The shift in the robotics paradigm — the Hardware Robot Operating System (H-ROS); an infrastructure to create interoperable robot components

Víctor Mayoral
Erle Robotics
contact@erlerobotics.com

Alejandro Hernández
Erle Robotics

Risto Kojcev
Erle Robotics

Iñigo Muguruza
Erle Robotics

Lander Usategi
Erle Robotics

Asier Bilbao
Erle Robotics

Irati Zamalloa
Erle Robotics

***Abstract*—This article introduces the Hardware Robot Operating System (H-ROS), a joint hardware and software infrastructure to create vendor-agnostic reusable and reconfigurable robot hardware components. These robot components dynamically modify and extend the internal representation model of the robot enabling robots to switch on or off components selectively and literally exchange hardware components between them reducing the integration effort of building robots. Our results demonstrate the feasibility of the hypotheses presented, discuss future work and improvements on top of the proposed H-ROS infrastructure.**

***Keywords*—robotics, standard, hardware, ROS, real-time Ethernet.**

I. Introduction

Robotics is a technically challenging field where engineers often face situations where the integration of the robot effort, generally composed by diverse sub-components supersedes many other tasks. A common infrastructure reduces the integration effort by facilitating an environment where components can simply be connected and interoperate. Because each component is optimized for such integration at its conception, the infrastructure handles the integration effort. The result is both a high quality and superior performance over that of the integrated compound mix of single vendor solutions. The strength of an infrastructure is that it can integrate the best-of-breed components for the highest quality and performance. Components can come from different manufacturers, yet when supported by a common infrastructure, they will interoperate together.

The existing trend in robotics is producing a significant number of hardware devices. When compared to each other, these components typically consist of incompatible electronic components with different software interfaces. The rapid growth of the consumer robotics market provides solutions that are end-user oriented. When building robotics solution, they obscure the difficulty of system integration and interoperability. Conversely, industrial and professional robots are demanding solutions that enhance customization and reduce the costs of integration.

The Robot Operating System (ROS) [1] is a well-known, open source standard for robot application development. Recent advances in adopting ROS as a standardized software framework coupled with the availability of affordable hardware components suggests using ROS for the definition of a generic concept for *robot components*. Robot components that are vendor-agnostic, should reduce the integration time, favor reusability among robots and ultimately benefit the development of the field.

In this paper, we introduce the concept of the **Hardware Robot Operating System (H-ROS)**. A vendor-agnostic infrastructure for the creation of robot components that interoperate and can be exchanged between robots. H-ROS builds on top of ROS which is used to define a set of standardized logical interfaces that each physical robot component must meet if compliant with H-ROS. The following sections introduce our work behind the H-ROS concept. Section II discusses related work, Section III covers H-ROS in greater detail, and Section IV provides some conclusions, remarks and future work.

II. Related Work

Building a robot is accepted as a challenging task, thereby it makes sense to reuse previous work to reduce complexity. To the best of our knowledge, nowadays, there are few efforts that reuse hardware in both academia and industry. Robots are built by multidisciplinary teams, comprised of a whole research group or a company division, where engineers with varying backgrounds get involved in the mechanical, electrical and logical design. Teams spend the majority of their time dealing with the integration of the hardware/software interfaces and while little time is invested in behavior development or real-world scenarios. This trend of ”reinventing the wheel” results in a significant inconsistency between hardware and software components that are available today. The following sections present some relevant work in the field of hardware and software architectures for robots.

A. Robot software development frameworks

As pointed out by Scholl et al. [2], the construction of a robot constitutes the engineering domains of mechanics,

978-1-5386-3439-4/17/$31.00 ©2017 IEEE

electronics and software. Over the last several years, we have observed how software development has generated a number of systems and frameworks that can be used to program the behavior of a robot and abstract it in different modular and reusable components.

Reuse of research efforts in robotics started showing relevant traction around 2000 when the Player Project initiated (formerly the Player/Stage Project) [3], a project to create free software for research in robotics. Player/ Stage became relevant to the point where robotics journals and conferences regularly published papers featuring real and simulated robot experiments using these open source tools. Similar initiatives appeared at the time such as the Open RObot COntrol Software (OROCOS) project [4], Orca [5], the Evolution Robotics Software Platform (ERSP) [6] and the Carnegie Mellon navigation (CARMEN) toolkit [7].

In 2005, a second paper [8] about the Player/Stage/Gazebo (P/S/G) project was published including Gazebo as a separate project (previously a plugin within Stage) for 3D robot simulation. The P/S/G project spawned the growth of several robot platforms. In addition to the reuse of software, hardware (e.g. sensors and actuators) started being interoperable among these platforms.

In 2009, Quigley et al. [1] made the first publication about *"ROS: an open source Robot Operating System"*. ROS was part of the software development within Willow Garage in their effort to build the PR2 robot. Engineers put special focus into defining levels of abstraction, usually through message interfaces, that would allow much of the software to be reused elsewhere. The ROS framework, initially designed for attending the needs of the PR2 robot, ended up being adopted by a relevant part of the academic and industrial community for use cases that were beyond the aims of the PR2 (a *Robot for Research and Innovation* as defined on its product page). As the number of ROS-based products entering the market increased, including manufacturing robots, agricultural robots, service robots, and others, the platform proved to be limited and several flaws were repeatedly patched. Because of the limitations with ROS, ROS 2 is currently under development.

ROS 2 is designed with new use cases in mind and built on top of the standardized communication middleware Data Distribution Service (DDS) [9]. Examples of these cases include, enabling teams of multiple robots, small embedded platforms, real-time systems, non-ideal networks, production environments and prescribed patterns for building and structuring systems [10].

Compared to the field of software development, which generated numerous and popular frameworks, the areas of electronics and mechanical components for robotic applications lack such approaches. Note that given the variability of the components used in robots to meet different applications and use cases, enforcing a set of mechanical constraints makes little sense. However, reaching a common agreement for the electrical interfaces will bring relevant benefits and ultimately favor the creation of a marketplace of compatible and reusable robot components. This fact was already highlighted by Scholl et al. [2], but to date no relevant progress has been made.

B. Modular Robot Architecture (MRA)

Released in September 1991 [11] as a technical report from the Naval Ocean System Center in San Diego (CA), the Modular Robot Architecture (MRA) describes both the hardware and the software components that are used to create reconfigurable, modular robots (named MODBOTs).

The proposed architecture emphasized standard electrical and mechanical hardware interfaces between distributed processing modules, along with standard software libraries that provided communication services and process control across a wide range of processors. The product of the research behind MRA is a standardized set of tools for building robotic systems that can be easily reconfigured depending on project requirements and change in technology.

Even though the concept of MRA is an interesting one, the proposed architecture has never been implemented in any of the current robotic solutions.

C. The Joint Architecture for Unmanned Systems (JAUS)

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). From a software perspective, JAUS architecture has certain similarities with ROS. Each *Node* can be contained in one or different computers. A group of nodes put together a robot (called "Subsystem" in the JAUS jargon). Robots are grouped together creating "Systems".

The main advantages of JAUS are: its modularity, which is achieved by strict partitioning of subsystems into software components. The components of JAUS are reusable, and can be used in another robot platform due to its Object-Oriented principle of Inheritance. JAUS also offers interoperability[1] since, its subsystems communicate with same type of message set. Components from other vendors could be used to operate systems with no modification.

A relevant aspect to note about JAUS, it does not define the transport mechanism whereby messages are communicated between subsystems, nodes, and components. According to Rowe et al. [12] the current JAUS standard does not guarantee interoperability. As described by Rowe, the main limitations of JAUS are:

- **Rigid responsibility**: JAUS defines the messages that the component must process. Being JAUS-compliant means that the developer's design choices will be limited, which may hamper new creative solutions.
- **Required a priori knowledge**: 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.

[1] Subject to components using the same transport mechanism.

- **Lack of real-time facilities**: At the time of its conception, apparently JAUS did not include a notion of real-time responsiveness. Although messages are sequenced and tagged as requiring acknowledgement, there is no temporal component.
- **Acceptance**: For any standard to succeed it needs to be accepted in the development community. Since JAUS is a closed standard it is very difficult to be accepted by the wider community.

D. EtherCAT, a real-time Ethernet fieldbus standard

EtherCAT is an Industrial Ethernet solution [13], [14] that delivers high-performance at a low-cost. It's easy to use and provides a flexible topology. The technology was developed initially by Beckhoff and later handed off to the EtherCAT Technology Group (ETG). EtherCAT has been identified as the most widely adopted real-time Ethernet fieldbus standard [15], [16]. It offers the best price/performance and according to its specifications. EtherCAT delivers results that include the processing of 12,000 digital I/O in 350 microseconds, or 100 servo axes in 100 microseconds.

Medium access control employs the *master/slave* principle. From an Ethernet point of view, an EtherCAT bus is simply a single Ethernet device (the EtherCAT *master*) that sends and receives Ethernet telegrams. Its key functional principle lies in how its nodes process Ethernet frames: each *slave* node reads the data addressed to it and writes its data back to the frame all while the frame is moving downstream until it reaches the *master*. This leads to improved bandwidth utilization (one frame per cycle is often sufficient for communication) while also eliminating the need for switches or hubs.

The EtherCAT telegrams are either transported directly in the data area of the Ethernet frame and can either be coded via a special Ethernet type or within the data section of an UDP datagram transported via IP (which is known as Ethernet over EtherCAT or EoE). The first variant is limited to one Ethernet subnet but provides lower latencies. The second variant (EoE) via UDP/IP generates a slightly larger timing overhead (IP and UDP header), but allows using IP routing.

	EtherCAT Standard	Ethernet/IP	Powerlink Standard	PROFINET IRT Standard	SERCOS Standard
STANDARDS BODY	EtherCAT Technology Group (ETG)	ODVA	Ethernet POWERLINK Standardization Group (EPSG)	PROFINET International (PI)	SERCOS International
TOTAL MEMBER COMPANIES (including non-Ethernet members)	3200	300	250	1400	90
MANUFACTURING MEMBER COMPANIES	500	30	25	80	90
REAL TIME PART MFG	180	2	10	6	3
SERVO DRIVE COMPANIES	150	5	9	12	7
I/O COMPANIES	100	8	5	6	4

Figure 1. Real-time, Ethernet-based fieldbuses comparison. As shown in this statistics EtherCAT is the most common automation communication standard. *Original source [16]*

While EtherCAT does not strictly deliver a standardized software and hardware infrastructure for robotics, it does provide a robust Industrial Ethernet solution that answers the needs of the physical and low-level (logical) infrastructure of a robotic framework. Compared to the other standards for automation communication protocols, EtherCAT offers not only best performance but also focuses on making sure that the information arrives to physical devices appropriately and in-time. During the research performed, our team considered this technology as a relevant candidate for the construction of an infrastructure for robotics.

III. The Hardware Robot Operating System (H-ROS)

Figure 2. Official logo of the Hardware Robot Operating System (H-ROS). The image was inspired by the initial logo of the Robot Operating System (ROS) with the consent of the ROS-community steered by the Open Source Robotics Foundation (OSRF).

None of the current state-of-the-art robot development frameworks provide common hardware and software groundwork to create vendor-agnostic reusable robot components. The Hardware Robot Operating System (H-ROS), is a novel infrastructure and lays the groundwork for providing a solution for reusable robot hardware components that dynamically modify and extend the internal representation model of the robot. The presented infrastructure enables robots to exchange hardware components between them.

H-ROS is built on top of ROS, which is the *de-facto* standard for robot software development [1]. In more detail, H-ROS utilizes the functionality of ROS 2; a redesigned version of the robot middleware that as described above, targets use cases not included in the initial design of ROS.

The H-ROS compatible robot modules are classified depending on their functionality as follows:

- **Sensing**: these components perceive or *sense* the environment and report about its state. Components include but not limited to: cameras, lidars and range finders.
- **Actuation**: components that produce some form of physical change in the environment. Among the most typical examples are actuators and within actuators, the servo motors.
- **Communication**: components specialized in communication. These parts could either expose new communication channels to the overall H-ROS network, for example, WiFi, 3G, and Bluetooth, or provide means of interconnection between different H-ROS hardware parts.
- **Cognition**: components specialized in computation and coordination. These components perform most of the computationally expensive tasks within the robot.

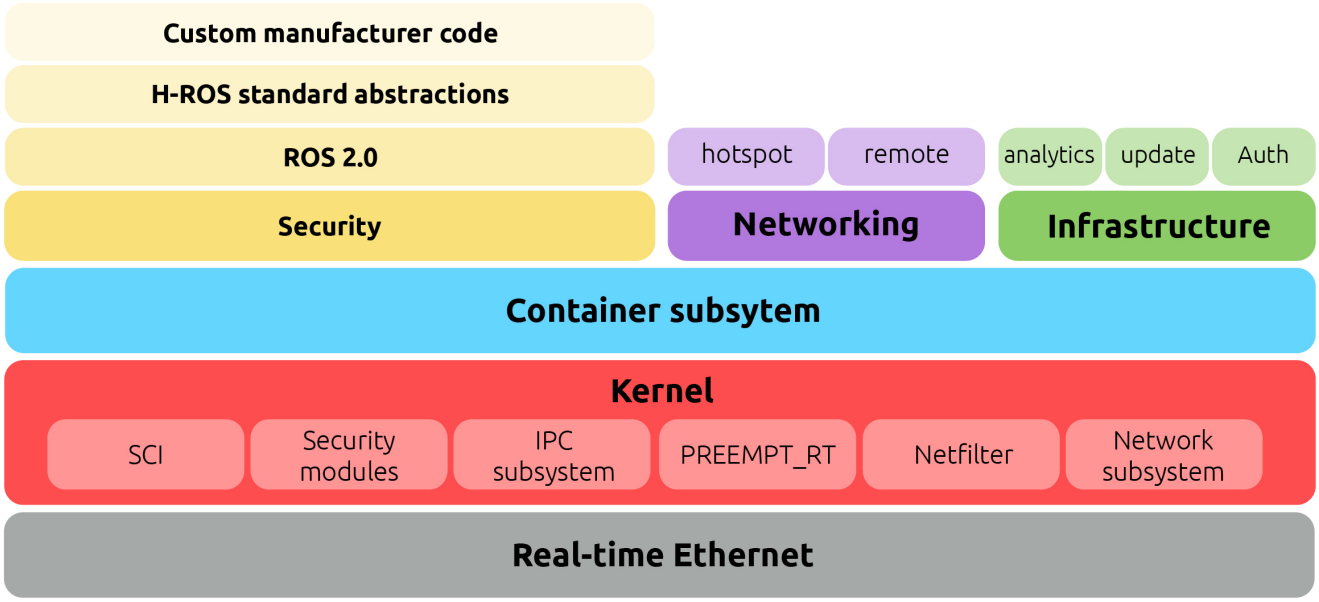

Figure 3. **The Hardware Robot Operating System (H-ROS) infrastructure as defined by the groundwork**. H-ROS is composed of different layers which create a robot infrastructure that can be deployed to any robot hardware component. The physical layer is based on real-time Ethernet and guarantees fast and reliable communication between different components. The Linux kernel sits on top, providing a high degree of customization when deploying the infrastructure in a wide range of embedded devices. The kernel used within H-ROS is a fork of the mainline kernel with custom modifications to serve the purposes of components for robots. The container subsystem allows developers to package and isolate applications which makes it easy to move the contained application between environments (development, test, production) or robot components while retaining full functionality. Upper layers of abstraction include different security mechanisms, the ROS 2 framework, H-ROS standard definitions, modules that allow the manufacturer to simplify the acquisition of metrics from individual components or send updates to the overall software infrastructure remotely. Details about a number of these modules that belong to the H-ROS infrastructure are covered below.

They also deal with the reconfiguration process of H-ROS parts (plug and play functionality) and expose ROS interfaces for new computations.

The philosophy behind H-ROS is that creating robots is about placing together components that are compliant with the standardized H-ROS interfaces, regardless of the manufacturer. The following subsections will introduce some key details about the H-ROS infrastructure.

A. *Groundwork*

In this section we present the groundwork that has been undertaken for each layer, as summarized in Figure 3.

1) Communication standard: Existing research about the use of robots in industry highlights the fact that techniques which permit real-time control must be found and extended for new and demanding application areas. According to Blomdell et al. [17] the most promising hardware interfacing possibilities (from a cost and performance point of view) are shared memory access via the peripheral connection interface (PCI) system bus and standard high-speed Ethernet communication.

While traditional networked interfaces may introduce delays and have limited performance when compared to the shared memory interfaces (PCI), previous work [18]–[20] showed promising steps towards achieving real-time responses using Ethernet-based networks. More recent technical reports, such as the one presented by Enner [21], show experimentally the viability of Ethernet and UDP for robot real-time control under an appropriate networking setup.

As Enner nicely summarizes, roboticists typically show two main concerns when considering technologies for real-time control which are the *latency predictability* and the *out-of-order delivery*, both addressed by newer switching technologies within local networks and without redundant routes or load balancing. Many industrial automation communication protocols have been designed on top of real-time Ethernet, see Figure 1 for details. H-ROS relies on Real-time Ethernet to provide means of communication among its components. Depending on the requirements, different implementations could be selected.

The groundwork performed in the development of H-ROS focused on evaluating *traditional Ethernet* (using UDP) as the communication underlying platform for the whole infrastructure. The benefits of using this technology are: standardized communication protocol, uniform and well understood addressing per each device, widely used technology (cost-effective and easy to prototype) and wide available chip-sets from different manufacturers. Future work will involve extending H-ROS communication layer to include other interfaces such as CAN FD [22], serial peripheral interface (SPI) [23] and recent Internet of things (IoT) technology [24].

2) Power: One of the novelties H-ROS aims to introduce is the use of power and data in the same cable. This concept has already been in standards such us Power over Ethernet (PoE) [25] and EtherCAT P [26]. Simplifying the cabling of a robot is beneficial for integration of robot components, since

the wiring and maintenance gets easier. Under the assumption of traditional Ethernet for this groundwork, PoE was used in the first iteration of H-ROS and selected as the default method for powering H-ROS components. This standard provides up to 25.5W per device. Knowing that some devices could require more power, particularly certain actuators, an independent and additional power supply was also considered as part of the reference electrical designs produced. Each robot part also measures its power consumption and reports it. Furthermore, each robot part can be selectively powered on and off, enabling custom activation of robot parts remotely.

3) Processing: Due to the rapid development and growth of Single Board Computers (SBCs) and embedded-electronics, cost-effective embedded processors are flooding the market, which enables the creation of size-constrained devices with the capability of running complex algorithms. Our observations suggest that the current landscape of application specific processors, specifically high-end microcontrollers, is intersecting the market of SBCs. With this vision in mind, our team established that every H-ROS component should have its own dedicated microprocessor to handle each module computing tasks. Each H-ROS part is required to include a microprocessor that can run ROS 2 and possesses the appropriate computational resources to interface with other subsystems. Under this assumption, large computational tasks can be performed in each robot part, abstracting the rest of the components from this complexity and simplifying the interfaces necessary to reuse these components among different robots.

4) Real-time Operating System: In order for a computing device of any kind to meet real-time requirements there are two aspects that should be addressed: 1) an operating system or abstraction that provides real-time responses and 2) user code with deterministic execution time.

While the communication standard selected is real-time compliant at the physical level, it still relies on the overlying Real-time Operating System (RTOS) to ensure that the right threads receive appropriate cycles to meet the real-time constrains [27]. Among the different solutions available for RTOSs, the PREEMPT_RT patch set, an initiative funded and supported by the Real Time Linux (RTL) collaborative project and supported by the Linux Foundation[2], has proved to obtain a deterministic response that meets the constraints H-ROS has to face even in industrial applications [28]. Experimental tests performed by our team validated these claims, which lead to the adoptation of RTL within the kernel layer of the infrastructure.

5) Software Architecture: On top of the kernel layer, the overlying abstractions are containerized. Containers [29]–[31], provide the ability to package and run processes in loosely isolated environments. This architecture provides isolation and security while allowing the user to run many containers simultaneously in the same host. Containerization also brings relevant benefits for developers by simplifying the development, test and deployment process of containers in different robot components. Within the H-ROS infrastructure, the container subsystem exposes a *base container* that wraps up a minimal file-system that is inherited by upper layers to create more specific containers. Within this layered architecture, the infrastructure delivers containers that provide functionality that includes but is not limited to: remote access to robot components, enable specific hardware capabilities or receive metrics or analytics from components. This brings together an architecture with the capability of easily installing and upgrading easily containers within robot components and make the maintenance of software easier than in traditional file-systems.

[2]Refer to https://wiki.linuxfoundation.org/realtime/start for more information.

Range finder standardized interface

```
- hros_sensing_distance_<mac address>
  - **Parameters**
    - update_rate [integer, default: 10]
  - **Topics**
    - **published**
      - distance:
          [hros_sensing_msgs/Range.msg]
      - features:
          [hros_generic/feature.msg]
      - info:
          [hros_generic/info.msg]
```

Figure 4. The H-ROS standard interface for a range finder sensor device is shown. Standard interfaces uses common ROS abstractions such as ROS messages, parameters or services.

The ROS 2 container inherits from the *base container*. This layer exposes a base ROS 2 distribution that is used by upper layers to build software using ROS. The ROS container exposes a number of options that can be easily configured to meet different real-time requirements.

Built on top of the ROS 2 container, a whole Hardware Abstraction Layer (HAL) has been set up. The *H-ROS standard abstractions* layer provides the necessary interfaces to make H-ROS vendor-agnostic. Figure 4 provides an abstract example of one of such standardized interfaces. Component manufacturers will need to comply with all these required interfaces. In order to facilitate this task, specialized containers are provided so that component vendors can build their *custom manufacturer code* by simply inheriting from these containers.

B. Validation and Evaluation

Provided all the hypotheses described above and after completion of the infrastructure development groundwork, we proceeded to create the first H-ROS robot parts, as seen in Figure 5. Using these parts, an H-ROS compliant robot was created as pictured in Figure 6. This first robot has relevant design similarities with the popular Turtlebot robot in the ROS-world. All components of the Turtlebot comply with the H-ROS standard interfaces and are connected together through communications components that provide a means of interconnection. The information from sensors and actuators reaches the cognition component, which is responsible for processing the data and executing user-defined behaviors. In this particular case, depending on the attached hardware, these were the implemented behaviours:

- Collider: robot navigates around the environment and collides with different obstacles.

Figure 5. Examples of a number of H-ROS parts built during the process of validation and evaluation. Each one of the parts pictured here was built using the principles described above.

- Cyclop: obstacle avoidance, when the obstacle is detected under certain distance, the robot avoids collisions, by changing the defined trajectory.
- Smartnav: robot navigates around and creates a 2D and 3D map of the environment, which is visually available in one of its screens.

Figure 6. H-ROS Turtlebot, first robot H-ROS compliant. The H-ROS Turtlebot is composed by a set of hardware parts that comply with standard interfaces as defined by H-ROS. These parts include sensors, actuators, communication and cognition components.

Based on these realizations, our team concluded that the use of H-ROS technology brings the next benefits, compared to other robotics solutions:

- **Interoperability:** The standardized interfaces enabled components to interoperate even when produced with completely different underlying hardware that came from different vendors. This hypothesis was validated experimentally and enabled the H-ROS Turtlebot to maintain the same behaviors when using different sets of H-ROS compliant components.
- **Reusability and reconfigurability:** Reusability implies that these distributed components should be able to reconfigure themselves dynamically when hot-plugged or removed for its use in different robots. This process is coordinated, monitored and controlled by an H-ROS cognition part. By doing so, the cognition part will be able to identify which new robot component has been added to the system, its kind, capabilities and where it's physically connected in the robot. For this to happen, each one of the H-ROS components should provide enough information about its state so that the cognition part is able to rebuild dynamically an internal model of the robot. In order to explore the feasibility of the reuse and reconfigurability of the different H-ROS components, our team added a variety of sensors on each one of the H-ROS parts that reported information about its state. Active research is undergoing to explore the feasibility of these hypotheses with different robots. First results provided promising insight about the reuse of such components between robot manipulators with different configurations.

The presented groundwork shows the feasibility of creating a common infrastructure for robotic components that offer new capabilities to the robots containing them. Once connected, different components would be able to instantly reconfigure themselves and adapt the internal model of the robot allowing any integrator to connect components in a plug-and-play fashion. The plug-and-play functionality together with the inherent extensibility capabilities of the systems described allows adding new functionality to the robot or improving existing ones, testing the results easily and allowing agile iteration when building a robot which ultimately enhances customization and reduces the integration effort.

While the results obtained showed a relevant improvement when compared to other mechanisms, our team observed that a few of our initial hypotheses were not completely met. With the infrastructure described above. The following items summarizes some of the most relevant negative aspects encountered while validating and evaluating our approach:

- **Power management with *daisy-chained* components**: Having an architecture where robot components can dynamically be plugged imposes several challenges from a powering perspective. Among the different approaches prototyped, *daisy-chaining* components required each one of the H-ROS parts to monitor and report its consumption at every moment resulting in a more complex set of power electronics embedded on each one of the H-ROS compliant parts. With this information, the power management system within the robot could determine dynamically what is the available power for a new daisy-chained component and selectively decide to switch it on or off based on available power.
- **Real-time evaluation of the communication standard**: Our experimental tests delivered acceptable results with the architecture described above in terms of real-time. Isolated tests with a range of H-ROS compliant components confirmed that a careful and

controlled network using *traditional Ethernet* (over UDP) could indeed meet the real-time requirements of many robot systems. However we observed that as the robot network got populated with more and more components —requiring switching mechanisms— an appropriate setup of the network for optimized real-time responses increased the engineering (and integration) effort substantially which defeated one of the main purposes of the H-ROS infrastructure. Furthermore, we noticed that maintaining real-time responses over UDP was specially challenging when 1) adding or removing dynamically new components and 2) mixing real-time and non-real-time traffic together. This led us to conclude that our initial assumption about the underlying communication standard for H-ROS would not cover all the use cases for robots. Particularly, mission critical or industrial robots might not accept the jitter observed.

The following section covers briefly some of the current work our team is undertaking to address the limitations presented.

C. Moving forward, a solution for professional and industrial robot components

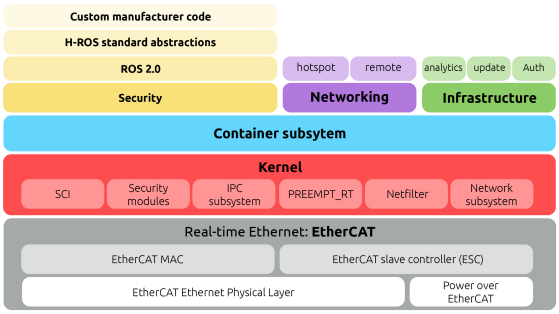

Figure 7. Hardware Robot Operating System (H-ROS) infrastructure for professional and industrial applications.

Beyond the arguments exposed above, results state that the bandwidth in a traditional Ethernet-based network is limited [32]. According to Juergen et al. data transfer times in traditional Ethernet is around 100 milliseconds, which is not sufficient for the requirements of industry automation protocols. Compared to standard Ethernet, automation fieldbuses can achieve data transfer of 10 milliseconds. Further enhancements of the MAC layer leads to better transfer times, however for utilizing this techniques dedicated hardware or software is necessary [32].

Based on all the aforementioned reasoning, our team concluded that the *standard Ethernet* protocol is not sufficient for the construction of a robotics infrastructure in professional and industrial environments. To address this limitation, the physical layer of the H-ROS infrastructure was replaced by EtherCAT as displayed in Figure 7. The process of creating an H-ROS compliant robot component is summarized in Figure 8.

Preliminary results from robot components built using an infrastructure as described in Figure 7 fulfilled all the initial goals. Several manufacturers have shown interest in adopting the H-ROS infrastructure within several of their components for robots and there is an ongoing discussion with relevant forums to introduce H-ROS within several ecosystems of robots.

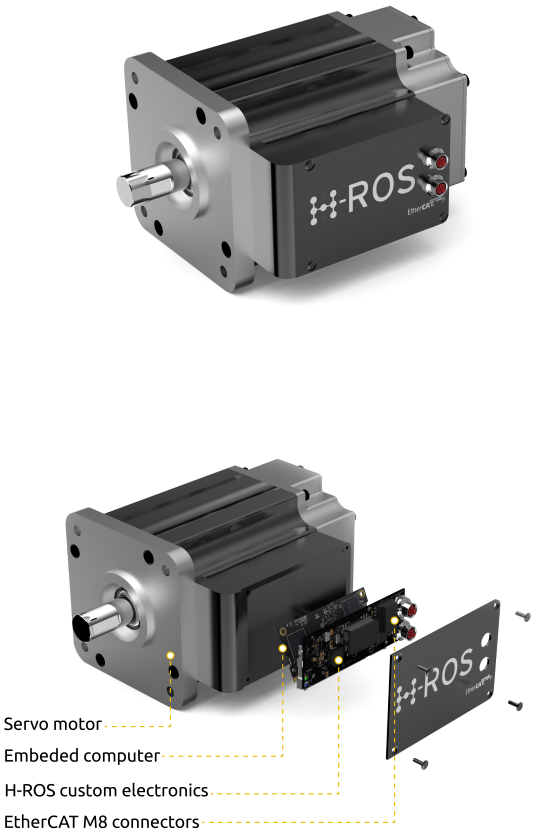

Figure 8. Generic H-ROS servo motor part. Attached to the servo motor, an embedded microprocessor is added. This allows to interact and control the motor using vendor driver plus communicate with the rest of H-ROS parts. The custom power electronics enables powering up and providing communications using just a single cable.

IV. Discussion

This work presented a novel concept of modular, vendor-agnostic infrastructure named as the Hardware Robot Operating System (H-ROS), which will shift the robotics paradigm towards standardization and democratizing the way robots, and more general automation systems are built today.

H-ROS provides a fast way of building robots choosing the best component for each use-case from a common robot marketplace. It complies with industrial environments where variables such as time constraints are critical. Building or extending robots is simplified to the point of placing H-ROS compliant components together. The user simply needs to program the cognition part (i.e. brain) of the robot and develop their own use-cases without facing the complexity of integrating different technologies and hardware interfaces.

As described in Section III-A, we have shown the feasibility and functionality of developing and implementing the H-ROS

infrastructure. The benefit of this infrastructure is the roboticist can easily integrate and exchange different robot components in a common ecosystem of components without the typical difficulties that engineers face today when building or upgrading a robot. The limitations identified on this first approach were explained in Section III-B and addressed by replacing the corresponding physical layers and software abstractions which was introduced in Section III-C. With these changes, the proposed H-ROS infrastructure is built upon standard automation fieldbus technology and complies with standard software frameworks, such as ROS 2 delivering the best of both worlds.

In a time when the importance of robotics and their potential is being intensely highlighted, hardware components have recently become affordable products and accessible tools for a wide majority of users. With a lack of standards and thousands of integrators reinventing the wheel on every product, H-ROS aims to become the infrastructure establishing the required basics for an exponential growth of this technology.

ACKNOWLEDGMENT

This work received funding from the Defense Advanced Research Projects Agency (DARPA) through their Robotics Fast Track (RFT) program and the Business development basque agency (SPRI) through the HAZITEK program.

The authors would like to also thank the Acutronic group and particularly Nicholas Wichowski who helped reviewing and refining the H-ROS work.

REFERENCES

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[2] K. Scholl, V. Kepplin, J. Albiez, and R. Dillmann, "Developing robot prototypes with an expandable modular controller architecture," in *Proceedings of the International Conference on Intelligent Autonomous Systems, Venedig*, 2000, pp. 67–74.

[3] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, vol. 1, 2003, pp. 317–323.

[4] H. Bruyninckx, "Open robot control software: the orocos project," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 3. IEEE, 2001, pp. 2523–2528.

[5] A. Makarenko, A. Brooks, and T. Kaupp, "Orca: Components for robotics," in *International Conference on Intelligent Robots and Systems (IROS)*, 2006, pp. 163–168.

[6] M. E. Munich, J. Ostrowski, and P. Pirjanian, "Ersp: A software platform and architecture for the service robotics industry," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE, 2005, pp. 460–467.

[7] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2003, pp. 2436–2441.

[8] T. H. Collett, B. A. MacDonald, and B. P. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Proceedings of the Australasian conference on robotics and automation (ACRA 2005)*, 2005, p. 145.

[9] G. Pardo-Castellote, "Omg data distribution service: architectural overview," in *IEEE Military Communications Conference, 2003. MILCOM 2003.*, vol. 1, Oct 2003, pp. 242–247 Vol.1.

[10] B. Gerkey, Open Source Robotics Foundation, Inc., Tech. Rep. [Online]. Available: http://design.ros2.org/articles/why_ros2.html

[11] R. Laird, R. Smurlo, and S. Timmer, "Development of a modular robotic architecture," DTIC Document, Tech. Rep., 1991.

[12] S. Rowe and C. R. Wagner, "An introduction to the joint architecture for unmanned systems (jaus)," *Ann Arbor*, vol. 1001, p. 48108, 2008.

[13] D. Jansen and H. Buttner, "Real-time ethernet: the ethercat solution," *Computing and Control Engineering*, vol. 15, no. 1, pp. 16–21, 2004.

[14] G. Beckhoff, "Ethercat: The ethernet fieldbus [z]," *EtherCAT Technology Group*, 2006.

[15] M. Felser, "Real time ethernet: Standardization and implementations," in *Industrial Electronics (ISIE), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 3766–3771.

[16] Kingstar, "White paper: 5 real-time, ethernet-based fieldbuses compared," Kingstar, Tech. Rep., January 2016.

[17] A. Blomdell, G. Bolmsjo, T. Brogardh, P. Cederberg, M. Isaksson, R. Johansson, M. Haage, K. Nilsson, M. Olsson, T. Olsson *et al.*, "Extending an industrial robot controller: implementation and applications of a fast open sensor interface," *IEEE Robotics & Automation Magazine*, vol. 12, no. 3, pp. 85–94, 2005.

[18] A. Martinsson, "Scheduling of real-time traffic in a switched ethernet network," *MSc Theses*, 2002.

[19] J. Kerkes, Tech. Rep., February 2001. [Online]. Available: http://www.embedded.com/design/connectivity/4023291/Real-Time-Ethernet

[20] G. Prytz and S. Johannessen, "Real-time performance measurements using udp on windows and linux," in *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, vol. 2. IEEE, 2005, pp. 8–pp.

[21] F. Enner, "Analyzing the viability of ethernet and udp for robot control," Tech. Rep., November 2016. [Online]. Available: https://ennerf.github.io/2016/11/23/Analyzing-the-viability-of-Ethernet-and-UDP-for-robot-control.html

[22] B. Cheon and J. W. Jeon, "The can fd network performance analysis using the canoe," in *Robotics (ISR), 2013 44th International Symposium on*. IEEE, 2013, pp. 1–5.

[23] F. Leens, "An introduction to i 2 c and spi protocols," *IEEE Instrumentation & Measurement Magazine*, vol. 12, no. 1, pp. 8–13, 2009.

[24] S. Tayeb, S. Latifi, and Y. Kim, "A survey on iot communication and computation frameworks: An industrial perspective," in *Computing and Communication Workshop and Conference (CCWC), 2017 IEEE 7th Annual*. IEEE, 2017, pp. 1–6.

[25] P. S. Equipment, "Power over ethernet," 2004.

[26] B. Automation, Tech. Rep., February 2001. [Online]. Available: https://www.beckhoff.com/english.asp?ethercat/ethercat-p.htm

[27] H. Fayyad-Kazan, L. Perneel, and M. Timmerman, "Linux preempt-rt v2. 6.33 versus v3. 6.6: better or worse for real-time applications?" *ACM SIGBED Review*, vol. 11, no. 1, pp. 26–31, 2014.

[28] M. Mossige, P. Sampath, and R. G. Rao, "Evaluation of linux rt-preempt for embedded industrial devices for automation and power technologies-a case study," in *9th RTL Workshop [Online]. Available: http://www.linuxdevices. com/files/article081/Sampath. pdf*, 2007.

[29] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[30] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*. IEEE, 2013, pp. 233–240.

[31] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support paas," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE, 2014, pp. 610–614.

[32] J. Jasperneite, M. Schumacher, and K. Weber, "Limits of increasing the performance of industrial ethernet protocols," in *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*. IEEE, 2007, pp. 17–24.