An Introduction to the Joint Architecture for Unmanned Systems (JAUS)

Steve Rowe
Christopher R. Wagner
Cybernet Systems Corporation
727 Airport Blvd.
Ann Arbor, MI 48108
734-668-2567
srowe@cybernet.com, cwagner@cybernet.com

Keywords:
JAUS, Unmanned Systems, Robotics, Standards.

ABSTRACT: *The Joint Architecture for Unmanned Systems (JAUS) standard is a messaging architecture enabling communication with and control of unmanned systems (including, but not limited to unmanned air, ground, and sea vehicles). JAUS is being developed by a working group of military and volunteer commercial organizations, and will be released as a Society of Automotive Engineers (SAE) standard. Unlike the NATO STANAG 4586 standard, JAUS addresses unmanned system capabilities beyond those of aerial vehicles, including payload control (e.g. manipulators), autonomous systems (as opposed to teleoperated), and weapons systems.*

The SISO RD&E group has requested an introduction to this emerging standard; This paper provides an introduction to JAUS, including history, underlying design principles, current status, and future direction.

1. Introduction

1.1 Motivation

Robots are now an accepted part of our military force, deployed on missions including surveillance, bomb disposal, and inspection. The Army has plans to deploy armed robotic vehicles (ARVs) and materiel carriers in the immediate future. In 2002, The United States Senate mandated that 30% of all military vehicles be unmanned by 2015[1]. There are between four and five thousand robots deployed today, and that number will continue to increase as the Future Combat Systems (FCS) initiative is deployed.

At the same time, the Department of Defense (DOD) is working to integrate all branches of the military, requiring formerly incompatible systems to communicate constructively. Part of this sweeping standardization initiative to create a joint force is the creation of a standard to allow the interoperability of unmanned systems.

1.2 History

In 1998, the Office of the Under-Secretary of Defense (OUSD) for Acquisition, Technology and Logistics Joint Robotics Program commissioned a working group to create a standard for interoperability of unmanned ground vehicles (UGVs)[2]. The standard was then called JAUGS (Joint Architecture for Unmanned Ground Systems), but later was generalized to unmanned systems of all sorts, including surface, submersible, aerial, and non-vehicular unmanned systems.

In 2005, JAUS was adopted by the Society of Automotive Engineers (SAE) as a standard under the auspices of its aerospace standards division. The full name of the standard is now AS-4/JAUS (for brevity, this paper will simply use "JAUS").

The current version of the standard is 3.2, but the version 3.3 draft standard is in ratification by the JAUS Working Group (JWG) at the time of this writing. This paper discusses version 3.2, but upcoming features will be discussed in section 8.

1.3 Working Group Structure

The JWG is a primarily volunteer organization with representatives from the military, industry, and academic communities. It is organized as a shallow hierarchy with a chairman, secretary, treasurer, and subcommittee leads. There are currently about 85 members representing over 30 organizations in the group. Approximately half of the membership represents the development community and the other half represents the user community. Within the JWG are a number of subcommittees that focus on a particular area of the technology (e.g. message transport, overall architecture, experimentation). JWG members may participate in as few or as many subcommittees as they desire. Changes to the standard are incorporated via a proposal/discussion/vote cycle.

2. JAUS Architecture

2.1 Overview

The JAUS standard is divided into two parts: the Domain Model (DM) [3] and the Reference Architecture (RA) [4]. The DM provides the motivation and goals for JAUS, while the RA provides engineering specifications. The DM can be summarized as "Provide a mechanism by which unmanned systems of any type can interoperate, and that can be rapidly inserted into the military and commercial space." The RA specifies an architecture framework, a message format definition, and a set of standard messages used to communicate among components. JAUS is not a communication protocol; it does not define (or limit) the mechanisms used for message transport, nor does it specify how the standard must be implemented. The remainder of this paper will expand on the concepts from the RA.

2.1 Architecture Framework

JAUS specifies a hierarchical organization as shown in Figure 2.1. A *System* is composed of Subsystems. A *Subsystem* is a self-contained entity such as a robot or an operator control unit (OCU). Subsystems are composed of *Nodes*, which roughly correspond to individual computers. On each node, one or more Components reside. A Component is an entity (usually software) that performs a specific function or provides a specific service, for example a sensor or motor driver.

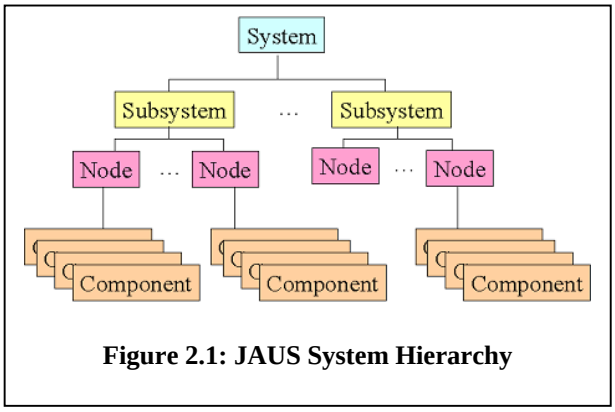

Figure 2.1: JAUS System Hierarchy

Components are referred to by their JAUS Address, a quadruplet of the form: **`<SubsystemID>.<NodeID>.<ComponentID>.<InstanceID>`** that uniquely identifies every component in a system. The subsystem, and node ID numbers are arbitrary integers in the range 1 to 254. Component ID numbers, on

the other hand, are specified in the RA for every defined component type. The Instance ID is provided in the case where multiple instances of the same component type reside on the same node.

There are two components of particular importance: the Communicator and the Node Manager. There is exactly one communicator per subsystem. It is responsible for communicating JAUS messages into and out of its subsystem. Since a subsystem can contain several nodes, the communicator must have some mechanism for routing messages it receives to and from each physical node. Once a message is directed to a node, the node manager for that node is responsible for routing the message to an individual component that resides on that node.

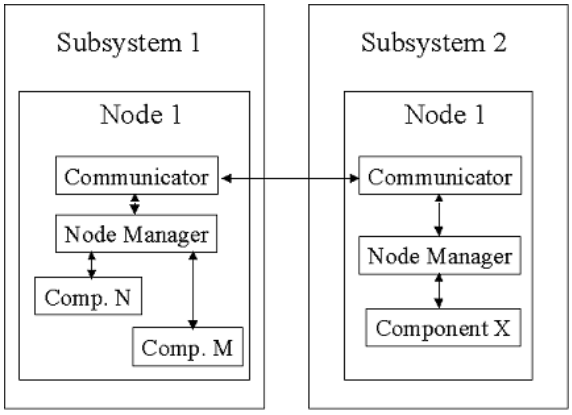

Figure 2.2: Communication within a JAUS System. Note that components do not communicate directly with each other, but rather through the node manager. Messages that pass outside of a subsystem boundary are routed by the Communicator.

Examples of other defined components include Mission Planner, Pose Sensor, and Waypoint Driver.

2.3 Levels of Compliance

The JAUS Compliance Specification [5] defines three levels of compliance that affect how a JAUS system is implemented. Level I compliance (the weakest level) requires JAUS compatibility at the subsystem level. That is, that there is a communicator component with an appropriate JAUS address, and that the subsystem implements a set of "core" messages (see section 2.5). This level of compliance is intended to provide a mechanism whereby legacy, non-JAUS platforms and controllers can be made to interoperate.

Level II compliance further requires that each node have a node manager and implement more of the "core" messages. This level is intended to allow robotic devices, such as manipulator arms, mobility platforms, and sensor packages to be assembled into a JAUS subsystem without restricting the underlying implementation.

Level III compliance requires JAUS compatibility at the component level. That is, each functional block of the system must have a way to send and receive JAUS messages through its node manager, and must implement a set of "core" messages. As of RA version 3.2, the mechanism whereby messages are communicated between components is not defined.

3. JAUS Message Format

All information in a JAUS-compliant system is communicated in the form of messages, which are variable-length sequences of bytes. Information about how to decode a message is contained in a fixed-length message header. Some fields in some messages are optional, and their presence or absence is indicated by a bit in a "presence vector" field in that message's payload.

3.1 Message Classes

There are 7 classes of message: Command, Query, Inform, Event Setup, Event

Notification, Node Management, and Experimental. These distinctions are made so that users can more easily organize their messages and find a particular message quickly in the specification; a message's class does not impact its format in any way.

3.2 Message Header

All JAUS messages begin with a 16-byte header (shown in Table 3.1) that defines characteristics common to all messages

Table 3.1 – The JAUS Message Header

Field#	Description	Data Type	Size (Bytes)
1	Message Properties	Unsigned Short	2
2	Command Code	Unsigned Short	2
3	Destination Instance ID	Byte	1
4	Destination Component ID	Byte	1
5	Destination Node ID	Byte	1
6	Destination Subsystem ID	Byte	1
7	Source Instance ID	Byte	1
8	Source Component ID	Byte	1
9	Source Node ID	Byte	1
10	Source Subsystem ID	Byte	1
11	Data Control (bytes)	Unsigned Short	2
12	Sequence Number	Unsigned Short	2
	Total Bytes		**16**

In particular, each header contains the JAUS address of the source and destination, a unique command code that identifies how the message payload is to be decoded, and a sequence number used to ensure that messages are not missed or received out-of-order.

The Message Properties field contains housekeeping information about the message, such as its priority, whether or not it requires acknowledgement, the version of JAUS that the message is compliant with, and a number of bits reserved for future use. The length of the payload is encoded in the Data Control field. Note the lack of a timestamp; this is discussed in section 7.3.

3.3 Message Payload

Although JAUS messages are variable-length, their format is very rigidly defined for each message type. Volume II of the RA specifies the data types used in the payload (down to the bit-ordering), the coordinate frames used, and the units for each field.

4. Defined Messages

One way that JAUS helps to ensure interoperability is by defining messages that are used for a particular purpose. For instance, there are many ways that one might implement a message that reports the robot's location. By specifying the exact message format for how locations are to be communicated, JAUS prevents developers from re-inventing that wheel. This leads to naturally interoperable systems at the expense of potentially increased development time if the appropriate message uses units other than those of the native system. If a developer discovers a need for a message that is not defined in the JAUS standard, he can create a new message in the Experimental class. The caveat is that a system cannot be claimed "JAUS compliant" if it requires the use of experimental messages; Therefore, developers are encouraged to submit their experimental messages to the working group for inclusion in the RA. The RA 3.2 defines over 100 standard messages. The version 3.3 draft RA defines about 50 more. As new technologies are developed, new messages will be added.

5. An Example

In this section, we discuss how the JAUS standard was applied in the creation of the control software for the Armadillo, Cybernet's entry into the Defense Advanced Research Projects Agency (DARPA) Urban Challenge.

5.1 Structure

The Armadillo is a Level I compliant autonomous vehicle. Its control system consists of three computers, interface electronics, and numerous sensors and actuators. The software structure for Node 2 is shown in Figure 5.1.

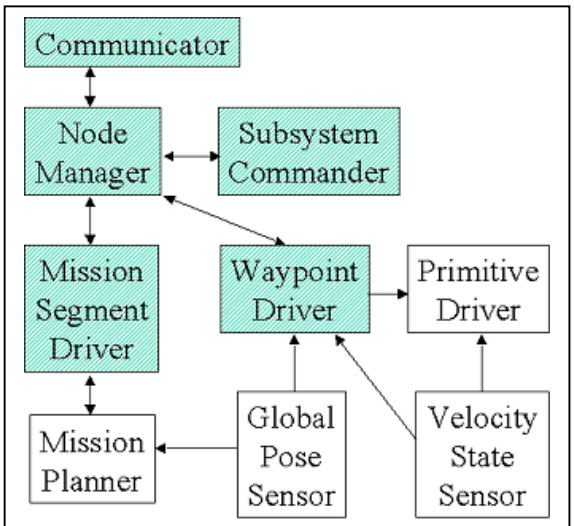

Figure 5.1: Component Structure for Armadillo. Shaded boxes are JAUS Components. The presence of non-JAUS components disqualifies Armadillo from Level 3 Compliance.

The Communicator receives messages from an external entity, in this case, the remote Operator Control Unit (OCU). The OCU issues high level commands such as Run, Pause, and Emergency Stop. It also sends a "heartbeat" message that ensures the vehicle is in contact with the control unit. We implemented the communicator with UDP/IP, although TCP/IP, RS-232, or some other serial protocol would serve as well. Incoming messages are handed from the Communicator to the Node Manager. The Node Manager examines the JAUS address of the recipient, and then places the message in an input queue on the appropriate component. The inter-component structure is established at system initiation, with each component making an explicit registration call on the Node Manager.

The Subsystem Commander component is responsible for maintaining the overall state of the subsystem (i.e., the vehicle). In response to a "Run" message, it dispatches messages to the other components, orchestrating their cooperation.

Although the control computer (Node 2) is JAUS-compliant at the Node level (Level II compliant), the other two computers in the vehicle subsystem are not JAUS-compliant at all. Consequently, Armadillo is only Level I compliant. The utility of JAUS within Armadillo is that it provides a mechanism for external control via JAUS-compatible control devices, and also that it provides a natural modular structure to the high-level software tasks. The original intention was to develop Armadillo as a Level III compliant subsystem; we found that our need for real-time data response from our sensor sub-systems conflicted with the unnecessary overhead incurred by all messages being routed through the node manager. By choosing this approach, we gained some amount of extra efficiency at the expense of some modularity and aesthetic purity.

6. Advantages

6.1 Modularity

JAUS encourages modularity by defining strict partitioning of subsystems into software

components. The interface to these components is narrow, i.e. a single point of connection to the Node Manager. The messages that can be processed by each component are well defined, as are the side-effects of processing a message (such as changing the speed of the platform).

6.2 Reusability

As a result of the modularity of design, individual JAUS components are good candidates for software reuse. For less application-specific components, this is certainly true; for instance, the communicator and node-manager components of Armadillo are completely domain-independent and could be used without modification in other JAUS subsystems. In fact, our OCU program uses the same Node Manager and Communicator software as the vehicle itself, without any modification.

Application specific components, such as the Primitive Driver, cannot be used in another robotic platform without re-writing the low-level hardware implementation. The utility of reusing such components is therefore questionable. However, we found that the core JAUS Component structure was highly reusable, and implemented it using the Object-Oriented principle of Inheritance.

6.3 Interoperability

One of the key goals of JAUS is to promote interoperability among robotic platforms and controllers in the military. Since all JAUS-compliant subsystems operate on the same message set, interoperability seems to be assured. Indeed, an OCU developed by another vendor could be used to operate the Armadillo with no modification, provided that it communicates on the same IP port and address. This reveals a weakness in the JAUS specification: Because JAUS does not define the transport mechanism whereby messages are communicated between subsystems, nodes, and components, the current JAUS standard *does not guarantee interoperability*.

This deficiency is of concern, and the JWG is working to address it and the other limitations discussed in the next section.

7. Limitations

7.1 Rigid responsibility

For every component, JAUS defines the messages that that component must process. For example, the Primitive Driver must process the Set Wrench Effort message, which specifies the power sent to each actuator as a percentage of its capability. The set of defined components can be (and has been) assembled into working subsystems. However, there are numerous ways that one could partition the functionality of a subsystem. Being JAUS-compliant means that the developer's design choices will be limited, which may hamper new creative solutions.

7.2 Required *a priori* knowledge

As noted in section 6.3, two JAUS-compatible devices are not necessarily interoperable; the communication protocol must be known in advance. Likewise, the assignment of subsystem ID and node ID must be known in order to properly address a message into a subsystem.

Further, the components contained in a subsystem must be known in order for any practical interaction to take place. For example, one must know if a subsystem has a Waypoint Driver before it can consider sending "Set Waypoint" commands.

Because this knowledge is required, current implementations of JAUS control units tend to be highly configurable. Configuring a

device is considerably easier than having to do new implementation, but is certainly not the turn-key solution desired.

7.3 Lack of real-time facilities

One of the most glaring omissions in the current JAUS specification is any notion of real-time responsiveness. Although messages may be sequenced and tagged as requiring acknowledgement, there is no temporal component. In robotic systems in general, and especially automated weapon systems, correct timing is absolutely critical. A command that is perfectly reasonable one second may be disastrous if executed at a later time.

The JWG has recognized this shortcoming, and is working to address it in an upcoming version of the standard. However, it is impractical to dictate minimum latency when the underlying hardware and communication protocol are not specified. This leads to a dilemma: should the JAUS standard dictate the implementation of the communication layer (leading away from its goal of platform neutrality) or purposefully omit things that it cannot guarantee (leading away from the goal of universal applicability)? This question is one that the JWG will continue struggling with for the foreseeable future.

7.4 Acceptance

Practically speaking, no standard can succeed if it is not generally accepted in the development community. The current robotics climate is one in which each developer generally creates proprietary interfaces for its subsystems; interoperability does not promote profit. If Acme Robotics has a contract to deliver some robots, having those robots able to be controlled by another vendor's OCU is bad for Acme, assuming that they also sell their own OCU. It is only pressure from the customer (e.g., the Army) that will cause vendors to include interoperability as a feature.

There is some development cost in making a system JAUS compliant. As we discovered in our own work, universal application of JAUS would have been more expensive and resulted in a less efficient system than the one that we implemented.

There are a number of competing standards in the domain. The North Atlantic Treaty Organization (NATO) developed a Standard Agreement (STANAG) to allow the standardization of control of unmanned aerial vehicles (UAVs). There is substantial overlap between STANAG 4586 and JAUS in terms of scope, although JAUS goes well beyond the former in terms of general applicability. However, STANAG 4586 is widely accepted and mature, so the probability of NATO changing its UAVs to use JAUS is vanishingly small. The situation is similar for STANAG 7085 (image data interoperability), MIL-STD 1760 (weapon systems interoperability), and AAST F-41 (unmanned undersea vehicle control). It would be beneficial if all of these standards could be fused into one, overarching interoperability mechanism, but that seems unlikely.

8. Future Direction of JAUS

The JWG continues to extend the message set and component repertoire of JAUS, making it increasingly useful. As developers speculate on the JAUS standard, they will submit their new messages and components to the JWG for standardization, making JAUS more attractive to the next iteration of development.

There is a development group within the JWG working to create a service-oriented architecture (SOA) built on top of the existing message and component structure that will substantially enhance the flexibility of JAUS

subsystems. This initiative includes features such as dynamic service discovery, and guaranteed service semantics. These capabilities will complement the existing JAUS standard, rather than replacing it.

9. References

[1] United States Senate, Fiscal Year 2001 Defense Authorization Bill, Sec 217.

[2] The JAUS Working Group Website, http://www.jauswg.org/ working_group/working_group.shtml

[3] The JAUS Domain Architecture Document, 2005. http://www.jauswg.org/ baseline/Domain Model v3.2 10Mar05.doc

[4] The JAUS 3.2 Reference Architecture Specification, http://www.jauswg.org/ baseline/refarch.html

[5] The JAUS Compliance Specification version 1.2, http://www.jauswg.org/baseline/ CS V1.2 25 OCT 06.doc

Author Biographies

STEVE ROWE is a Research Engineer at Cybernet Systems Corporation in Ann Arbor, MI. He is the software architect for Cybernet's autonomous vehicle, and a member of the JAUS working group.

CHRISTOPHER R. WAGNER is a Research Scientist at Cybernet Systems Corporation in Ann Arbor, MI. He holds a Ph. D. in Engineering Science from Harvard University. Dr. Wagner's current work involves creating a message translator between JAUS and the Joint Variable Message Format (JVMF).