H4H: An Open Framework for Rapid Human–Machine Teaming Prototype Integration

Edgar F. Martinez, Rebecca M. Crockett, and Ian A. Grissom

ABSTRACT

Effective human–machine teaming systems are becoming critical to the success of the modern warfighter. However, these systems are traditionally costly to develop because of the complex integration of hardware and software components from disparate organizations and vendors. In response to this challenge, Johns Hopkins University Applied Physics Laboratory (APL) researchers developed the Human–Machine Interfaces for Human–Machine Teaming (H4H) architecture, a platform that simplifies this complex integration. By modularizing components and simplifying interfaces, H4H aims to reduce the overall cost to establish an initial prototype of a system while enabling reuse of the system's components.

INTRODUCTION

APL develops and evaluates new human–machine teaming (HMT) technologies to augment the modern warfighter and to enable the United States to hold a competitive edge over current and future adversaries. HMT refers to collaboration between humans and complex technologies to achieve a specific goal, and it relies on three equally important elements: the human, the machine, and the interactions between them. Because HMT innovations are often a novel combination of existing technologies, components are frequently rapidly prototyped and integrated so that new ideas can be evaluated before too much time or money is invested in creating a final design. However, improvements to current HMT prototyping practices could lead to even more savings.

In general, developing prototypes reduces costs because there is no concern with future modifications, extensions, or reuse when integrating components. Although this practice makes sense for most proof-of-concept efforts, HMT prototypes could benefit from these characteristics since many HMT concepts of operations share hardware and software components. Making these components modular and reusable could save costs across multiple prototyping efforts.

Consider the following small-scale concept of operations: a warfighter and an unmanned ground vehicle (UGV), where the vehicle acts as scout that reports nearby threats. The warfighter is equipped with an augmented-reality (AR) display that visualizes the threats detected by the UGV. Today, most prototype development teams would purchase an AR display and a viable UGV after a brief market survey and then integrate these products. However, the initial system components selected may need to be changed. For example, the warfighter's environment may change to terrain that would require using an unmanned aerial vehicle (UAV) instead of a UGV, or perhaps the development team is asked to switch AR display vendors. In most cases, short-term time and cost savings will discourage the team from considering these scenarios. As a result, the initial system implementation will be tightly integrated, and any changes will incur reintegration costs.

To avoid these common pitfalls of prototyping, the Human–Machine Interfaces for Human–Machine Teaming (H4H) architecture facilitates component substitution and reuse while adding minimal design and implementation overhead. H4H's ultimate goal is to lower the time and cost required to integrate, modify, and extend HMT prototypes.

H4H OVERVIEW

H4H prototypes are built as a collection of small components serving distinct purposes. H4H's overall design goals are to ensure that these components can be replaced or updated as needed, to simplify or abstract the interfaces between them, and to enable reliable or semi-reliable transport of data across them. To achieve these goals, H4H uses an open microservices architecture and a publish-subscribe messaging framework built for distributed components communicating over User Datagram Protocol (UDP). The following subsections summarize the concepts and the reasoning behind the decisions that led to the final H4H design.

Open versus Closed Architectures

System architectures are categorized as open or closed. In a closed system architecture, components are predefined, and their interactions consist of proprietary protocols tailored to those specific components. Designing the interactions between specific components, however, usually leads to a tight coupling between components and makes the system hard to modify or extend. In contrast, open architectures are designed with consideration that existing components will possibly be replaced or new ones introduced in the future. Instead of using proprietary protocols and knowledge of the implementation details of other components, the interactions between an initial set of components in an open architecture will take place through abstract interfaces and standardized protocols. Designing interactions this way enables components to remain loosely coupled and modular. Since HMT technologies are constantly evolving, it is impossible to predefine all the desired components in an HMT system. Therefore, when considering rapid prototypes in the field of human–machine systems, it is essential to consider the use an open architecture to facilitate integration while maintaining modularity for future development.

Microservices

With modular components, the functionality of one component does not depend on the implementation details of another component. This separation of logic is the main idea behind the widespread use of microservices architectures. In a microservices architecture, software applications can be decomposed into several loosely coupled single-function components referred to as services. This functional organization enables each individual service to be highly maintainable and easily testable. Additionally, services can be tailored to specific capabilities and to be owned by independent development teams.[1] Since HMT prototypes often use components developed by different groups, H4H's microservices architecture has the potential to increase the development speed of HMT prototypes. Instead of dwelling on implementation and delivery details, teams will need to agree only on the inputs and outputs of their services. Then, they can work in parallel with minimal interdependencies across components.

Publish-Subscribe Messaging

Data exchange in HMT systems tends to be event driven. For this reason, H4H uses the publish-subscribe messaging model to create abstract interfaces between components. Publish-subscribe messaging involves channels in which one or more publishers can publish information about an event. When information is published to a channel, all of the channel's subscribers receive the information. In H4H, channels have predefined data types to serve as an abstract interface through which any number of publishers and subscribers (i.e., components) can simultaneously interact. This quality makes extensibility easier because new components need only adhere to the data type of each channel to interact with other components.

H4H Architecture Applied

As an example, H4H is applied to the concept of operations of the warfighter with an AR headset and a UGV (Figure 1). An adapter service is created and deployed for each external hardware component (UGV and AR display). In this context, an adapter service is essentially software that acts as an interface between an external component and an internal data channel. Adapter services are therefore tightly coupled to their respective hardware but not to any other service. The threat detection service is also decoupled from all hardware. It consumes the data from the UGV adapter through the sensor data channel and provides the AR display service with data through the threat alert channel. As a result, if the UGV needed to be swapped for a UAV, then a UAV adapter would replace the UGV adapter and no other services would need to be modified. Similarly, if the AR display's vendor changed, then only the AR display adapter would need to be changed. This approach isolates the impact of changes to a single cohesive module, which simplifies the development and expedites the integration of a new component. The use of publish-subscribe channels also adds the ability to seamlessly scale the system, meaning adapter services developed for different external hardware components, such as two different AR displays, could be simultaneously deployed with significant ease. Ultimately, it provides prototyping teams with the ability to quickly test and evaluate alternatives or vendor products.

IMPLEMENTATION

To meet the aforementioned design goals, the H4H team selected open-source and commercial off-the-shelf technologies. These technologies have large user communities and are more likely to have existing support or publicly available documentation that facilitates integration with our platform.

Many kinds of services could be introduced into the architecture. Some examples are a database, a machine learning model, or an adapter for a new hardware component. Regardless of the functionality they provide, all services are essentially software that any prototyping team using H4H should be able to deploy. For this reason, H4H uses Docker, a containerization platform for packaging applications into standardized executable components called containers.[2] Docker enables service functionality to be wholly contained within a modular, shareable package in a way that is largely independent of the compute environment. If the packaged application works on one machine with Docker Engine, it should work on any other machine with a matching version of Docker Engine and the same CPU architecture (e.g., x86, ARM64). Packaged applications ready for execution, referred to as container images, can be shared in a repository and downloaded on demand. Docker enables a long-term solution for creating, managing, and storing version-tracked capabilities across efforts and projects.

Complementing the ability to containerize each service for easy execution, Kubernetes[3] provides a suite of powerful tools for managing the networking, monitoring service health, or creating the support infrastructure for the deployed containers. Kubernetes uses a series of resource files to define how a container is deployed and how its life cycle is managed thereafter. The Kubernetes Control Plane implements this functionality through a set of core services that respond to cluster events. A Kubernetes cluster is composed of one or more worker machines, referred to as nodes, that can share the workload. H4H deployments will run a single cluster in a single edge-compute device. When edge-devices are networked together, communications are managed as cluster-to-cluster, rather than node-to-node. This design choice is necessary to enable the Kubernetes Control Plane, which manages the service deployments, to run independently on each device. The streamline installation and setup, Canonical, the maintainers of Ubuntu, created MicroK8s.[4] MicroK8s is a powerful, lightweight, production-ready Kubernetes distribution. As a whole, it has been designed with edge-computing in mind and has a small disk and memory footprint. H4H currently supports simple and easy-to-use installation scripts built on MicroK8s.

HMT capabilities are expected to be deployed as a collection of services (Figure 1) using Kubernetes resource files. For larger projects integrating multiple capabilities, resource files can become unwieldy. This is amplified in a highly modular HMT system where varying configurations of several or individual services are possible depending on the operational need. To streamline this process, H4H uses Helm, a Kubernetes package manager.[5] Helm users create "chart" files that allow a collection of Kubernetes resources to be packaged together for

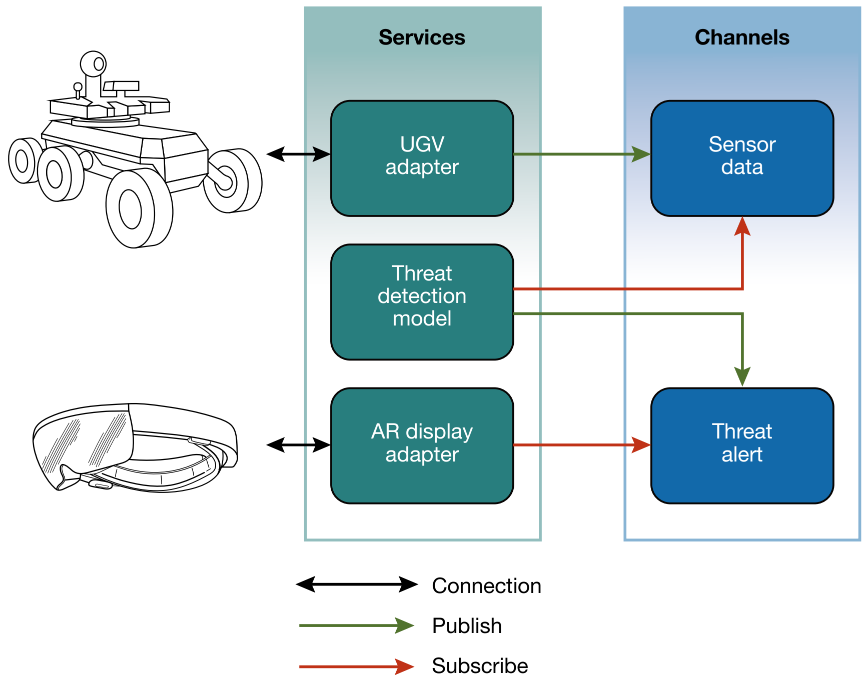

Figure 1. H4H architecture. Services in the architecture interact through publish-subscribe channels with predefined abstract data types, allowing services to remain decoupled. Replacing a service requires only that the replacement adhere to the predefined abstract data type of the channels it will interact with.

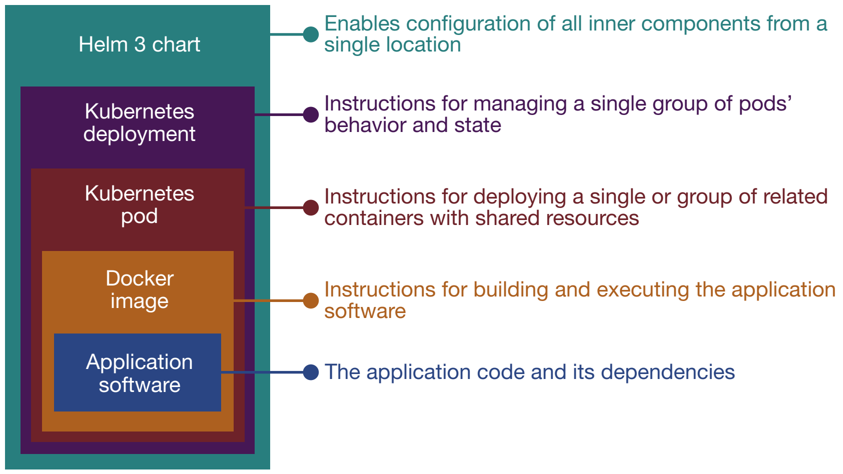
Helm 3 chart
Kubernetes deployment
Kubernetes pod
Docker image
Application software
Enables configuration of all inner components from a single location
Instructions for managing a single group of pods' behavior and state
Instructions for deploying a single or group of related containers with shared resources
Instructions for building and executing the application software
The application code and its dependencies

Figure 2. The multiple layers of configuration provide Docker and Kubernetes the instructions to build, execute, monitor, and manage applications. Once a capability is wrapped in a Helm chart, it can be shared and reused across projects with little to no effort.

easy deployment. Charts offer two key features: version tracking and templating. Version tracking enables capabilities to be incrementally versioned as changes are made, while templating enables users to supply a single "values" file containing deployment-specific settings to base templates of Kubernetes resource files. Templating is a key feature H4H uses to deploy applications across different systems. Figure 2 shows how Docker, Kubernetes, and Helm wrap an application. Each layer provides some abstraction or control over the layers beneath it.

All these tools function together to provide a method for deploying modular service elements. Meanwhile, the Robot Operating System 2 (ROS 2) provides the framework for communication between services.[6] By default, ROS 2 uses the Data Distribution Service standard for messaging. DDS is standard backed by the Object Management Group that aims to enable dependable, high-performance, scalable data exchange over UDP. ROS 2 allows developers to define message types, which can be shared and versioned similarly to packaged applications, as well as to create publish-subscribe channels, named Topics, among many other things. H4H uses this functionality to define the abstract interface through which services communicate. A message type defines the data structure, while the Topic provides the medium. In the UGV example, the Threat Alert and Sensor Data Topics would have different message types (Figure 1). If each service were developed by a different team, the teams would need to be aware of only the message types they will produce or consume to integrate with other teams.

CONCLUSION

H4H provides a framework that expedites the integration of HMT prototypes. Specifically, it uses a microservices approach to separate components and provides methods for abstracting component interfaces. These features enable modularity. As a result, adding, upgrading, replacing, and reusing hardware and software components is more efficient. Although still in its early development stages, H4H has already shown promising results on an initial test bed integrating AR displays, UGVs, UAVs, gesture controllers, and more. The H4H team plans to continue refining the H4H framework and expanding its set of tools and features in further pursuit of the end goal of augmenting tomorrow's warfighter capabilities.

ACKNOWLEDGMENTS: This paper was written in 2021, and H4H has grown since that time. Matthew Breiner (matthew.breiner@jhuapl.edu) or Brandon Filo (brandon.filo@jhuapl.edu) can be contacted for updates.

REFERENCES

[1]P. Richardson. "What are microservices?" microservices.io. https://microservices.io/ (accessed Nov. 22, 2021).
[2]"Empowering app development for developers." Docker. Docker, Inc. https://www.docker.com/ (accessed Nov. 22, 2021).
[3]"Production-grade container orchestration." Kubernetes. https://kubernetes.io/ (accessed Oct. 7, 2021).
[4]"MicroK8s—Zero-ops Kubernetes for developers, Edge and IOT: MicroK8s." microk8s.io. Canonical Ltd. https://microk8s.io/ (accessed Nov. 22, 2021).
[5]"The package manager for Kubernetes." Helm. https://helm.sh/ (accessed Nov. 22, 2021).
[6]"ROS documentation." Open Robotics. https://docs.ros.org/ (accessed Sep. 28, 2021).

Edgar F. Martinez, Research and Exploratory Development Department, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Edgar F. Martinez is a software engineer in APL's Research and Exploratory Development Department. He holds a BS in computer science with minors in cybersecurity and astrophysics from Texas A&M University. As part of the rotational Discovery Program, he has rotated through multiple teams at APL and contributed to projects related to full-stack web development, modeling and simulation, and data visualization. During his rotational assignment with the Cyber-Physical Systems Development Group, he developed introspections tools and services to facilitate test and evaluation efforts for the H4H project. His email address is edgar.martinez@jhuapl.edu.

Rebecca M. Crockett, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Rebecca M. Crockett was a process engineer in APL's Asymmetric Operations Sector. She holds a BS in chemical engineering with a concentration in cybersecurity from the University of Tulsa. During her time at APL, Rebecca contributed to a wide variety of projects with focuses ranging from sensor fusion to robotics to human–machine teaming and augmented reality. Rebecca was the assistant program manager on the H4H effort.

Ian A. Grissom, Asymmetric Operations Sector, Johns Hopkins University Applied Physics Laboratory, Laurel, MD

Ian A. Grissom was a software engineer in APL's Asymmetric Operations Sector. He holds a BS in electrical engineering from the University of Maryland, College Park and an MS in electrical and computer engineering from Johns Hopkins University. During his time at APL, Ian contributed to a wide variety of projects with focuses ranging from safety critical embedded software development to robotics to software architecture building and software reverse engineering. Ian was the project manager and a technical contributor on the H4H effort.