Modern DevSecOps architectures can be designed to meet hard-real-time latency (μ) requirements using modern computing environments and computing infrastructure:

H_0: $\mu \leq$ tbd ms with jitter within latency bounds
H_a: $\mu >$ tbd ms with jitter exceeding latency bounds
$\propto = 0.05$

Experiments will include single-node and multi-node configurations per the referenced papers (see Section 2).

Additional hypothesis statements were derived from predicted results based upon the previously identified research questions to put the null and alternate hypothesis results within a SoS and mission context. The independent variables to be measured are throughput, latency, and jitter while the associated dependent variables are rapid fielding of capability and usability. The Research Framework relationships to be used for analysis of the hypotheses are depicted in Fig. 4. A similar format was used in (Moreland Jr., 2013). The independent variables are throughput, latency, and jitter that are measurable as the tactical environment varies to assess the dependent variables of "Rapid fielding of capability to pace threats" and "Usability." Associated hypotheses are depicted in the center column.

From Q1 it is predicted (P1) that AI microservices architecture will have a positive impact on time to implement capability upgrades and development cost. Therefore, Hypothesis 1 (+H1) is captured accordingly: The scalability of microservices as new data sources added to an architecture enables maintaining high **throughput** and predicted to have a positive impact on **rapid fielding of capability**.

From Q2 it is predicted (P2) that DevSecOps container orchestration technologies (e.g. Kubernetes) will have a positive impact on combat system availability and latency due to the ability to tune deployment configurations and load balance. The resulting Hypothesis 2 (+H2) reads: Microservices orchestration through DevSecOps technologies enables maintaining low **latency** service call responses and is predicted to have a positive impact on **usability**.

From Q3 it is predicted (P3) that web-based user interface technologies will have a negative impact on deterministic hard-real-time performance needed for positive control of organic weapons. This prediction results in the following Hypothesis 3 (-H3): Web-based interfaces (e.g. RESTful HTTP) will increase **jitter** which is predicted to have a negative impact on **usability** and the deterministic performance needed for positive control of organic weapons.

From Q4 it is predicted (P3) that a systems architecture model will have a positive impact on micro-services architecture performance prediction and prediction of associated mission thread impacts. This leads to the final hypothesis or Hypothesis 4 (+H4): A system architecture model can predict end-to-end **latency** within a mission thread to quantify SoS **usability**.

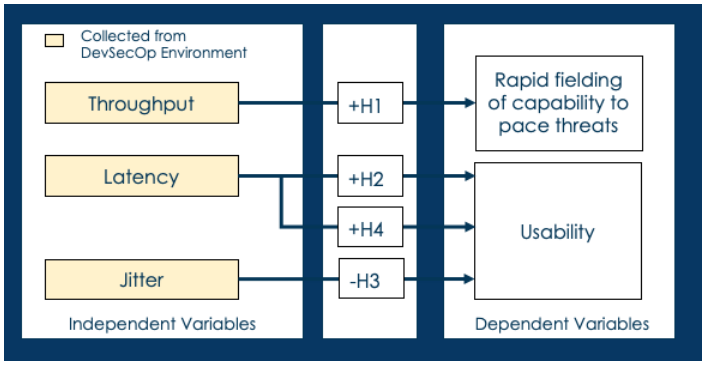

Fig. 4. Conceptual Analysis Framework.

4. Research Methodology and Framework

This research seeks to use an Empiricist, Positivist, Deductive paradigm as defined by (Siangchokyoo & Sousa-Poza, 2012).

- Epistemological Position: Empiricist – Justification of Knowledge through observation
- Ontological Position: Positivist – Seek to find reality independent of the observer
- Mode of Reasoning: Deductive – Usage of confirmatory reasoning to obtain knowledge

A quantitative analysis approach is envisioned to gather data required for deductive analysis. To achieve this goal, this research will leverage a CMS Prototype built within a DevSecOps cloud computing infrastructure (e.g. Amazon Web Services) to simulate an operational mission thread scenario and generate representative CMS data under operational mission conditions for analysis. The use of a DevSecOps environment supports the positivist ontological position through the use of automation. A primary precept of DevSecOps is to automate everything which creates an environment of repeatability.

A mission-based approach will be used to set the context for applied research. A hypothetical yet operationally relevant strait transit scenario has been established to provide context for definition of **experimental** parameters to be set while assessing the hypotheses. System models and data from a cloud computing environment will be used to collect data for **quantitative** analyses.

4.1. Mission context

Political hostility between Country Red and Country Green has escalated over the past six months to a point where military conflict is imminent. The Country Green Navy Cruiser (CG), USS Dahlgren, has been tasked with a mission to escort a Country Green Command Ship (LCC) through a Strait between the two countries to establish a command post in the event of wartime activity. USS Dahlgren will lead a Surface Action Group (SAG) composed of a destroyer (DD) and an unmanned surface vehicle (USV). Additionally, USS Dahlgren is equipped with an unmanned aerial vehicle (UAV). During mission planning, USS Dahlgren acquires intelligence data that indicates that Country Red has made modifications to their surface combatant weapon systems that will impact the USS Dahlgren abilities to engage and develop fire control solutions against Country Red's surface-to-surface missiles (SSM). USS Dahlgren contacts its Country Green shore support activity to identify and develop software upgrades within 72 hours to react to the emergent threat. Country Green develops potential solutions within their DevSecOps Software Factory to provide a rapid response. After User Centered Design (UCD), system analysis through simulation, CI/CD system integration testing, and DevSecOps automation enabled certification, Country Green is able to deliver an effective and suitable solution from their Software Factory prior to Country Green SAG deployment. The USS Dahlgren Task Group enters the Strait. Once the SAG enters the Strait, National Technical Means (NTM) detects a Country Red surface threat near the end of the Strait (see Fig. 5). A non-organic UAV in close proximity to the surface threat is tasked to provide additional targeting information

(e.g. Triton). Joint assets may also be tasked to provide support. A targeting solution is provided to the CMS and over-the-horizon (OTH) weapons are employed. Figure 5 depicts the operational concepts in an operational view (OV-1) with the corresponding mission thread using the Find-Fix-Track-Target-Engage-Assess (F2T2EA) mission essential tasks taxonomy. The CMS is inclusive of AI microservices.

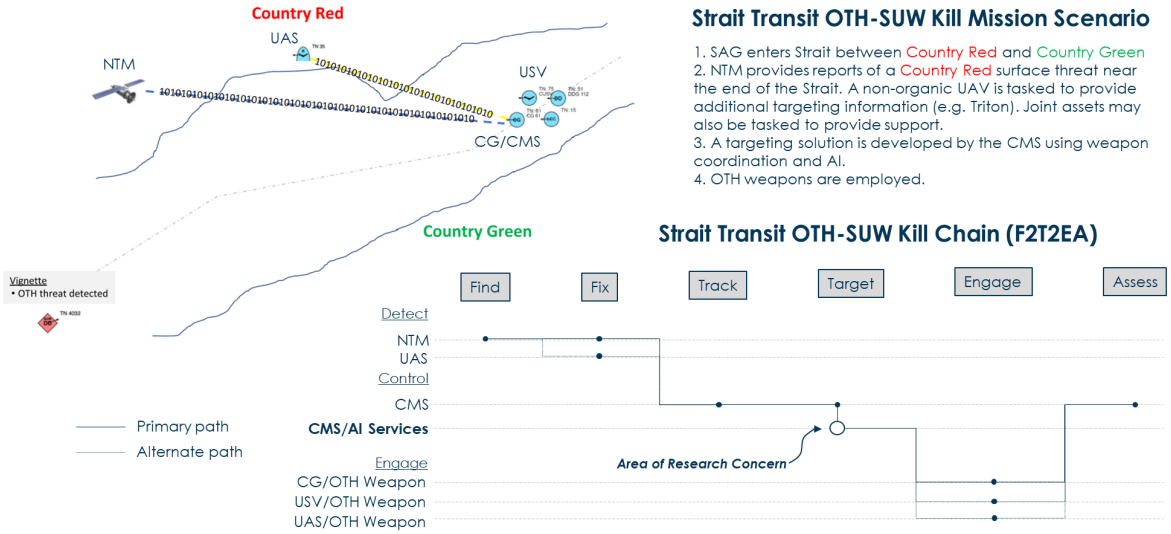

Fig. 5. Mission Scenario (OV-1) and Mission Thread.

4.2. SoS integration

Figure 6 depicts the SoS integration context within a single ship communicating with offboard assets. The figure leverages an End-to-End (E2E) data flow analysis technique developed by the Software Engineering Institute (SEI) to summarize critical data flows (Firesmith, 2019). The primary system of interest is the CMS where cloud technologies have replaced legacy implementations. SoS interactions from the time that NTM detects the track to order for engagement is summarized as follows:

- 1,2: NTM "tip" on OTH target provided to Tactical Actions Officer (TAO)
- 3,4,5,6,7,8,9: UAS tasked to "Find and Fix" OTH target to provide surveillance and targeting data
- 10,11,12,13,14: Surveillance and targeting data provided to TAO
- 15,16,17,18: TAO requests and receives "Engage" performance prediction(s) from CMS and available OTH Weapon(s); e.g. SoS
- 19,20: TAO orders OTH Weapon System(s) engagement

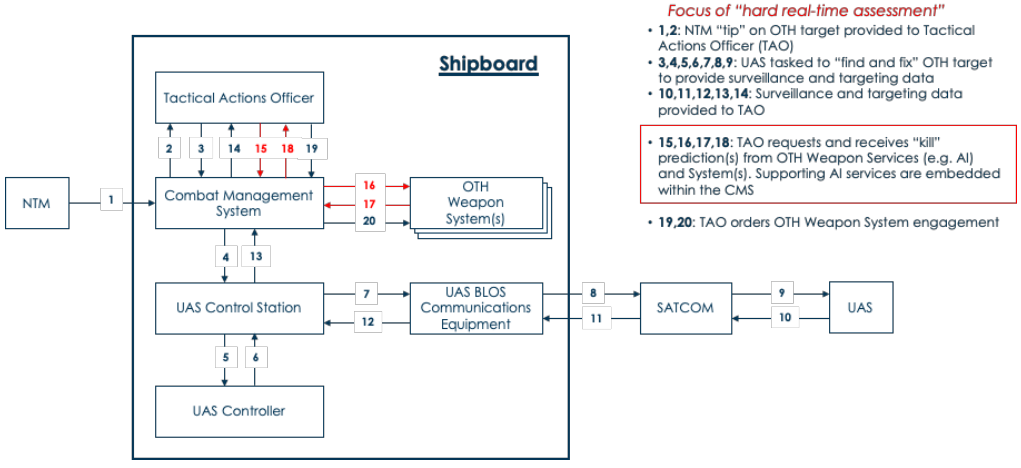

Fig. 6. SoS E2E integration Mission Thread "Cartoon."

4.3. Sample problem

An experimental simulation will be constructed within one or more computing environments to assess the hard-real-time performance of representative AI microservices. User requests to the AI microservices will be injected to assess response latency, jitter, and throughput. The context, container, component, and code (C4) model style from (Brown, 2019) is used to present the sample problem architecture. Fig. 7 presents a context for a generic AI microservices implementation. The context includes an SoS database to provide data to the AI microservices to generate AI based recommendations. SoS sensors and effectors provide track data, systems statuses, and current tasking to the SoS database.

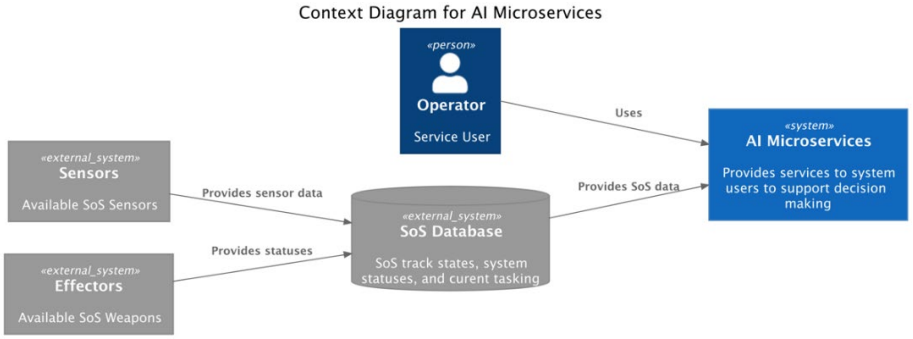

Fig. 7. SoS context diagram for AI Microservices Prototype.

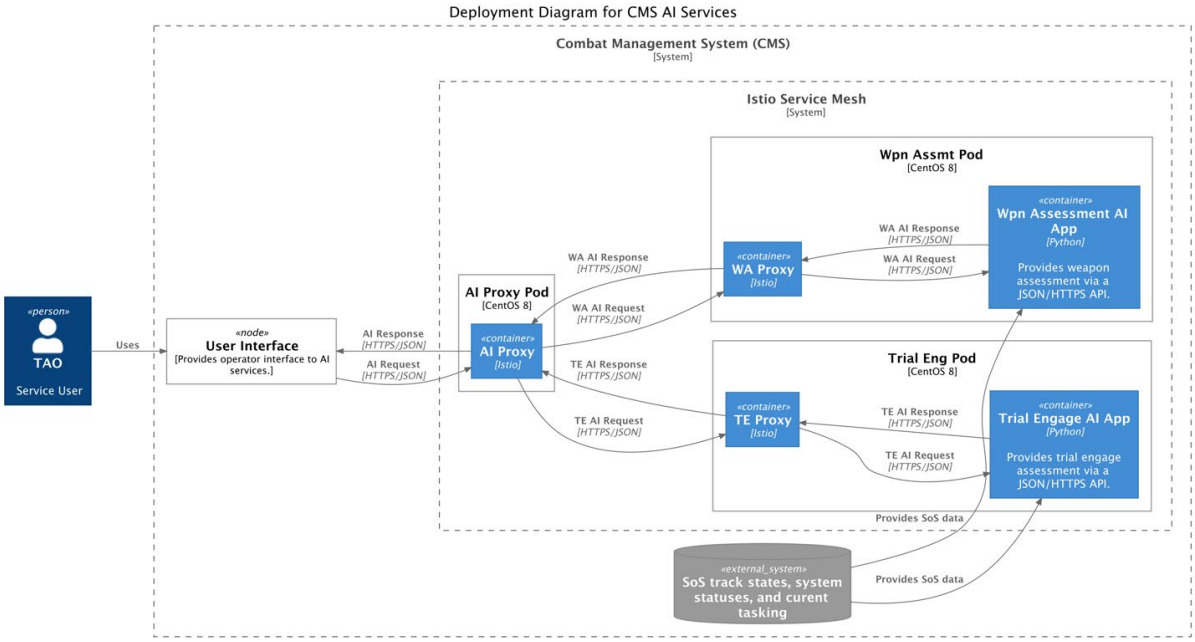

Fig. 8. Deployment diagram for AI Microservices Prototype.

Figure 8 expands the generic context into a specific sample deployment of two AI microservices within cloud-based infrastructure and a Navy Combat Management System (see Fig. 1, Fig. 3, and Fig. 6). The generic "Service User" is now as shipboard TAO. The Weapon Engageability Services review target track kinematics and organic weapon systems capabilities to provide and assess weapon/target pairing for engagement of the specific target. The assessment is based upon known weapons' capabilities against the position of the target and kinematic capabilities. The Trial Engage Service assesses the target against weapon capabilities and known tactics to use against a target. Both services require hard-real-time responses to enable operator weapon selection within the time for weapons release and consummation of the engagement. If recommendations are latent, destruction of the targeted platform is imminent.

A sidecar implementation pattern using Istio is planned to add security to assure data integrity. The Istio service mesh adds a layer of security (e.g. trustworthiness), but may impact deterministic performance. The Istio data plane within the mesh is used to define what services can talk to each other via the proxies that reside within the container pods. All traffic within the mesh is controlled and protected by the Istio technology. (Li et al., 2019) discusses service mesh challenges in detail as well as future research opportunities.

5. Analysis Example

Table 2 presents notional data extracted from the experimental test environment showing measurements of critical attributes when moving data across an SoS mission thread. Table 3 provides definitions of the extracted data and the associated units.

Table 2. Notional Sample Data for Analysis.

Time <int>	Service <chr>	TN <int>	Range <int>	Latency <dbl>	RespCode <int>	Wpns <chr>	OptRvd <int>
153020	WA	1232	27	0.43	200	Type-C, Type-D	3
153022	TE	1232	25	0.35	200	Type-C	2
153025	TE	3242	100	0.22	401	N/A	1
161502	WA	2365	100	0.32	404	N/A	2
161607	WA	2365	80	0.83	200	Type-A	4
161802	TE	2367	74	0.37	402	N/A	1
165005	WA	2453	83	0.52	200	Type-B, Type-C	4
165204	WA	2453	81	0.49	200	Type-B, Type-C	3
165258	TE	2453	76	0.31	200	Type-C	1
165332	TE	2453	72	0.39	200	Type-B	1

Table 3. Variable Definitions.

Variable	Data Type	Units	Description
Time	Integer	HHMMSS	Time of recorded event in hours, minutes, seconds.
Service	String	N/A	WA = Weapon Assessment AI Service. Determines what weapons have capability against the track with subject TN.
			TE = Trial Engage AI Service. Determines engagability of a particular weapon against track with subject TN.
TN	Integer	Octal	Assume octal at this time to ensure compatibility with existing systems and TDL standards that have not evolved away from octal.
Range	Real	NM	Range from ownship reference point to subject TN in nautical miles.
Latency	Real	Seconds	Time from service request to service response in seconds.
RespCode	Integer	Enumeration	Response code from the called service.
			Standard response codes exist, but only a few are used here for illustrative purposes. 200 = OK; e.g. the request has succeeded. 401 = Unable to process (Invalid TN). 404 = Unable to process (No intercept solution exists). 402 = Unable to process (Invalid Track).
Wpns	String	N/A	For WA, provides a list of weapons with an interceptability solution. For TE, identifies the weapon be using for trail engage assessment.
OptRvd	Integer	Options	Total number of options evaluated while determining a solution. This can provide an indication of processing complexity.

5.1. Exploratory data analysis

Figure 9 and Fig. 10 present sample exploratory data analysis plots from the sample data. The plots were generated using R within a Juypter Notebook. Exploratory data analysis will be used first to make an initial

assessment of the data to identify multivariate analysis techniques to be used to assess hypotheses. Quantile-quantile plots (Q-Q) are presented to help assess if a set of data plausibly came from a well-defined theoretical distribution (i.e., normal or exponential). A Q-Q plot is a scatterplot created by plotting two sets of quantiles against one another. If both sets of quantiles came from the same distribution, we should see the points forming a line that's roughly straight. Histograms are also provided to see the distribution of latency and options reviewed. The plots depict the latency of each of the individual services as well as latency attributed to the complexity of each AI microservice assessment (e.g. options reviewed). The intent is to use this sample problem analysis framework for the real experimental data to evaluate the hypotheses.

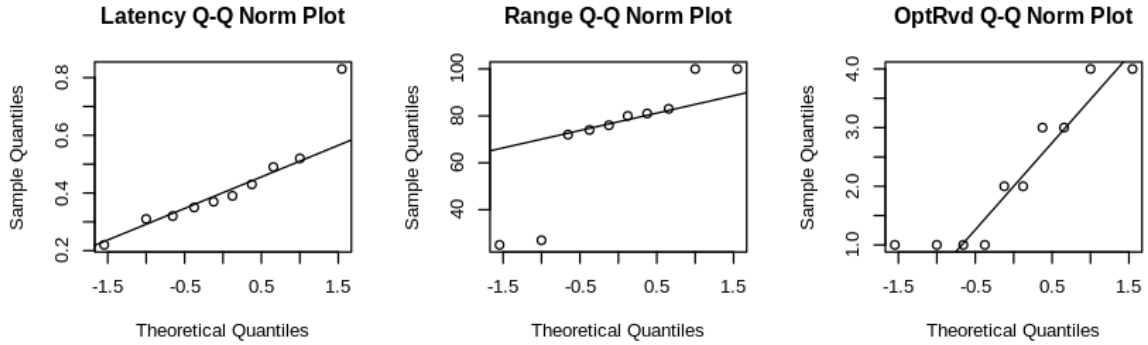

Fig. 9. Quantile-Quantile (Q-Q) Plots.

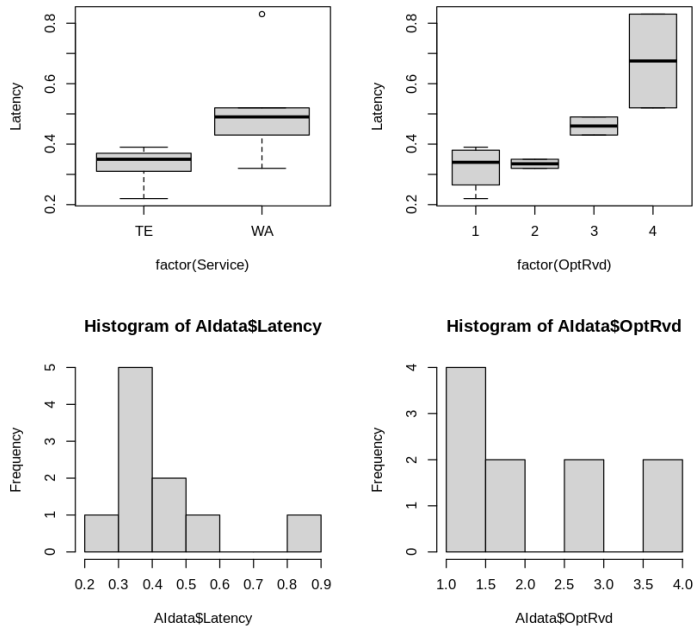

Fig. 10. Exploratory Data Analysis Plots.

5.2. Hypothesis testing

Based upon exploratory data analysis, we will set our latency threshold at 500 ms for analysis until a defendable operational requirement is defined. We'll start with a Shapiro-Wilk test to establish the normality of the latency distribution.

H0: The data points are normally distributed
Ha: The data points are not normally distributed
$\propto = 0.05$
Statistical analysis using the R-based Juypter Notebook yielded the following results:

```
    Shapiro-Wilk normality test

    data:  AIdata$Latency
    W = 0.85735, p-value = 0.07101
```

Given that p-value = 0.071 > $\propto$ = 0.05, we fail to reject the null hypothesis that the data is normally distributed; i.e. we will assume that the data is normally distributed.

Next we'll perform a t-test to assess our primary hypothesis that latency meets a threshold of 500 ms; i.e. 0.5 seconds.

```
One Sample t-test

   data:  AIdata$Latency
t = -1.4511, df = 9, p-value = 0.9097
alternative hypothesis: true mean is greater than 0.5
95 percent confidence interval:
 0.3257298       Inf
sample estimates:
mean of x
    0.423
```

Given that p-value = 0.910 > $\propto$ = 0.05 we fail to reject the null hypothesis that latency is ≤ 500 ms; i.e. we will assume that latency is below our threshold and AI microservices can meet our 500 ms threshold requirement within an SoS environment.

6. Conclusion

Research into the suitability of DevSecOps to meet hard-real-time requirements within a CMS based SoS is needed to ensure mission success. Mission context and an analysis framework have been defined to collect data required for deductive analysis. Our "toy problem" establishes a methodology for quantitative analysis and demonstrates that the hypothetical data can meet hard-real-time latency requirements within the context of an AI microservice architecture. The next step is to develop representative microservices within a cloud-based software environment to proceed with testing, data collection, and evaluation. The AI microservices are envisioned to use DevSecOps practices for design, development, integration, testing, and analysis. It is anticipated that a DevSecOps approach is suitable for the acquisition of future CMS software; however, design analysis is required to assess architectural patterns and options to make informed design decisions.

References

Abbott, D. (2017). *Linux for Embedded and Real-Time Applications*: Newnes.

AFCEA. (2019). Defense Department's Devsecops Initiative Is on the Move, from https://www.afcea.org/content/node/20892/

Ahn, J., DeAngelis, D., & Barber, S. (2007). *Attitude Driven Team Formation Using Multi-Dimensional Trust*. Proceedings of 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'07). 229-235. doi:10.1109/iat.2007.77

Arciszewski, H. F. R., de Greef, T. E., & van Delft, J. H. (2009). Adaptive Automation in a Naval Combat Management System. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 39*(6).

Bogner, J., Fritzsch, J., Wagner, S., & Zimmermann, A. (2019). *Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality*. Proceedings of 2019 IEEE International Conference on Software Architecture Companion (ICSA-C). 187-195. doi:10.1109/icsa-c.2019.00041

Brown, S. (2019). *Software Architecture for Developer, Volume 2*. Leanpub.

Endsley, M. R. (1987). The Application of Human Factors to the Development of Expert Systems for Advanced Cockpits. *Human Factors Society - 31st Annual Meeting*,

Firesmith, D. (2019, 5 August). Mission Thread Analysis Using End-to-End Data Flows - Part 1. *Mission Assurance* Retrieved 5 May, 2020, from https://insights.sei.cmu.edu/sei_blog/2019/08/mission-thread-analysis-using-end-to-end-data-flows---part-1.html

Fowler, S. J. (2016). *Production-Ready Microservices*: O'Reilly Media, Inc.

Grandison, T.& Sloman, M. (2000). A Survey of Trust in Internet Applications. *IEEE Communications Surveys & Tutorials*.

Griffor, E. R., Greer, C., Wollman, D. A., & Burns, M. J. (2017) Framework for Cyber-Physical Systems: Volume 2, Working Group Reports. NIST.

Gruber, T. R. (1993). Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal Human-Centered Studies, 43*, 907-928.

Huang, J. (2018). *From Big Data to Knowledge: Issues of Provenance, Trust, and Scientific Computing Integrity*. Proceedings of 2018 International Conference on Big Data, Seattle, WA.

Huang, J., Gheorghe, A., Handley, H., Pazos, P., Pinto, A., Kovacic, S., Collins, A., Keating, C., Sousa-Poza, A., Rabadi, G., Unal, R., Cotter, T., Landaeta, R., & Daniels, C. (2020). Towards Digital Engineering: The Advent of Digital Systems Engineering. *Int. J. System of Systems Engineering, 10*(3), 234-261.

Jamshidi, M. (2009). *Control of System of Systems*. Proceedings of INDIN, Cardiff, UK.

Kho Lin, S., Altaf, U., Jayaputera, G., Li, J., Marques, D., Meggyesy, D., Sarwar, S., Sharma, S., Voorsluys, W., Sinnott, R., Novak, A., Nguyen, V., & Pash, K. (2018). *Auto-Scaling a Defence Application across the Cloud Using Docker and Kubernetes*. Proceedings of 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion). 327-334. doi:10.1109/UCC-Companion.2018.00076

Lewis, J.& Fowler, M. (2014, 1 March). Microservices. Retrieved from https://martinfowler.com/articles/microservices.html

Li, W., Lemieux, Y., Gao, J., Zhao, Z., & Han, Y. (2019). *Service Mesh: Challenges, State of the Art, and Future Research Opportunities*. Proceedings of 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE). 122-1225. doi:10.1109/sose.2019.00026

Matei, I., Baras, J. S., & Jiang, T. (2009). *A Composite Trust Model and Its Application to Collaborative Distributed Information Fusion.* 12th International Conference on Information Fusion, 1950-1957

Moreland Jr., J. D. (2013). *Service-Oriented Architecture (Soa) Instantiation within a Hard Real-Time, Deterministic Combat System Environment.* Dissertation, The George Washington University, ProQuest.

Parasuraman, R., Sheridan, T. B., & Wickens, C. D. (2000). A Model for Types and Levels of Human Interaction with Automation. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 30*(3), 286-297. doi: 10.1109/3468.844354

Pollard, J. R. (1991) Combat Systems Vision 2030: Functional Architecture for Future Shipboard Combat Systems. NAVSWC.

Richards, M. (2015). *Software Architecture Patterns*: O'Reilly Media, Inc.

Siangchokyoo, N.& Sousa-Poza, A. A. (2012). Research Methodologies: A Look at the Underlying Philosophical Foundations of Research. *2012 International Annual Conference of the American Society for Engineering Management*, 714-722

W3C. (2010). Rdf Vocabulary Description Language 1.0: Rdf Schema (Rdfs) Retrieved 30 Jan, 2021, from https://www.w3.org/2001/sw/wiki/RDFS

Wang, R. R. (2011, 1 Mar 2020). Opher Etzion on Four Types of Real-Time. Retrieved from http://blog.softwareinsider.org/2011/06/20/mondays-musings-real-time-versus-right-time-and-the-dawn-of-engagement-apps/screen-shot-2011-06-20-at-6-38-45-am

Wei, T., Malhotra, M., Gao, B., Bednar, T., Jacoby, D., & Coady, Y. (2018). *No Such Thing as a "Free Lunch"? - Systematic Benchmarking of Containers.* 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)

Author Biographies

Alvin C. Murphy is a Principal Engineer within the Integrated Combat Systems Department at NSWC Dahlgren. He earned a B.S. degree in electrical engineering from Virginia Tech in 1991, and a M.S. in systems engineering from George Mason University in 2007. Mr. Murphy is currently a Ph.D. Candidate at Old Dominion University. He has spent the past 30 years engineering, developing, testing, integrating and assessing Warfare Systems and C4I for AEGIS, the Navy, and Joint Warfighter. In 1999, Mr. Murphy broadened his focus from individual platforms to strike force systems engineering as a plank holder in the development of the Navy's Distributed Engineering Plant (DEP) and Battle Force Interoperability Requirements (BFIR) and Metrics definition. Mr. Murphy currently leads combat system requirements and architecture development for Future Navy surface platforms and surface Navy Enterprise systems engineering initiatives. Additionally, Mr. Murphy is currently leading surface Navy combat system DevSecOps initiatives and associated collaboration at Naval and DoD levels.

James D. Moreland, Jr. retired from the Government and Senior Executive Service on 28 March 2020, and is now Deputy VP, Strategy and Business Transformation and Executive Director, Transformational Strategy for Raytheon Technologies in the Missiles and Defense organization. He retired as the Office of the Under Secretary of Defense for Acquisition and Sustainment (OUSD (A&S)) Executive Director, Mission Engineering and Integration (MEI) with 31 years of service. He has tremendous experience in leading advanced scientific research and engineering design, development, and integration for national security defense systems. Dr. Moreland is a world-renowned expert in MEI, and serves as a Senior Executive Science Advisor to the White House Office of Science & Technology Policy providing advice on scientific, engineering, and technological matters. He earned a Ph.D. in Systems Engineering from The George Washington University in 2013; a M.S. in National Resource Strategy from the Industrial College of the Armed Forces in 2001; a M.S. in Systems Engineering from Virginia Tech in 1997; and a B.S. in Mechanical Engineering from the University of Maryland in 1988. Dr. Moreland is a member of the International Council on Systems Engineering (INCOSE) and senior member of the Institute of Electrical and Electronics Engineers (IEEE) and serves as an Adjunct Professor and Doctoral Research Advisor at multiple universities (e.g., ODU, VT, MIT, and GWU).