Huxley: A Flexible Robot Control Architecture for Autonomous Underwater Vehicles

Dr. Dani Goldberg
Bluefin Robotics Corporation
553 South Street
Quincy, MA 02169 USA
dgoldberg@bluefinrobotics.com

***Abstract*—This paper presents "Huxley," a production robot control architecture that was developed by Bluefin Robotics for its fleet of autonomous underwater vehicles (AUVs). Huxley was designed with flexibility foremost in mind, allowing it to be easily adapted and reliably deployed on a wide range of platforms. The architecture follows a layered paradigm, providing a clean and logical abstraction for the major control functions. It also provides an interface for interaction with the layers, enabling expansion of the core capabilities of the architecture. This interface provides users the flexibility to develop smart payloads capable of utilizing available data, modifying the behavior of the AUV and even exploring new frontiers of autonomy.**

Keywords—control architectures, layered architectures, robotics, autonomous underwater vehicles, AUVs, unmanned underwater vehicles, UUVs

I. Introduction

Control architectures for autonomous mobile robots have been an active area of research and development for the better part of three decades. While many of the existing published architectures were innovative in their time and helped influence the current state-of-the-art, the vast majority of these architectures has been focused on academic research and has never been deployed on commercial production systems. Furthermore, while arguably applicable to all mobile robotic platforms, few of these architectures have been demonstrated on autonomous underwater robots. This paper presents a control architecture, dubbed "Huxley," that was specifically designed for use on commercially-available, production autonomous underwater vehicles (AUVs). In addition to describing the motivation and design of Huxley, this paper will compare and contrast Huxley with other well-known robot control architectures.

Bluefin Robotics began development of Huxley in 2003 as a next generation, flexible control system for its fleet of AUVs. Owing to its nature as a production software system, Huxley had a number of important constraints placed on it, the most fundamental of which was the need for flexibility. If Huxley had not been able to accommodate the current and future AUVs manufactured by Bluefin, or if the cost of doing so had been too high, then Huxley would have had a short life indeed. An interesting consequence of Huxley's need for flexibility was that it drove many other characteristics that are generally associated with good software systems. Flexibility meant that Huxley had to be robust and reliable on many different platforms; flexibility meant that it had to be easily extensible to accommodate new hardware and new system capabilities; flexibility meant that it had to be maintainable such that improvements or modifications in one area would not cascade across the system; and flexibility meant that it had to be testable to ensure the integrity of Huxley across all deployed systems. With these goals in mind, Bluefin developed Huxley over the course of several years. Huxley began shipping in early 2005 and has since proven itself to be the flexible control system that was intended.

The Huxley control architecture has two core layers: a reactive layer and an executive layer (Fig. 1). The reactive

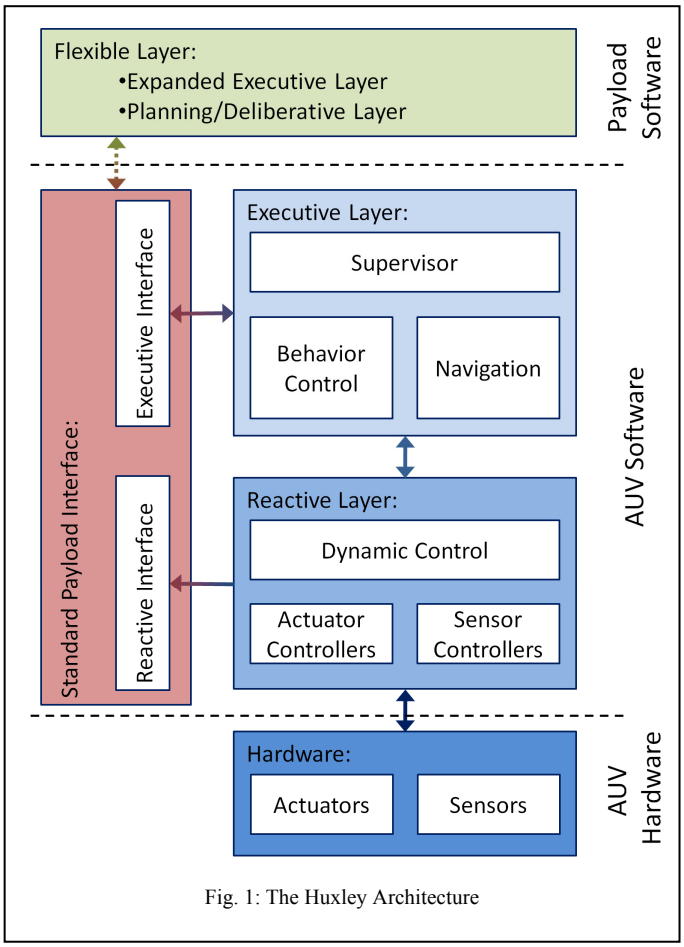

Fig. 1: The Huxley Architecture

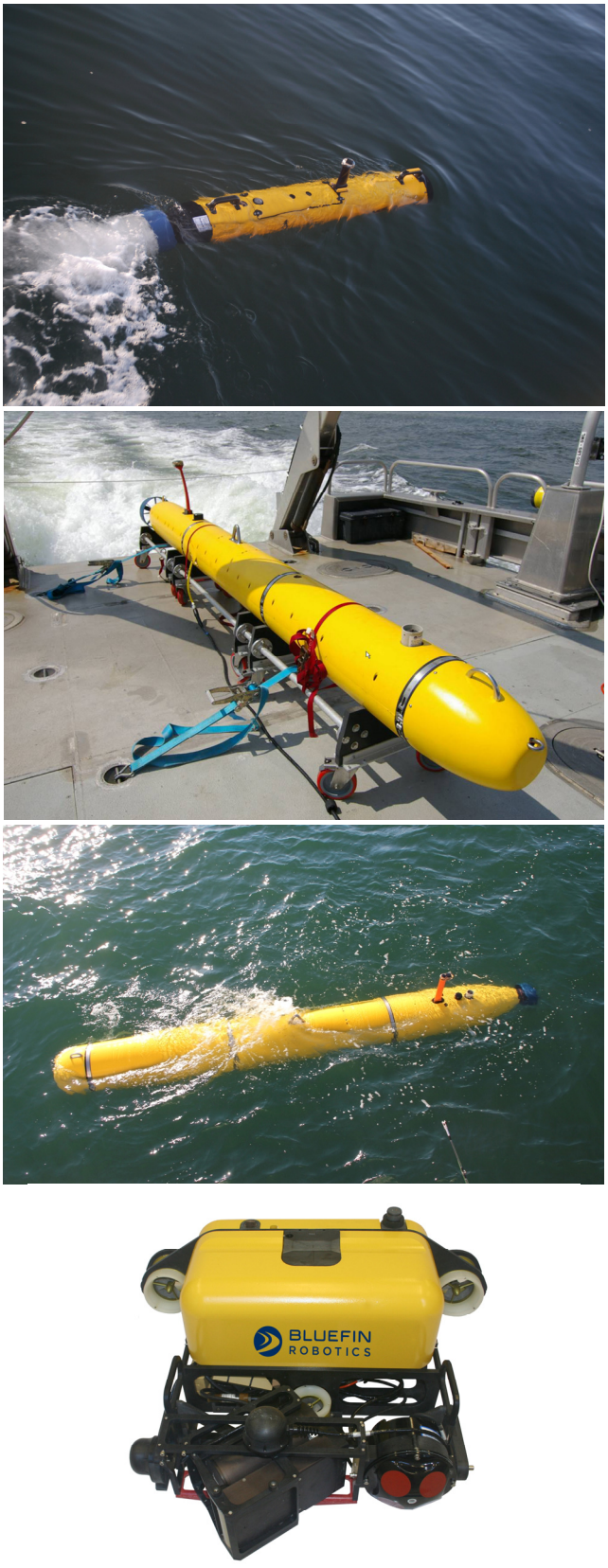

Fig. 2: Top to bottom: Bluefin-9, Bluefin-12D, Bluefin-21, and HAUV.

layer performs tight-loop control of the system with little notion of maintained state. The reactive layer contains individual controllers that interface with hardware sensors and actuators to issue commands and receive data/feedback at a high frequency. Built on top of these controllers, though still in the reactive layer, is the dynamic control system that monitors and adjusts the AUV's position and attitude so as to maneuver the AUV as desired by the executive layer. The executive layer is responsible for more protracted, state-based activities, including navigation, behavior control and a supervisor function. Navigation fuses all appropriate information over time to maintain the best estimate of the vehicle's position and attitude. Behavior Control decomposes high-level plans into actions to be accomplished by the reactive layer. A Supervisor monitors the overall state and health of the system.

One of the most important aspects of the Huxley architecture is that it provides the flexibility to expand the core reactive and executive layers via a standard interface (Fig. 1). This interface is built on the core Huxley infrastructure and works organically within the architecture. It provides the opportunity for users to create two things: one, an expanded executive layer that issues commands to the reactive layer; and/or two, a higher-level planning/deliberative layer that can issue commands to the executive or reactive layers.

These aspects of Huxley and its use on AUVs will be explored in more detail in the following sections. We begin with a discussion of the design goals for Huxley, followed by a detailed examination of the Huxley architecture and how it meets the designed goals. We then delve into a core enabling technology of the Huxley implementation, the communications protocol, and show how it fits into a working Huxley system. Throughout the paper, we will touch on Huxley's relationship to other related work in the literature.

II. Huxley Design Motivations

At the time of its design in 2003, the overarching requirement for Huxley was that it be applicable to all of Bluefin's major AUV platforms and allow for easy customizations of each delivered system to meet specific customer needs. The platforms and their applications, described below, are varied and help illustrate the motivation behind the Huxley architecture.

Bluefin's platforms include the Bluefin-9, Bluefin-12S, Bluefin-12D and Bluefin-21 classes of torpedo-style autonomous vehicles (Fig. 2), each categorized by the outside diameter of its hull (i.e., 9, 12 and 21 inches). These AUVs are generally employed as data collection platforms by customer with very different needs. Defense customers have utilized the platforms for applications such as mine-countermeasures (MCM) and battlespace preparation (BP) activities in shallow water, though interest is also expanding to areas such as persistent surveillance and anti-submarine warfare (ASW). Commercial customers are focused on applications such as shallow- and deep-water hydrographic surveys and collecting data for the oil and gas industries, among others. Scientific customers are interested, for example, in data for oceanographic research, developing new underwater sensors and data-collection sub-systems (e.g., new imaging sonars),

and conducting research on single- and multi-vehicle autonomy. Huxley was also intended to be applied to Bluefin's Hovering Autonomous Underwater Vehicle (HAUV) class [1] that uses thrusters along three axes to provide five degree-of-freedom control of the vehicle allowing it to hover and maintain position relative to other objects (Fig. 2). Applications for the system include ship hull inspection, port and harbor security, and infrastructure assessment. While outwardly the control of the HAUV may appear very different from that of Bluefin's other AUVs, the goal was for HAUV to share a common architecture and the vast majority of common code with the torpedo-style platforms.

The requirement for Huxley to be able to support multiple platforms [1] (current and future), the varied applications for those platforms, and individual customizations for each delivered system translated to the overarching design goal of *flexibility*.

Flexibility as a design goal, however, is of limited benefit in and of itself. In the extreme, total flexibility means total generality, the absence of any significant supporting architecture or infrastructure – clearly of limited benefit. Rather, flexibility is most useful when it guides the development of an architecture by informing and balancing other major design goals and decisions.

For Huxley, flexibility of the control architecture helped balance other requirements, including:

- *Reliability* – In order to truly be said to support multiple platforms/customizations, Huxley had to be able to do so reliably. That meant having common elements and infrastructure to provide a well-tested and proven core software system.
- *Extensibility* – Huxley had to be able to accommodate new capabilities, platforms, and next-generation enhancements.
- *Maintainability* – If Huxley were to become more difficult to maintain over time as the number of configurations and customizations increased, Huxley would not have been truly flexible or met the design need.
- *Testability* – Without the ability to test Huxley quickly and easily across the supported platforms and configurations, extending and maintaining the system while keeping it reliable would have been a practical impossibility.

Other important design considerations for software systems include usability, efficiency and portability. While these are important attributes that played a role during the detailed design and implementation of Huxley, they were not major players in the high-level architectural design, our focus in this paper.

Given the design goals stated above, it is not surprising that Huxley had to be architected with a proper level of modularity and significant reusability – a design philosophy even more important because it was shared with Bluefin AUV hardware architecture and overall system configuration. Luckily, there was no need to completely reinvent the wheel in this regard. At that time, in 2003, mobile robot control architectures had been an active area of research for several decades with major principles fairly well established, if not universally accepted (see [10] and [11] for relevant discussions). Two such principles for autonomous, physically embodied mobile robots (or "situated agents," if you prefer) may be expressed as follows:

- *Maintain appropriate interaction with the world.*
- *Perform activities at the appropriate timescales.*

The state of the real world (i.e., the robot's environment) is to a certain extent unknown, uncertain, unconstrained and unobservable. Regardless of the control algorithms and decision making algorithm that a robot employs, it must make observations of the world around it and take actions within that world, and it must do so at a frequency that is appropriate to the dynamics of its environment. An autonomous robotic car driving down a busy freeway, for example, must both observe lane markings, signs, and other cars, and act to get to its goal safely; failure to observe or act would be disastrous. No less for an AUV skimming the ocean floor thousands of meters below the surface, or operating within a busy harbor environment – it must interact with its environment to accomplish it goals safely.

The activities that a robot must undertake in interacting with the world and accomplishing its objectives take place on different timescales. On a very short timescale, the robot must send commands to its actuators and sensors and receive back status/data in order move and interact with the physical world. This level of control is sometimes referred to as *"reactive"* owing to its short timescale, tight coupling with the physical world, and little-to-no maintained state.

On a longer timescale, the concern is achieving the goals set forth for the vehicle (e.g., driving to a destination, or surveying a region of the ocean floor) while maintaining the system within certain constraints of safety, execution time, battery capacity, etc. These goals may be expressed programmatically, though they are most useful and flexible when expressed in a high-level plan representation. In the latter case, the controller must decompose the plan representation into lower-level tasks and ultimately actions to be taken by the reactive layer. Due to its role in the execution of plans and accomplishment of system goals, this layer is often referred to as the "*executive*." It is often viewed as the mediator between plan generation (whether done by human or otherwise) and the lower-lever reactive layer.

On a timescale greater than that of the executive layer is the element of the robot typically concerned with establishing the goals for the system. These goals typically involve some sort of planning such as path planning, resource management, constraint management, task prioritization, high-level fault detection and recovery, scheduling of activities, or high-level coordination with other elements of the system. Not

[1] Bluefin also manufactures vehicles which, due to particular hardware, software, or program requirements, do not use Huxley.

surprisingly, this layer of control is often referred to as the "*planning*" or "*deliberative*" layer.

The notions of layering the robot control architecture according to timescales of activity and maintaining an appropriate level of interaction with the environment helped guide the structure of the Huxley architecture. They also helped balance the flexibility of the system with the need for reliability, extensibility, maintainability and testability. In the following section we explore how the design goals and notion of layered control were applied to the Huxley architecture.

III. HUXLEY'S LAYERED ARCHITECTURE

The power of Huxley, or any good software architecture, is how it helps constrain the solution space from everything possible to a subset appropriate to the problem domain. For Huxley, this meant focusing on the problem of controlling real world AUVs for an ever-expanding customer base with new requirements and customizations while also allowing for future enhancements. While daunting at first, as we shall see in the following paragraphs, the major elements of Huxley fell out of the basic requirements for the system.

A. The Reactive Layer

At the most basic level, Huxley has to be able to control AUV hardware devices – sensors and actuators – that directly interact with the physical world. The issuing of commands and receipt of data needs to be at a high frequency with low latency to make the vehicle accurate and responsive in its actions while also providing high resolution data to customers. These needs argued for a *reactive layer* of control (Fig. 1). While useful conceptually, a monolithic reactive control layer would have been of limited benefit when it came to flexibility, extensibility and reusability. Within the reactive layer, therefore, Huxley requires a control module (or "driver") for each hardware component. As an example, see Fig. 3 in which the reactive layer of a hypothetical vehicle contains three individual drivers, one each for a GPS unit, a sonar, and a thruster. Note that, in an actual AUV, Fig. 3 would be expanded with drivers controlling navigation sensors (e.g., inertial measurement unit or inertial navigation system, Doppler velocity log, pressure sensor), actuators (e.g., propulsors, thrusters), communications subsystems (e.g., acoustic modems, satellite communications) and data collection sensors (e.g., sidescan sonar, synthetic aperture sonar, multibeam sonar, sub-bottom profiler, forward look sonar, water quality sensors).

In addition to drivers, the reactive layer includes a *Dynamic Control* component [2] that contains the low-level control loops for stable vehicle flight. On Bluefin's torpedo-style vehicles, for example, Dynamic Control is responsible for stable depth and altitude maintenance, as well as a variety of horizontal control modes such as heading, trackline, trackcircle and waypoint following. The Dynamic Control module, however, can implement whatever low-level control scheme is appropriate to the class of AUV. Dynamic Control of the HAUV platform, for example, is unique (owing to its multiple degrees of freedom and its use of an object-relative coordinate frame) and, yet, its Dynamic Control module fits naturally within Huxley.

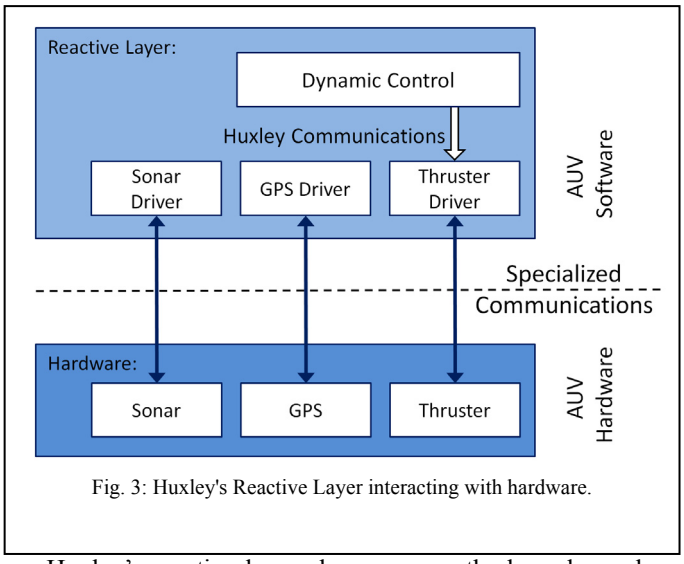

Fig. 3: Huxley's Reactive Layer interacting with hardware.

Huxley's reactive layer also serves as the boundary where specialized communications protocols and mechanisms are translated to a common, shared protocol. This is illustrated in Fig. 3 where the communication pathway between drivers and their corresponding devices is labeled as "Specialized Communications" (solid arrows) while communication between Dynamic Control and the Thruster Driver is labeled as "Huxley Communications" (represented as a white arrow). Some hardware devices may share a communications protocol (e.g., most GPS units use NMEA [3] messages), but many use protocols unique to the device or vendor. Part of the responsibility of a Huxley driver is translating these hardware-specific interfaces to a common communications protocol (described later in this paper). The notion of keeping non-standard interfaces at the edges of the Huxley architecture (here, the boundary between devices and drivers) is a key to Huxley's flexibility. Substituting, adding or removing a hardware device is relatively straightforward as major changes are localized to the driver. Localizing changes also promotes extensibility, maintainability and reliability of the software system since changes do not propagate or destabilize the entire system.

B. The Executive Layer

Another core capability of Huxley is mission execution. A Huxley-based AUV had to be able take a high-level mission plan, decompose it into executable elements that it then provides to the reactive layer along with data for proper control of the vehicle's hardware. This capability is the purview of Huxley's *executive layer* of control (Fig. 1). Similar to the reactive layer, though, it is not sufficient simply to state that there is such a layer; the components and organization of the executive layer must promote the desired qualities of the system.

Huxley's executive layer is composed of three major elements: *Behavior Control*, *Navigation*, and a *Supervisor*. Behavior control is responsible for decomposing high-level plan representations into control commands that are passed to the reactive layer for execution on the hardware. The plan representation may be generated several different ways:

1. By a human using an application. For example, Bluefin has a chart-based, graphical Mission Planner application that serves this purpose. As an alternative, Mission Planner also supports open external interfaces for planning via other tools.[2]
2. Via an automated mechanism. Examples of this are the deliberative/planner layer of the architecture creating the plan, or an automated external application doing so via an available interface.

Regardless of where and how the plan is generated, it is the responsibility of Behavior Control to decompose the plan into control commands that it sends to Dynamic Control (Fig. 4). Note that these commands utilize the common Huxley communications protocol (white arrows). Behavior control is also able to send Huxley commands to device drivers in the reactive layer to control various parameters (such as device operating settings and data logging) that are represented in the mission plan. In order to send the right commands at the right time, Behavior Control must monitor the AUV's progress based on time, position, or other factors.

Navigation is responsible for fusing and filtering all available navigation sensor inputs into the best estimate of the vehicle's position, attitude and rates, providing these data to other components of the system. One component is Behavior Control, which uses the data to help monitor mission plan execution. The other is Dynamic Control, which utilizes the data as feedback to adjust control parameters to ensure that the AUV is following the control rules and maintaining proper stability. Note that, to preserve the clarity of Fig. 4, the inputs to Navigation are not drawn. In a fully elaborated diagram, one would see Huxley communications from the GPS Driver to Navigation. Navigation would also be seen consuming data from other sensors (e.g., Doppler velocity log, pressure/depth sensor, inertial unit, etc.) in order to produce its estimate of state.

The final element of the executive layer is the Supervisor, a component responsible for vehicle lifecycle management and health and status monitoring. In other words, it *supervises* the overall state of the vehicle. As shown in Fig. 4, the Supervisor communicates with all of the other modules in the system, both in the executive and reactive layers, using Huxley's communication protocol (white arrows). The Supervisor monitors the components to ensure that they are functioning properly and reports overall health and status of the system. While the Supervisor is not directly responsible for running missions (that is Behavior Control's job), it is responsible for coordinating mission start and stop by making certain that system and all sub-components are in the proper state.

The elements of Huxley's executive layer (Behavior Control, Navigation, and Supervisor) constrain the system in a useful manner while still allowing flexibility. Consider Navigation, as an example. The Huxley architecture says that there must be a component responsible for maintaining an estimate of the vehicle's position/attitude and providing those data to other components. The architecture, however, does not require any specific implementation of Navigation. One could develop a navigation module based on the use of an inertial navigation system (INS), an inertial measurement unit (IMU) and compass, use of long baseline (LBL) beacons, or other schemes. At Bluefin, for example, we have developed multiple navigation modules for our torpedo-style vehicles, including both INS-based [4] and IMU/compass-based versions providing global vehicle position. Depending on the customer's requirements, we can simply "drop-in" the appropriate module for the vehicle. As a further example, there is Bluefin's HAUV, which navigates relative to other objects, not in a global frame like the torpedo vehicles. The HAUV Navigation module fits just as cleanly in the Huxley architecture as any of the other navigation modules.

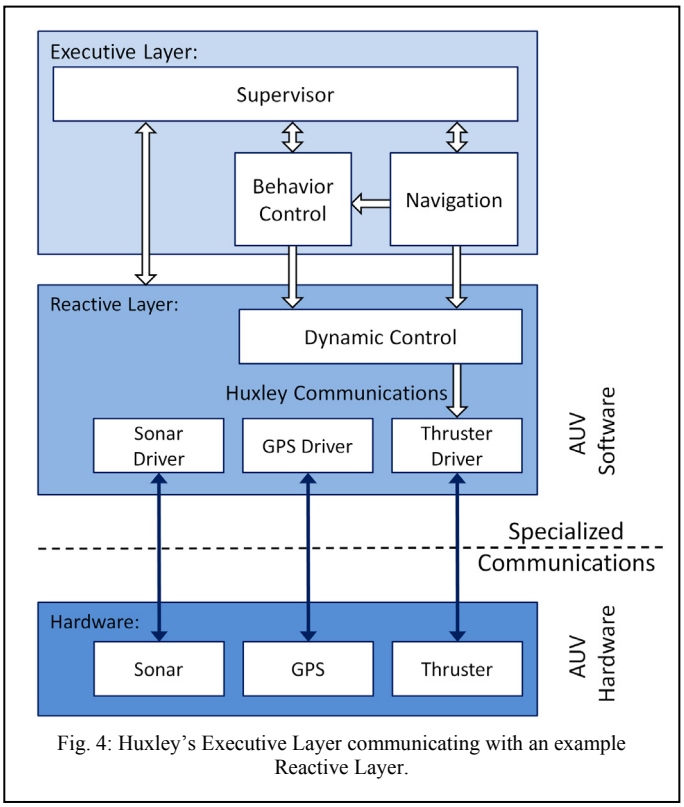

Fig. 4: Huxley's Executive Layer communicating with an example Reactive Layer.

Huxley's executive layer of the Huxley architecture provides three core elements for an AUV: Navigation; mission plan decomposition and execution (Behavior Control); and system health and status management (Supervisor). The architecture allows extending capabilities by dropping-in appropriate implementations for each of these elements. It also facilitates reliability and maintainability through the reuse of common components within a stable framework.

C. *The Standard Payload Interface: Another Level of Flexibility*

The reactive and executive layers form the stable core of the Huxley architecture. There is plenty of flexibility in choosing hardware and matching it to Huxley drivers and other components to meet customers' requirements. A Huxley vehicle with a complete reactive and executive layer is a fully

[2] Two such interfaces are: The Common Operator Interface Navy (COIN), and the Mine Warfare and Environmental Decision Aids Library (MEDAL).

operational AUV able to take mission plans and execute them successfully, thereby providing the customer with the desired data. Bluefin has implemented a large set of Huxley components and a rich plan representation language capable of expressing a myriad of different missions that are executed by Behavior Control.

As flexible as the core Huxley capabilities are, they are not sufficient for some of our customers – in particular those who are developing their own payload hardware or are interested in developing new behavioral modes or higher-level deliberative control functions. For these customers, Huxley departs from the typical notion of a layered control architecture to provide an external interface to the executive and reactive layers. This *Standard Payload Interface* (Fig. 5) utilizes the Huxley communications protocol (white arrows) with components in the reactive and executive layers. It also provides an external interface for communications with the payload, ideally utilizing a messaging format that is easy for customers to use. Bluefin's implementation of the Standard Payload Interface, for example, utilizes NMEA style messages. The Standard Payload Interface is an optional component that can be dropped-in for systems that require such capability, without necessitating changes to other components. As such, it is in-line with the Huxley goals of extensibility, maintainability and reliability.

The Standard Payload Interface provides communications with both the reactive and executive layers. At the reactive layer, the Standard Payload Interface receives data directly from Huxley drivers and provides those data across the payload communications channel; it does not send commands directly to Dynamic Control or the drivers. This asymmetry between data and commands is intentional and is motivated by the need to maintain safe, reliable core AUV control. If a payload were able to send commands directly to Huxley drivers, it could compromise the control and monitoring

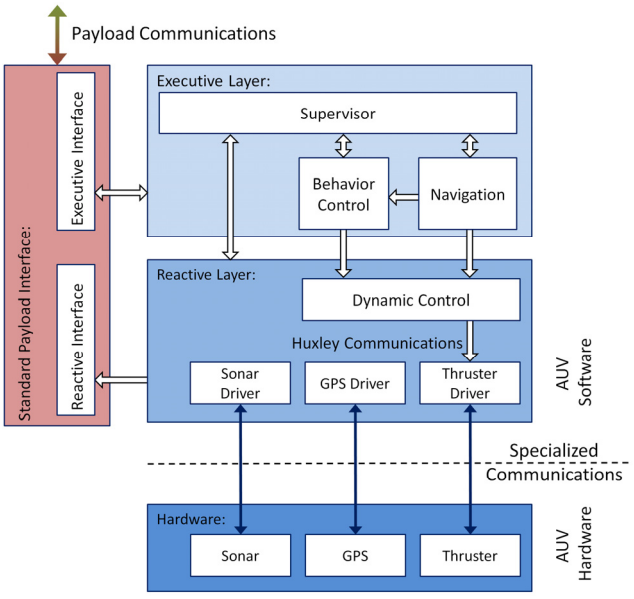

Fig. 5: Huxley's Standard Payload Interface for interaction with the executive and reactive layers of the architecture.

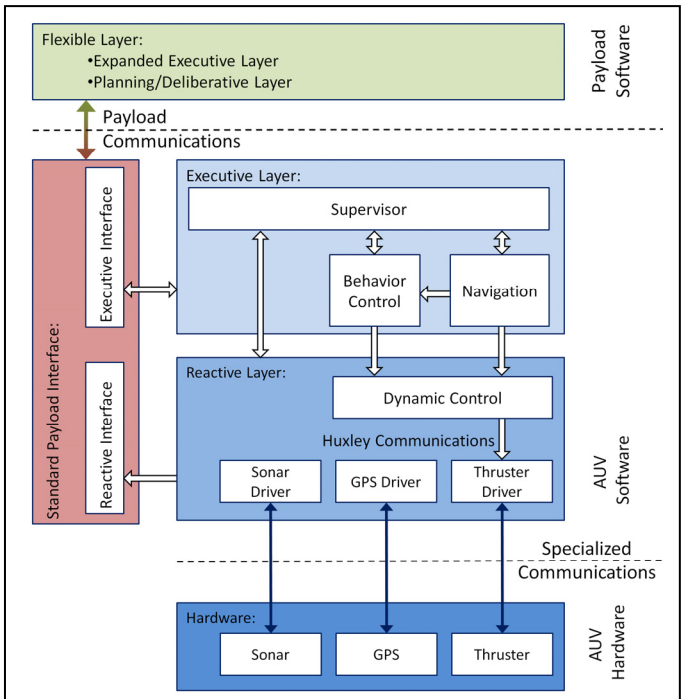

Fig. 6: Huxley's Flexible Payload Layer communicating with other layers via the Standard Payload Interface.

functions of the executive layer, leading to unpredictable behavior, reduced reliability and safety hazards.

At the executive layer, the Standard Payload Interface translates control requests from the payload into Huxley messages to the executive layer components. These are "requests" rather than "commands" because the executive layer reserves the right not to honor them. If the payload, for example, were to ask the AUV to collide with the ocean floor, the executive layer would typically be within its rights not to do so. In general, however, the executive layer will attempt to fill requests faithfully, whenever possible. This give-and-take relationship between the payload and the Huxley executive layer is extremely powerful. It gives the customer the freedom/flexibility to experiment and develop "smart" payloads capable of receiving data and issuing complex commands, while keeping the core software systems and vehicle reliable and safe. Standard Payload Interface requests from the payload can include: stopping the mission, sending communications, changing the vehicle's current actions, inserting new mission elements, etc. To aid the payload, data are also available from the executive layer including overall vehicle health and status from Supervisor, mission execution status from Behavior Control, and vehicle position/attitude from Navigation. These data can be logged by the payload as part of its data collection requirements, or used as an aid in formulating requests via the Standard Payload Interface.

In the next section, we will examine some of the possible system enhancements enabled by the Standard Payload Interface.

D. The Flexible Payload Layer

The Standard Payload Interface allows the addition of an optional third layer to the architecture that can serve various functions depending on the requirements for the system and the needs of the customer. This optional layer is completely at the control and discretion of the customer, constrained only by the data and commands available across the Standard Payload Interface.

For customers developing their own sensor packages, the Flexible Layer may take the form of a "smart" payload. The customer can develop their own software to interface with their developmental sensor package and communicate with the core Huxley system via the Standard Payload Interface. In such a system, the payload package may need data from core systems (e.g., navigation data for localization, sound speed corrections, etc.) and commands to synchronize the payload's behavior with the core Huxley system. Bluefin's current implementation of the Standard Payload Interface provides a variety of data streams, and allows arbitrary user-defined payload commands to be embedded into mission plan representations and sent to the payload at desired points in the mission.

For some applications, it may also be necessary for the smart payload to send command requests to the Executive Layer across the Standard Payload Interface. These commands could constitute small modifications to the executing plan localized in time. For example, upon seeing an object of interest on the ocean floor, the payload could request an in-stride excursion to approach or get a different angle on the target. A payload performing localized modifications of the executing mission could be considered to constitute an "expanded" Executive Layer. Similar to the core Huxley Executive Layer, the payload relies on the high-level plan representation to define the overall mission, but enhances the capabilities of the Executive with appropriate modifications and in-stride excursions.

Continuing along the spectrum of smart payloads, one eventually finds payloads capable of significantly modifying the mission plan, or alternatively, defining their own mission plans. These payloads effectively constitute a traditional Planning or Deliberative Layer for the architecture. They rely on the Huxley Executive and Reactive Layers for data and safe execution of requested actions, but are capable of generating (and possibly reasoning about) their own high-level plans. These are the payloads of customers interested, for example, in exploring the boundaries of planning, autonomy, and multi-vehicle coordination.

As we have seen in this section, the Huxley architecture promotes a reliable and maintainable core system that is also extensible, able to easily accommodate different hardware devices and software capabilities (e.g., navigation solutions) with clean localized changes. On top of this, Huxley provides the flexibility of a Standard Payload Interface that can be used to safely enhance the capabilities of the system with smart payloads that expand the Executive Layer or even constitute a full Planning Layer.

A good software architecture, like Huxley, is important in achieving the desired qualities of a system – but so is the implementation. In the next section, we will examine a few of Huxley's implementation details that help Huxley realize its potential as a robot control architecture.

IV. Stream-Oriented Messaging Architecture: A Huxley Enabling Technology

Bluefin's implementation of the Huxley architecture rests on a foundation of key infrastructure elements. One of the most vital of these is the Huxley messaging protocol, called the *Stream-Oriented Messaging Architecture (SOMA)* (and represented as white arrows in Fig. 3 through Fig. 6). SOMA is the communications "glue" that creates a unified system out of the elements of the Reactive Layer, Executive Layer, and the Standard Payload Interface. It is also vital to how Huxley achieves a high level of flexibility, extensibility, reliability and maintainability.

SOMA is a publish-subscribe messaging protocol with a central dispatcher (called the *SomaManager*) that establishes peer-to-peer connections between applications (or processes). In the Huxley implementation, each component of the core system (i.e., device drivers, Dynamic Control, Behavior Control, Navigation, Supervisor, and Standard Payload Interface) is an independent application that uses SOMA for communications. Each Huxley application running on a computer connects with a SomaManager running on the same machine to indicate what messages it is publishing and to what messages it wishes to subscribe. The SomaManager is responsible for connecting applications publishing specific messages to other applications that wish to subscribe to those messages.

Consider the example of the three Huxley applications shown in Fig. 7. The Navigation application (App1) informs SomaManager that it would like to receive (or subscribe to) GPS data. The CTD Driver (App2) informs SomaManager that it provides (or publishes) conductivity-temperature-depth (CTD) data, while the GPS Driver (App3) publishes GPS data. Seeing the match between subscriber and publisher, the SomaManager facilitates the creation of a peer-to-peer connection so that App1 (Navigation) gets its GPS data directly from App3 (GPS Driver).

Due to the publish/subscribe nature of SOMA, there is no enforced synchronization mechanism between subscribers and publishers. Publishers provide data at frequencies that are appropriate to the type of data they produce and subscribers consume the data at that frequency for their processing. A GPS device, for example, outputting data at 1 Hz would usually translate to the Huxley GPS Driver publishing data at 1 Hz. An IMU or INS, on the other hand, which outputs data at 10's of Hz may have a Huxley driver that publishes at that frequency or downsamples, if appropriate.

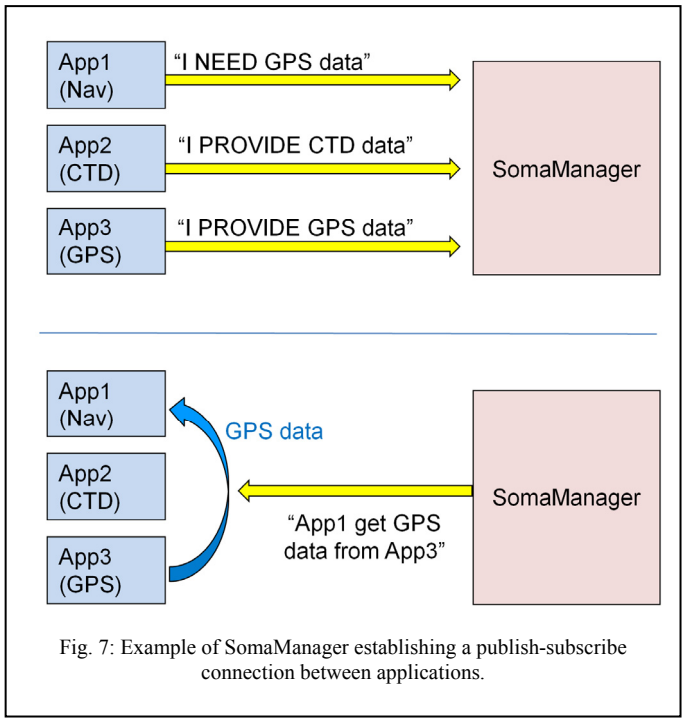

Fig. 7: Example of SomaManager establishing a publish-subscribe connection between applications.

A. *Types of SOMA Messages*

SOMA messages come in several varieties distinguished by their reliability and persistence. These distinctions are important to the proper function of a system and are described in more detail below.

1) *Reliability of Messages*

SOMA messages may either be reliable or unreliable, referring to the guarantee of receipt. Given an executing application publishing a reliable message, SOMA attempts to guarantee that any executing subscriber will eventually receive the message, if there exists sufficient bandwidth to send data. Unreliable SOMA messages do not have any guarantee of receipt. A subscriber will only receive an unreliable message if there is sufficient bandwidth to support its transmission. If not, the message will be dropped.

Within each Huxley process, reliable SOMA messages have priority over unreliable messages. Pending reliable message publications will always be serviced before unreliable messages. Given limited bandwidth, this helps ensure that reliable messages (assumed to be of greater importance) reach their destination.

2) *Persistence of Messages*

Independent of reliability, SOMA messages may be either persistent or non-persistent. Persistent messages have values that endure beyond the immediate publication of the message. Consider the case of an application that has published a persistent message only once. Subsequent to this publication, a different application subscribes to this publication. Bandwidth permitting, the subscribing application will receive the value that was last published as it has persisted beyond the immediate publication event. In the case of non-persistent messages, applications that subscribe to such messages after the last publication will not receive the data that were most recently sent.

SOMA message persistence provides a mechanism by which pertinent data can be transferred between applications without the need to synchronize start times or require that applications re-send the same (old) data every time a new application subscribes to a particular message. Such a mechanism would not only result in unnecessary message traffic but would also not be desirable for all types of data.

3) *Combinations of Reliability and Persistence*

The four combinations of reliability and persistence exist as distinct types of SOMA messages.

- ***Measurement*** – Refers to unreliable, non-persistent SOMA messages. Measurements are intended for data that require frequent transmission and for which only the current value is of importance. Examples of such data include sensor readings such as the latitude and longitude provided by a GPS unit. It is arguably poor practice to make GPS data reliable as they are frequently updating and if one is dropped, the next may be received. In addition, old GPS positions may have little relevance to actual position, so persistence is not important.
- ***Command*** – Refers to reliable, non-persistent SOMA messages. Commands are intended for data that are sent infrequently and require a guarantee of receipt, but have no value unless they are received in a timely fashion. An example would be a message telling Behavior Control to start a specific mission. If the system is functioning properly, then Behavior Control should reliably start the mission. If not, one would not want Behavior Control "spontaneously" starting a mission at a later time when the system may be functioning properly.
- ***Status*** – Refers to reliable, persistent SOMA messages. Status messages are most applicable to data that are seldom published but whose values remain relevant long after publication. Such messages are often status-related. Whenever a Huxley application, for example, changes its overall state (initializing, running, failed) it publishes it as a Status message. This is a key value that other applications need to see reliably, and whose value is relevant for long periods of time.
- ***StreamCommand*** – Refers to unreliable, persistent SOMA messages. StreamCommands are used for rapidly published commands where the correct response to a dropped message is not to retry, but to wait for the next command. For example, the rapid changes to a thruster's RPM would fall into this category.

4) *SOMA Interfaces (IFs)*

As part of Huxley's SOMA implementation, there exists the notion of a SOMA Interface (IF). IFs bundle together messages of a particular type to create a grouping that is common to components within the architecture. While applications within the Huxley framework can certainly use individual SOMA

messages for communication, it is really the IFs that help facilitate the level of flexibility, modularity and reuse that is promised by the Huxley architecture.

Consider the example of a Huxley driver for a conductivity-temperature (CT) sensor. The driver must communicate with the hardware to get conductivity and temperature values that it publishes as Measurements. Grouping these two measurements into an IF (call it *CondTempIF*) that represents data published by a CT sensor driver, allows drivers for different CT sensors to use the same IF. Bluefin has integrated multiple CT sensors on its AUVs. Even though each device has a different driver, the drivers all publish data using the same CondTempIF. This means that subscribers to CondTempIF can be agnostic as to the hardware and Huxley driver providing the data. It also means that the integration of a new CT sensor does not propagate changes beyond the driver itself.

Another example, described earlier in the paper, is that of Navigation. Bluefin has developed several different navigation solutions for its torpedo-style vehicles, each of which is designed for different sensor suites and/or customer needs. Each version of Navigation, however, utilizes the same Measurement IF for publishing the navigation solution (i.e., the vehicle's current latitude, longitude, depth, roll, pitch, yaw, velocities, accelerations, etc.). This means that no matter which version of navigation is employed in a system, other components (such as Behavior Control and Dynamic Control) can remain agnostic and unaffected.

As we have seen, SOMA and the use of IFs are key technologies in the implementation of Bluefin's Huxley architecture. As a publish-subscribe protocol, SOMA not only provides the messaging between core layers and components of the Huxley architecture, it does so in a way that promotes the desired flexible qualities of the system. Huxley can be extended with new functionality and configurations to meet customers' needs while localizing changes and allowing significant reuse of components. Localizing changes and reusing components facilitates maintenance of the software and promotes reliability of the delivered systems.

V. Related Work

Mobile robotic systems that interact with the world (AUVs being one type) have been an area of research interest for decades. Reference [4] provides a review of research touching on Artificial Intelligence and Robotics and explores foundational notions of situatedness, embodiment, intelligence and emergence. Within the wider field of Robotics, many investigators have explored the use of layered architectures including [5], [6], and [7]. Huxley, being a layered architecture, shares similarities with these. Reference [15] describes CLARAty, a two-layered architecture developed by NASA and its partners with the goal of establishing a system that is reusable across multiple platforms. This goal is shared with Huxley. The architectures, however, differ in that Huxley has two core layers and an interface for expanding the system with a third layer. Another well-known and popular approach to robot control is Behavior-Based Robotics [13], [14], which is founded, in part, on earlier work on a layered architecture called the Subsumption Architecture [12]. While Huxley overall is not a behavior-based architecture, the Behavior Control element in Huxley's executive layer is very much in the style of a behavior-based controller.

Reference [8] expands on earlier work by creating a three-layered architecture that allows coordination with other robots at each layer of the architecture. The Huxley architecture, in contrast, requires the use of the Standard Payload Interface for interacting with the layers of the architecture. This is meant to keep the core Huxley systems stable and reliable while providing flexibility to expand the system. Another difference is that Huxley does not define an explicit planning layer but allows one to be defined as needed through the use of the Standard Payload Interface. Reference [9] further expands on [8] with a market-based planning layer. The market-based coordination mechanism described is one possible planning layer that could be added to Huxley via the Standard Payload Interface. Many others are possible.

The Standard Payload Interface has been used on multiple Bluefin systems. References [16] and [17] are examples where the interface has been used directly for integration of smart payloads. Reference [18] describe work in which the MOOS-IvP software system [19] is used to provide a planning layer of control via the Standard Payload Interface.

VI. Conclusion

This paper has described the Huxley robot control architecture, developed by Bluefin Robotics as a production software system for its fleet of AUVs. Huxley was designed with flexibility foremost in mind, but with the understanding that other qualities (e.g., reliability, extensibility, maintainability, modularity, and reusability) were important in creating true flexibility. The architecture has two core layers (reactive and executive) providing the stable foundation for expansion via a Standard Payload Interface. While Huxley was designed for flexibility, the use of the SOMA messaging protocol in the implementation has helped realize the architecture's potential. Given Huxley's qualities, it promises to be the core software system on current and future platforms for years to come.

Acknowledgment

The author would like to thank Deanna Talbot, Louis Quartararo, Robert Panish and Jeff Smith for valuable feedback on earlier drafts of this paper. The Huxley architecture and implementation are the joint work of many folks at Bluefin Robotics.

References

[1] J. Vaganay, L. Gurfinkel, M. Elkins, D. Jankins, and K. Shurn, "Hovering autonomous underwater vehicle – System design improvement and performance evaluation results," in *Proc. 16th Int. Symp. Unmanned Untethered Submersible Technology*, Durham, NH, 2009.

[2] R. Panish, "Dynamic Control Capabilities and Developments of the Bluefin Robotics AUV Fleet," in *Proc. 16th Int. Symp. Unmanned Untethered Submersible Technology*, Durham, NH, 2009.

[3] *NMEA 0183 Interface Standard,* NMEA Standard 0183, 2002.

[4] R. A. Brooks, "Intelligence Without Reason," in *Proc. 12th Int. Joint Conf. Artificial Intell.*, 1992, pp. 569-590.

[5] R. Bonasso, D. Kortenkamp, D. Miller, and M. Slack, “Experiences with an architecture for intelligent, reactive agents,” *J. Artificial Intell. Research*, vol. 9, no. 1, 1997.

[6] N. Muscetolla, P. P. Nayak, B. Pell, and B. Williams, “Remote agent: To boldly go where no AI system has gone before,” *Artificial Intell.*, vol. 103, no. 1-2, pp. 5-48, 1998.

[7] R. Simmons, R. Goodwin, K. Haigh, S. Koenig, and J. O’Sullivan, “A layered architecture for office delivery robots,” in *Proc. 1st Int. Conf. Autonomous Agents*, 1997.

[8] R. Simmons, T. Smith, M. B. Dias, D. Goldberg, D. Hershberger, A. Stentz, and R. M. Zlot, “A layered architecture for coordination of mobile robots," in *Proc. 2002 NRL Workshop on Multi-Robot Syst.*, May, 2002.

[9] M. B. Dias, D. Goldberg, and A. Stentz, "Market-based multirobot coordination for complex space applications," in *Proc. 7th Int. Symp. on Artificial Intell., Robotics and Automation in Space*, May, 2003.

[10] R. Brooks, “Challenges for complete creature architectures,” In *Proc. 1st Int. Conf. on Simulation of Adaptive Behavior*, Paris, France, 1991.

[11] R. Simmons, "Structured Control for Autonomous Robots," *IEEE Trans. Robot. Autom.*, .vol. 10, no. 1, Feb. 1994.

[12] R. Brooks, “A robust layered control system for a mobile robot,” *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 14–23, Mar. 1986.

[13] M. J Matarić, “Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior,” *Trends in Cognitive Science,* vol. 2, no. 3, pp. 82-87, Mar. 1998.

[14] M. J Matarić and F. Michaud, “Behavior-Based Systems,” in *Springer Hanbook of Robotics*, B. Siciliano and O. Khatib, Eds. Berlin, Germany: Springer-Verlag, 2008, ch. 38, pp. 891-909.

[15] I. A. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I. Shu, and D. Apfelbaum, “CLARAty: Challenges and steps toward reusable robotic software,” *Int. J. Advanced Robotic Systems*, vol. 3, no. 1, pp. 23-30, 2006.

[16] J. Kloske, “AUV-12 Sonar Integration and Evaluation,” SRI International, Menlo Park, CA, Rep. ESD-18858-AR-10-075, Feb. 2010. [Also: DTIC ADA515048; http://handle.dtic.mil/100.2/ADA515048]

[17] A. A. Proctor, J. Kennedy, E. Gamroth, C. Bradley, and D. Gamroth, “The ocean technology test bed - From concept to operation,” in *Proc. IEEE OCEANS 2010*, Sept. 2010, pp.1-7.

[18] M. R. Benjamin, H. Schmidt, P. Newman and J. J. Leonard, “Nested autonomy for unmanned marine vehicles with MOOS-IvP,” *J. Field Robotics,* vol. 27, no. 6, pp. 834–875, 2010.

[19] M. R. Benjamin, P. M. Newman, H. Schmidt, and J. J. Leonard, “An Overview of MOOS-IvP and Users Guide to the IvP Helm Autonomy Software”, CSAIL Technical Report, MIT, Rep. MIT-CSAIL-TR-2010-041, Aug. 2010.

About Bluefin Robotics

Bluefin Robotics manufactures and develops Autonomous Underwater Vehicle (AUV) systems and technology. Founded in 1997, the company has grown to become a world leader in AUV products designed for defense, commercial, and scientific applications. Bluefin Robotics is a wholly-owned subsidiary of Battelle Memorial Institute. For more information, please visit www.bluefinrobotics.com.