Sensor Simulation Microservices

Steven Webster
Planned Systems International (PSI)
UNITED STATES

Dr. Robert Kewley
simlytics.cloud
UNITED STATES

Dr. Joseph McDonnell
Trideum Corporation
UNITED STATES

Susan Harkrider
Night Vision and Electronic Sensors Directorate, C5ISR Center
U.S. Army Combat Capabilities Development Command (DEVCOM)
UNITED STATES

ABSTRACT

Commercial information system providers increasingly employ microservice architectures to deliver capability. A finer resolution variant of Service Oriented Architectures (SOA), microservices are distinct processes organized to meet specific business needs, communicating over technology agnostic protocols. Microservices do not obligate users to architectures, but rather enable capability compositions through independent Application Programming Interfaces (API). Night Vision and Electronic Sensors Directorate (NVESD) has begun delivering sensor models and simulations as microservices, enabling end users to compose EO/IR performance metrics and run-time scene generation into their information solutions.

Our paper describes NVESD's initial implementation of sensor microservices, and its adaptation to various use cases and integration environments. An initial demonstration executes an operational decision support use case for positioning sensors in a tactical environment. We describe use cases where sensor microservices compute sensor effects for a training audience and provide acquisition decision support for multi-domain operations. We address microservices adaptation to emerging US Army Futures Command Cross-Functional Teams (CFT) capabilities. Future work includes data augmentation capabilities with associated truth metadata servicing Machine Learning (ML) development and deployment. NVESD benefits from deploying sensor simulation microservices since they are reused for personalized training, tactical and acquisition decision support, and Artificial Intelligence (AI) assisted operations.

INTRODUCING SENSOR AS A SERVICE

A key aspect of the Night Vision and Electronic Sensor Directorate (NVESD) Modeling and Simulation Division's (MSD) mission is to improve the Soldier's understanding of imaging sensor technologies and how to use them effectively and efficiently. MSD's capabilities encompass first principal physics models, such as the Night Vision Integrated Performance Model NV-IPM[i], field and laboratory experimentation and data collections, and calibrated imaging sensor simulations. Each area of capability uses a standard tool or standard technique, and the results of every measurement are verified and validated with at least one other capability. This results in an imaging sensor "life cycle," where the sensor is measured at each phase of

design and development in order to support the acquisition, experimentation, training and operational use.

Current methodologies for implementing new sensor and/or target models into simulation have required off line processing, with subsequent import into constructive environments. Although sensor simulations have matured, current implementations still require a high degree of expertise to configure and modify. Current manual integration processes reduce the resources available for individual sensor and associated collective experimentation. "Sensor as a Service" (SaaS) provides models and high fidelity imaging sensor representations, eases consumption by scientists, engineers, analysts and other professionals, and enables greater variability for more rapid exploration of sensor - target engagement timelines and performance.

An analysis or experiment currently depends on the collaboration of participants, to include the determination of an architecture, a protocol, data to be shared, system behavior, and the fidelity of each component. This process is time consuming and often requires several iterations to "get it right." Providing a service, in this case a sensor service, allows the participants to treat that service as a black box. As long as the API is defined and accepted, the owner of that black box can determine the behaviors, data input/output, and level of fidelity required to meet the event objectives. Current simulations design, develop and produce an overall solution that is modified at each use. Providing a sensor as a service allows any simulation to use the most current, verified and validated sensor data rather than having to develop their own, and possibly inaccurate simulation.

Sensor as a Service supports development and operational activities ranging from applied research through system acquisition, test and training. At the sensor component level, SaaS improves performance modeling run times, enabling component engineers to better understand the art of the possible early in the design process. Component activities feed Model Based Systems Engineering (MBSE), which transitions traditional document-based processes to a more efficient and effective code-centric processes that are executable model based. Given that sensor-oriented systems are increasingly automated, SaaS provides an early understanding of expected sensor content. SaaS synthesized content demonstrates representative characteristics such as image resolution, color depth, reflectivity and phasing, along with codec performance and bit rates when delivered as video. These products can also be applied to the Artificial Intelligence / Machine Learning (AI/ML) aspects of systems design, where large and variant data sets provide the foundation for training and verification for candidate AI/ML algorithms. While these examples are one application of SaaS that is specific to system acquisition, there are other communities that benefit from the SaaS capability.

SENSOR AS A SERVICE USE CASES

Sensor as a Service resides within a battlespace context that can be compartmentalized for a larger Component as a Service implementation. This battlespace context can be appropriately scaled in resolution and fidelity to meet the need of the community using it. SaaS intends to assist a variety of communities with their sensor representation needs.

The highest resolution models are often required by the Analysis and Operations Research (OR) communities. One may regard Artificial Intelligence (AI) problems in the battlespace as OR optimization problems. The solution to these problems often involves a sensitivity analysis. NV-IPM can provide the data necessary for scientific analysis of systems in a high-resolution environment. Implementing within SaaS enables the analyst to vary the sensor details while keeping the contextual environment fixed, allowing results to be compared across different sensor packages.

The training community requires high resolution models for individual skills training and lower resolution models for group training of Concepts of Operations (CONOPS) and Tactics, Techniques, and Procedures (TTP). SaaS can provide the sensor data at varying resolution while maintaining consistency within the training environment. A specific example of training includes the Simulation Training Environment (STE)

CFT, where the synthetic environment includes a constructive model integrated with a series of Man in the Loop (MITL) simulations, all of which use a common architecture and protocol. The implementation of sensor as a service ensures the sensor representation is consistent and accurate across all platforms.

Development Test (DT) and Operational Test (OT) activities have different goals. Development test is focused on the system and is conducted in a controlled environment. In many respects it is like analysis. Operational test focuses on testing the capability or suitability of a system within the operational environment. SaaS can be used to stimulate the test environment at the right level and allow the DT models to be the basis for the OT simulations.

SaaS supports acquisition decisions by providing appropriate resolution models throughout the acquisition lifecycle. Integrating simulations at the interfaces allows isolation and standardization of the input and output data for consistent sensor representation. SaaS also supports developing TTP and CONOPS, as well as Soldier User Assessment in a safe environment prior to costly systems development. Digital twins can be wrapped as a service to quickly represent sensors as designed, so design can be assessed prior to production.

Simulation supports all of the above communities through exercises. Using SaaS enables the exercise director to compose and trade simulation instances and fidelity to ensure mission success. Frequently, numerous instances of a simulation are needed to support higher fidelity and entity count. With the elasticity of cloud and cluster-based services, SaaS can efficiently balance both to provide low latency simulations that easily scale.

SENSOR AS A SERVICE MICROSERVICES ARCHITECTURE

The communities and acquisition process discussed above have been supported in simulation using Service Oriented Architectures (SOA). Microservices further decompose SOA concepts, and they are combined to provide simulation application services through orchestration. Microservices embody the Unix philosophy, to write programs that do one thing and do it well. The focus of microservices, as is the case for Unix programs, is on composability, where when additional features are desired, new microservices rather than extensions of current implementations are preferred. The re-use and composability of microservices improves system designer and implementer's ability to provide and integrate use case specific environments.

The microservices design pattern[ii] is well suited to the challenge of integrating disparate sensors in simulations. Most notably, the strong boundaries between microservices enables the decomposition of different sensors into different domains. It also mirrors the organizational structure of different labs, each having expertise and responsibility for different types of sensors. For example, the one lab has expertise in electro-optical and night vision sensors, another for radars, and another for acoustic sensors. The microservices approach enables each lab to independently develop and deploy software that simulates the effects of these different types of sensors. These models would be containerized and independently integrated by an interface that is typically based on HTTP standards or fast messaging systems. More subtly, the microservices approach allows each service to also isolate its data. For example, in the domain of terrain data, different characteristics of the terrain are important to different models. The radar model is concerned with terrain blocking the beam; the night vision model is concerned with terrain thermal properties; and the acoustic model is concerned with propagation of vibrations through the ground. So one overarching terrain data set would have difficulty meeting the needs of all three. In the microservices approach, each lab independently develops its own service, integrates with necessary data, and deploys behind a comparatively simple API that hides the complexity in the underlying domain, only publishing relevant data such as target detections.

However, independently deploying microservices is not sufficient for running a simulation model. That

requires a platform for the discovery, composition, and execution of these services in a simulation scenario. The NATO Science and Technology Organization developed the Modeling and Simulation as a Service (MSaaS) concept just for that purpose.[iii] In order to achieve the vision of "M&S products, data and processes are conveniently accessible and available on-demand to all users in order to enhance operational effectiveness," The MSaaS portal, shown in the center of Figure 1, plays a central role. Considering the example of integrating sensor services, the first step is to discover available services and data. Simulation engineers use the service registry to discover available sensor services along with documentation of their data requirements and APIs to enable integration. They then use the architectural building blocks of the reference architecture to compose sensor services with the other components of the simulation, such as terrain and combat models. The composition, along with available scenarios, is then added to the service registry so that it can be discovered and executed on demand by simulation users. This is possible because the service registry not only provides data about the stored assets, but also links to a repository, hosted in the cloud, to deploy and run the simulation, storing data and shutting down execution when complete.

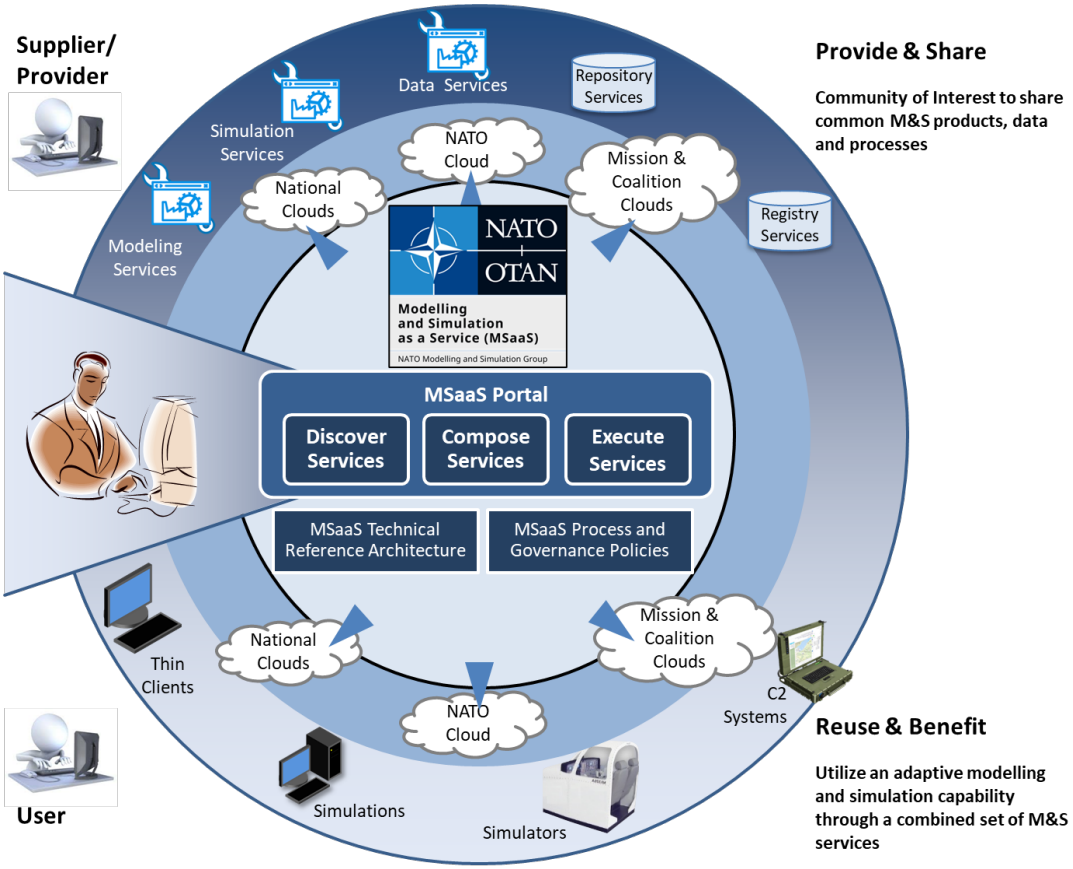

Figure 1: The NATO Modeling and Simulation as a Service (MSaaS) concept.

The US Army Futures Command C5ISR Center is developing a M&S as a service architecture to allow sensor models, communications models, cyber/EW models, and C2 models to cooperatively develop a common operational picture (COP), adjudicated by sensor and network effects, to present to shooter command and control (C2) nodes. An architecture is proposed, based in Integrated Sensor Architecture (ISA), to model the transmission of detection data along with its correlation and fusion into a realistic COP. The architecture will provide the capability to present a simulated secure intelligence data feed, using ISA, to simulated entities or to real C2 systems. C2 systems already compliant with ISA will be able to consume the data feed to support training or to support validation of intelligence and fusion capabilities embedded in those systems. The data can also support development, test, and validation artificial intelligence algorithms

that transform sensor data streams into actionable intelligence feeds. Finally, the data streams support intelligent behaviors in other simulated entities, such as route planning, sensor re-tasking, and sensor to shooter pairings.

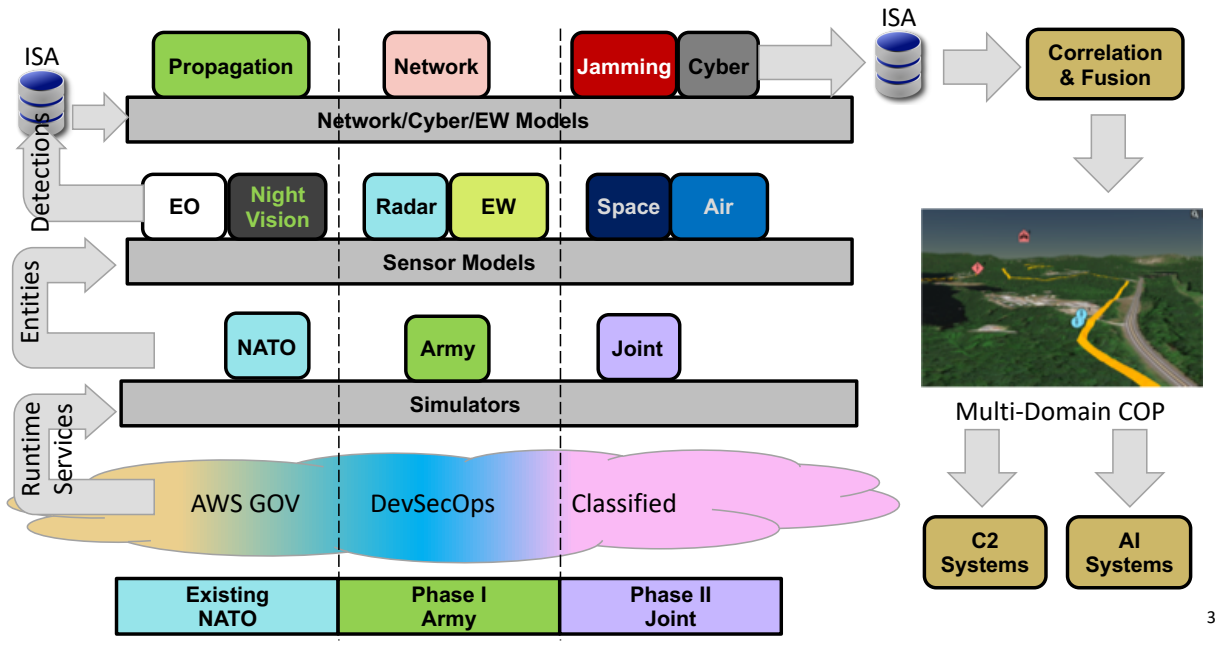

Figure 2: Architectural overview of Multi-Domain Sensor as a Service Concept.

The existing capabilities for this architecture, shown in Figure 2, were developed in concert with the NATO modeling and simulation group developing the MSaaS concept. Currently, the Army's EO and night vison models provide detections to the NATO Education and Training simulation environment. Additionally, a NATO radio propagation model computes radio link reliability before those detections can be propagated throughout the command hierarchy.

As the project moves into Phase I, it will use ISA as the backbone to share sensor data between sensor models and simulated C2 nodes. The plan calls for continuing to evolve the capabilities by adopting secure development and operations processes, integrating with Army simulation and radar/electronic warfare models, and expanding the network modeling capabilities. The final objective is to transition project to a classified cloud, integrate Joint sensor models, additional communications models, a validated jamming model, and a validated cyber effects model.

MICROSERVICES INFRASTRUCTURE

Development and implementation of SaaS follows de facto industry standard technologies. The key technology with our microservices implementation is the choice of Kubernetes (K8s)[iv] for container orchestration. Kubernetes is also microservices implementation, expressed as containers, which combine physical hosts, known as "nodes," into clusters capable of running other container workloads and services. Kubernetes clusters are available as a native offerings by the largest public cloud hosting companies, including Amazon's Elastic Kubernetes Service (EKS)[v], Microsoft's Azure Kubernetes Service (AKS)[vi], and Google's Kubernetes Engine[vii]. Kubernetes clusters can also be created using on-premise hardware, which only requires an operating system supporting container runtimes. While there are other choices for container orchestration, Kubernetes has dominated the market to become the de facto industry standard. Kubernetes has the broadest industry and open source support, established cloud services, and hence it is the lowest risk

technology for microservices implementation.

Another clear winner in the market has been Docker™ for their container technologies. Over time[viii], the dominance of Kubernetes has made this choice less of a commitment, as Kubernetes now relies upon containers implementing the Open Container Initiative (OCI) and Container Runtime Interface (CRI)[ix] standards. However, Docker continues to maintain presence, and given the available tooling, large user base, and portable declarations, Docker remains the obvious container technology choice.

Production container hosting is possible using common Linux distributions, container specialized Linux distributions, or recent Windows Server™ platforms. Windows™ support continues to mature, but a Kubernetes cluster still requires a Linux node for the control plane[x]. Production grade Kubernetes clusters are canonically bootstrapped with the "kubeadm" application[xi], although other tooling is available to simplify the cluster deployment. Consequently, production administrators have multiple Kubernetes deployment options including cloud Kubernetes services, cloud Infrastructure as a Service (IAAS), or on-premise hardware and operating systems.

For development purposes, there are a number of options to run a Kubernetes cluster on a single node such as a laptop. The Kubernetes tooling documentation[xii] identifies Minikube[xiii] for single node clusters, and it requires container or virtual machine manager on a host operating system. Should one need to support Windows containers, Docker Desktop[xiv] supports both Windows™ and Linux base images, can provide a single node Kubernetes cluster, and requires virtualization. Both development tools have been used during the development of SaaS, and our developer's preference is the Edge release of Docker Desktop[xv]. Regardless approach, there are many well supported options for Kubernetes orchestration of Docker containers, giving microservice practitioners multiple choices and flexibility for development and production.

NATO has also adopted Kubernetes container orchestration of Docker containers in their experimentation. Additionally, they have adopted Rancher[xvi] as the common platform for Kubernetes management. Standards have also been important in their integration of microservices. Key M&S standards include the Standard for Military Scenario Definition Language[xvii], the NATO Education and Training Network Distributed Simulation Federation Object Model[xviii], and the draft standard for Web Live, Virtual, Constructive (WebLVC) Protocol[xix]. Additionally, commercial technologies and standards for implementing services include Hypertext Transfer Protocol – Representational State Transfer (HTTP-REST)[xx] interfaces, websockets, Javascript Object Notation (JSON), and Google Protocol Buffers.

SENSOR AS A SERVICE INITIAL PHASE

A set of microservices providing the Night Vision Integrated Performance Model (NV-IPM) as a web service comprises the initial realization of SaaS. NV-IPM is a desktop software application that enables users to define a complete sensing chain from target and background, environmental effects, optics, focal plane arrays, signal processing, and display, through human vision, cognition, and ultimately perception. NV-IPM provides a thorough User Interface (UI) to compose the sensing pipeline, which enables Electro-Optics (EO) professionals great flexibility through parameterization to explore systems design and performance. NV-IPM also provides a "Simplified Interface" a reduction of parameterization to the most common use cases, which makes the tool viable for non-expert professionals. The SaaS implementation leverages this "Simplified" capability, and exposes it as a REST Application Programming Interface (API).

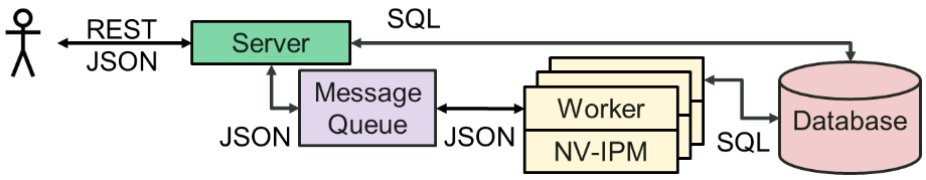

Figure 3: Sensor As A Service with NV-IPM

The current SaaS implementation comprises four microservices, the "Server," "Worker," "Message Queue," and "Database" as depicted in **Figure** . The Server microservice receives EO system performance requests at a REST URL where the content is a JSON document containing NV-IPM simplified parameters. The Server validates this JSON content utilizing JSON Schema[xxi], which is described in a set of IETF draft specifications. Assuming correctness, the original request is persisted via the Database microservice. In parallel, the JSON request document is sent to the Message Queue microservice. A set of Worker microservices pull requests from the Message Queue, apply NV-IPM logic to calculate results, and persist those results using the Database microservice. The user gets performance results from a result REST URL, where the Server transforms NV-IPM from the Database and returns it as another JSON document.

Beyond providing the REST web service for NV-IPM capabilities, SaaS enables concurrent execution of NV-IPM logic. A common use case for NV-IPM is to calculate the probability of detection as a function of sensor range to target, which requires iteration over a number of parameters. This calculation is time intensive, and the current NV-IPM application does not support concurrency. The desktop NV-IPM user either had to sequence their calculations or run multiple NV-IPM applications to generate multiple result sets. The current SaaS configuration declares multiple Worker instances to run concurrent NV-IPM calculations.

The Kubernetes based implementation of SaaS allows other microservice deployments to accommodate user demands. Kubernetes includes an Ingress, a public entry point into the cluster, which acts as a web load balancer. Consequently, multiple SaaS Server instances, stateless web services, can be instantiated to ensure that all user requests are accommodated. The other two SaaS microservices, the Message Queue and Database, are containerized deployments of NATS[xxii] and PostgreSQL[xxiii] open source technologies which support clustering. Therefore, each microservice of SaaS can be instantiated multiple times, so the system can be scaled as needed. The number of containers per microservice type is defined through Kubernetes Deployment configuration, which can be updated dynamically to accommodate changes in service demand. The only limitation to scale is the capacity of the Kubernetes cluster.

SENSOR AS A SERVICE OBJECTIVE IMPLEMENTATION

The containerization of each microservice provides encapsulation, where the use of language independent APIs over standard transports enables composition. In the original implementation, the PostgreSQL API was used for Server and Worker communications to the Database microservice. While specific to a technology, the PostreSQL API is supported across numerous programming languages, and moreover it is abstracted through the use of Object-Relational Mapping (ORM) libraries. These libraries not only contain drivers for many relational databases, but they also transform software runtime object instances to and from relational store tables.

The Message Queue is a tighter coupling with the supporting NATS technology, where the control (open, close, publish and subscribe) of the queue is a NATS specific API. However, the message queue payload is treated as a Binary Large Object (BLOB), which enables SaaS JSON documents to be passed. While the Database microservice can be replaced with alternate implementations with only configuration changes and library updates to paired microservices, the NATS Message Queue is technology specific and a replacement

would impact associated microservices. A messaging standard, Advanced Message Queuing Protocol (AMPQ)[xxiv] does exist, but AMQP does not have a standard API which means AMQP based microservices cannot be provider independent. Consequently, there is no standard message queuing choice, yet there are many competing implementations with multiple programming language bindings. In addition, there is no system constraint to use only one queue technology, so alternative queuing microservices can be introduced to facilitate the Inter-Process Communication (IPC) between other microservices. One example will be the ISA, which will be instantiated to communicate with external Command and Control (C2) applications. The SaaS system leverages this heterogeneous possibility to layer additional capabilities under a single REST API.

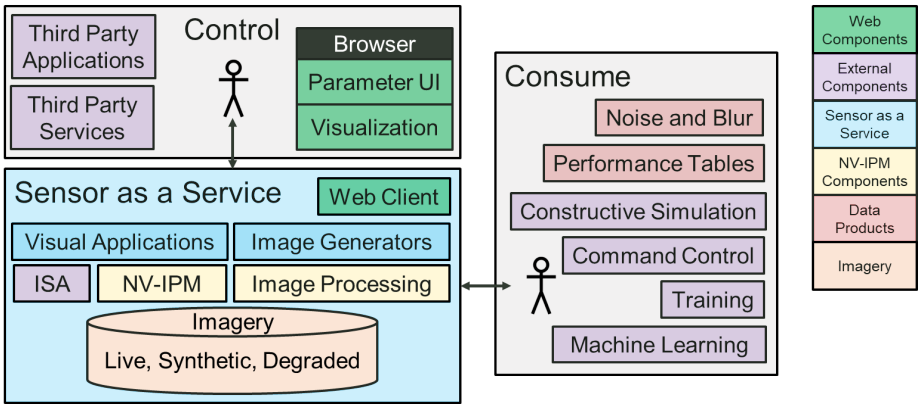

Figure 4: Sensor As A Service Objective Capabilities

The objective SaaS system, shown in **Figure** , extends beyond the initial NV-IPM based set of microservices. The "Control" grouping depicts anticipated third party applications and services that will utilize the SaaS, in addition to the SaaS developed web client application. The original NV-IPM library is reused, and it provides sensor noise, blur, and performance tables suitable for constructive simulation and training. Recently, NV-IPM has been extended with image transformation logic to transform ideal imagery and represent the targeted sensor system performance with degradations to this imagery. This new image processing relies upon external source imagery, typically from highly controlled live collections. Additional microservices, envisioned as an object store, will be added to SaaS to persist this source imagery with its attribution captured as metadata. The additional NV-IPM image degradation logic will become a separate microservice as it is implemented outside the current NV-IPM libraries and comprises a unique application. The imagery persistence microservices hosting the ideal imagery will also be used for these degraded imagery products. Moving beyond live imagery and image processing derivatives, image generators will be integrated as SaaS microservices to synthesize targets, backgrounds, and combined imagery products. Again, the imagery persistence microservices will host this generated content, so live, degraded, and synthesized imagery will be available. Moreover, SaaS will enable cooperation between synthetic generation and degraded imagery, where the superposition and combination of the two products will be possible. Ideally, there will be live capture of backgrounds combined with superimposed rendered targets with correlated thermal properties, to provide a hybrid product through a new microservice. These imagery products, performance metrics, sensor noise and blur, along with pairwise target – sensor Detect, Classify, Recognize, and Identify probabilities, will be made available through a single SaaS REST API.

CONCLUSION AND FURTHER WORK

The initial implementation of SaaS as microservices has benefited NV-IPM users, but also highlighted areas for improvement. The addition of a REST interface to NV-IPM and the enablement of concurrent runtimes through microservices orchestration demonstrated a scaled up NV-IPM environment suitable for AI/ML and other computationally demanding use cases. The current Kubernetes declarations enable a static number of

NV-IPM workers, but objectively, these workers should be automatically scaled on demand (cloud elasticity). Moreover, the current NV-IPM microservice contains all functionality of the application, which could be decomposed into multiple microservices. In theory, each element of the sensing chain from target and background, environment effects, optics, focal plane arrays, signal processing, and display, through human vision, cognition, and ultimately perception could be implemented as a distinct microservice. However, there is significant complexity in the composition of these elements, which has already been captured and matured within the NV-IPM application. Should the demand for NV-IPM compute services grow, this decomposition would be considered. However, the benefit of concurrency would need to surpass the investment to refactor, test and deploy.

Beyond the initial phase, the objective capability of SaaS remains to be implemented. Imagery degradation, the addition of synthetic content through image generators, and the integration of third party users are all challenges to be addressed. Key in this expansion will be the definition of the API, where technologies and paradigms aside from REST, such as gRPC[xxv] and GraphQL[xxvi], will be considered. Regardless of the mechanism, intuitive, normalized, and performant APIs will be crucial for SaaS user adoption and performance. Finally, as noted in the architecture discussion, there are combinations with other simulation elements, environments, and standards, so SaaS portability and inclusion into other work is a priority.

REFERENCES

[i] Brian Teaney, Joseph Reynolds, "Next generation imager performance model," Proc. SPIE 7662, Infrared Imaging Systems: Design, Analysis, Modeling, and Testing XXI, 76620F (22 April 2010); https://doi.org/10.1117/12.850876

[ii] Martin Fowler and James Lewis, "Microservices - a definition of this new architectural term," 2014, available at https://martinfowler.com/articles/microservices.html.

[iii] NATO Science and Technology Office, Modelling and Simulation as a Service (MSaaS) – Rapid Deployment of Interoperable and Credible Simulation Environments, Technical Report, 2019.

[iv] "What Is Kubernetes?," https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[v] Amazon Elastic Kubernetes Service, https://aws.amazon.com/eks/

[vi] Azure Kubernetes Service (AKS), https://azure.microsoft.com/en-us/services/kubernetes-service/

[vii] Google Kubernetes Engine, https://cloud.google.com/kubernetes-engine/

[viii] "A Comprehensive Container Runtime Comparison," https://www.capitalone.com/tech/cloud/container-runtime/

[ix] Open Container Initiative, https://opencontainers.org/

[x] "Adding Windows Nodes," https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/adding-windows-nodes/

[xi] "Bootstrapping clusters with kubeadm," https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/

[xii] "Install Tools," https://kubernetes.io/docs/tasks/tools/

[xiii] Minikube, https://minikube.sigs.k8s.io/docs/

[xiv] Docker Desktop, https://www.docker.com/products/docker-desktop

[xv] "Docker Desktop for Windows Edge Release notes," https://docs.docker.com/docker-for-windows/edge-release-notes/

[xvi] Rancher, https://rancher.com/

[xvii] Simulation Interoperability Standards Organization, Standard for Military Scenario Definition Language, SISO-STD-007-2008, 11 May 2015.

[xviii] NATO Science and Technology Organization, NATO Distributed Simulation Federation Object Model (NETN FOM), https://github.com/AMSP-04.

[xix] Simulation Interoperability Standards Organization, Standard for WebLVC Protocol, SISO-STD-017-DRAFT, 2020.

[xx] "Representational state transfer," https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[xxi] JSON Schema, http://json-schema.org/

[xxii] NATS, https://nats.io/

[xxiii] "PostgreSQL: The World's Most Advanced Open Source Relational Database," https://www.postgresql.org/

[xxiv] Advanced Message Queuing Protocol, https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol

[xxv] "gRPC, A high-performance, open source universal RPC framework," https://grpc.io/

[xxvi] "GraphQL, A query language for your API," https://graphql.org/