HRS-OpenBrain: A JAUS Compliant Robot Programming Framework

Anuj Kapuria, Gaurav Taank, Sahil Malhotra, Shashank Bhatia, and Chandan Datta

Hitech Robotic Systemz Ltd, Gurgaon, India
{firstname.lastname}@hitechroboticsystemz.com

***Abstract*—This paper describes HRS-OpenBrain, an open source software environment developed on robotic system framework guidelines as given in the Joint Architecture for Unmanned Systems (JAUS). A few common problems in the development of robotic systems are discussed with our observations. The ultimate aim of HRS-OpenBrain is to expedite and facilitate the development of mission oriented robotic systems. Another objective is to reduce the life cycle time and cost for developments like addition of new algorithms to robotics systems. We present the architecture and implementation of our software system along with testing results.**

I. Introduction

The common robotics systems' development life cycle comprises of the concept, design, execution and integration phases. A substantial amount of rework is being done at various robotics research groups during the execution phase, in re-implementing proven technologies for different platforms and applications. This rework undesirably prolongs the robotic systems' development life cycle, which in turn hinders the pace of growth for robotics as a technology. This fact is clearly observed while comparing the pace of development with the computer systems domain, where computer hardware has got 1000 times faster per dollar over the past decade [20]. Computer software design has become more modular and standardized, enabling the developers to focus directly on the application, rather than framework or infrastructure. On the other hand, a significant amount of time is spent in every robotics project towards redesign or maintenance of the infrastructure.

Common examples in Artificial Intelligence (AI) are algorithms like planning, mapping, localization, etc. which have received significant attention from the academia and the industry over the past two decades. These Algorithms have thus achieved a mature enough state for them to be deployed on real world applications, and yet the integrated development of a robotic system still requires their reimplementation. The integration of new hardware such as sensors, actuators or even computing hardware into the system also starts from scratch for every project, rather than building from some datum level. Most robotics developers get pulled into problems more relevant to software development before coming to core robotics development.

Another commonly encountered problem in the robotic system development life cycle is inextensible software. Any addition or modification to this system leaves the developer team with the option of either restarting at concept level or modification of existing software, which is neither modular nor standardized. Starting from concept level usually leads to inheriting the complexity at hardware, operating system and middleware levels. On the other hand, absence of scalability in design leads to loss in system stability. Modularity, extensibility and standardization have been adopted successfully by every mature industry and found to augment the development process by ensuring "re-usability", "replaceability" and "interoperability". A common approach towards achieving them is to design and develop all singular functionalities as segregated modules with standardized interfaces. All these aforementioned factors affect the growth of robotics and limit the reach of robotics for the improvement of the quality of life.

As an initial step towards the standardization and expedition of the development cycle of a robotic system, JAUS (Joint Architecture for Unmanned Systems) [9] has defined a framework and provided a set of guidelines to follow while designing a robotics system. The JAUS framework aims to reduce maintenance costs and life cycle costs, increase "re-usability" of code and provide interoperability. The JAUS framework demands platform independence, mission isolation, computer hardware and technology independence [10] for successful implementation of JAUS compliant robotic system.

JAUS framework originally provided a mechanism to allow solutions from different sources (such as vendors, manufacturers, programmers etc.) in the unmanned systems domain to inter-operate. It has evolved from being a simple message set to a well-defined framework with set topology, a standardized interface layer, and a variety of standard components. JAUS enables software to provide application level reuse for the end-user, while providing resource and module level reuse for the supplier.

The validation of JAUS concept and architecture was done through the OCU and Payload Committee (OPC) experiment [19] conducted recently i.e. 2004. The Unmanned Systems Committee, Avionics Systems Division of Society of Automotive Engineers has already adopted [11] the JAUS standard, which demonstrates the need for a universally accepted common standard in Unmanned Systems.

This paper describes software "HRS-OpenBrain" which is an implementation of the JAUS framework with modules,

Manuscript received 02 February, 2009.

functionalities and boundary definitions well suited to the autonomous mobile robot problem. The software provides a common platform for the development of robotics technology and enhances the efficiency of the research and development activities in robotics. The aim of the project is to extend the scope of robotics applications, fostering the emergence of new markets.

Details of HRS-OpenBrain architecture are given in section III, its features are explained in section IV and test results discussed in V. Finally, we conclude and discuss options of future work in section VI.

II. Background and Related Work

Mowbot in 1969 was the very first robot that would automatically mow the lawn. Beast (1961-1963) at John Hopkins University was a robotic platform for research, and could use a sonar to move around. Shakey in 1970 at Stanford was another researchers' robot base heralding the advent of autonomous mobile robots. Autonomous robots attracted public interest in 1980, through the HERO series of robots which was primarily directed at enthusiasts. The model HERO-1 was a self-contained mobile robot controlled by an on board computer with a Motorola 6808 CPU and 4KB of RAM. This robot featured light, sound, and motion detectors as well as a sonar ranging sensor [12].

The complexity of the robotic systems' development task has exponentially grown since then, fueled by the advances in the fields of sensors, actuators, computational hardware, and real time capabilities. This has led to development of various architectures for robot system control. In reference [13], Reid describes how architecture forms the back bone of a complete robotic system, and gives comparison and application areas of various software architectures like Hybrid, Subsumption and Three Layered variants, besides others.

There is a requirement for a flexible software system that provides the user with control over the implementation of variety of services such as AI algorithms, gathering of sensory information and control over actuators through commands used for robot navigation. A typical design of such software has been known to provide simulator as well as a physical hardware interface with the physical robot. Many software tools, such as Player/Stage [1],[3]-[4], MARIE[7], MSRS (Microsoft Robotics Studio)[8], OROCOS (Open Source Robot Control Software) [5], CARMEN (Carnegie Mellon Navigation Toolkit)[6], RT-Middleware [2] are widely available in the robotics community that provide many of the above mentioned services to the end user. These software environments however, are not JAUS Compliant and hence lack all the advantages associated with the JAUS framework.

Aforementioned softwares developed for robot control are mere alternatives tailored to robotics applications such as navigation or mapping, rather than being evolutionary advances in terms of utility or scope. Most of them require detailed perusal and analysis of the entire software before any use. The significant amount of work required to develop an application using these softwares, thus often encourages an informed user to build a new one. This may also be regarded as a valid reason for the existence of so many variants.

A study of above implementations suggests the need for a software environment that is flexible enough to be used by naive robotics enthusiasts for controlling various robots to amateurs for building simple intelligent machines. The same environment should be conducive to robotics algorithm specialists to develop and compare algorithms. The software environment should also serve application oriented groups to successfully create complex robotic applications.

OpenJAUS [18] is one of the first open source software to provide an implementation of the JAUS framework. It provides a library of JAUS messages, data types, entities, the JAUS Node Manager component [10] and, the JAUS Communicator [10] component. OpenJAUS is aimed at providing the developers with a ready to use JAUS interface, which can be integrated to an existing system as an, add on layer. However, OpenJAUS lacks the presence of directly interface-able hardware or software control components. Also, the mission planning and execution components, as mentioned in JAUS RA v 3.3 [10] are not available.

In the next section, we discuss HRS-OpenBrain, which is designed to obviate the aforementioned limitations and provide a complete software environment to promote and ease the absorption of JAUS into the robotics community. The objective of the project is to expedite the development of robotic systems by promoting standardization and facilitating the development of complex robotic missions.

III. HRS-OpenBrain

HRS-OpenBrain software is an implementation of the JAUS framework into a four-layered architecture, which aims at the standardization and interoperability of unmanned systems. The absence of interoperability between current unmanned systems reflects a gap between robotics research and the JAUS framework. Our vision is to bridge this gap by providing directly deployable, open source JAUS complaint robotic autonomy software.

The software is broadly divided into 4 layers as shown in figure 1.

The first layer from the top is the application development and mission definition layer. This is a layer directly relevant to the end user wherein only the robotic system's capabilities and mission specifications have to be modified to reconfigure the intended robotic system's objective.

The second layer consists of various JAUS resources such as JAUS messages, JAUS data types and JAUS component level behavior (implemented in HrsJausComponent base class). This is more directly relevant to JAUS developers, evangelists and promoters of the JAUS framework. Changes, if any, to the JAUS communication standard or

topology need to be reflected in this layer only, and the rest of the system need not be disturbed.

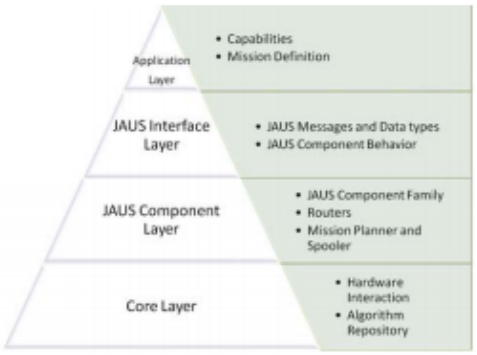

Figure 1: The 4 - Layered Architecture

The third layer contains JAUS defined Components such as Range Sensor, Path Segment driver, primitive driver, local pose sensor, etc. These are wrapper classes, which provide abstract interfaces for specific implementations of JAUS defined components to enable them for JAUS compliant communication.

The fourth layer includes specific implementations for the JAUS defined functionalities in the third layer. For simplicity, this layer is divided into two parts.

1. Hardware Interaction: Software modules that directly interact with attached hardware like sensors or actuators reside in this section of the core layer. Examples are hardware specific components such as SICK and Hokuyo laser range finding sensors being the specific implementation for the JAUS defined range sensor component. This ensures the interoperability of systems as well as easy means for component level substitution.

2. Algorithm Repository: Robotics developers and algorithm specialists are primarily concerned with making changes to existing algorithms of develop new ones according to their requirements. This section of the core layer allows for these changes. The interfaces are inherited from the upper layers and provide a "ready slot" for the algorithm implementation to occupy in the scope of the entire JAUS Framework.

IV. FEATURES

The Detailed Description of HRS-OpenBrain features are presented in the ensuing section.

A. *The JAUS Interface Layer*

This is the layer that contains the definition of common JAUS specified behavior such as components, messages and data types.

The three major sections of the JAUS framework are as follows:

a) **HrsJausComponent Base Class**: Extending the Object Oriented Software Model into JAUS implementation, the HrsJausComponent base class not only specifies the characteristics of every JAUS component but also determines its behavior. It does so by defining the state transition flow for every component at the base class level itself. Inherited component implementations just simply need to define and validate these states to achieve desired functionality. Only virtual functions need to be defined in the inherited classes so that the state transition process defined in the Base Class can be validated and iterated. Every JAUS component has a defined set of functions needed for JAUS compliance. Basic functionality, which is common to all components, is handled in this base class. Apart from that, this class also defines some virtual functions, which would be called in a specified sequential order to achieve the functionality. The base class provides the common functionalities to all the components, and already declared virtual functions can be used to implement the specific tasks (See Figure 2).

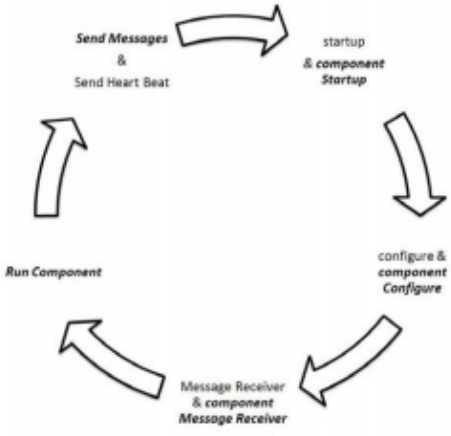

Figure 2: The State Transition Flow

b) **HrsJausMessage**: A message in JAUS protocol consists of a 16-byte header followed by specific message data. Each message is uniquely

identified by its command code [14]. The HrsJausMessage is a base class providing the header elements for all messages. All messages provided in HRS-OpenBrain inherit from this base class, which supports modularity, and requires minimum modifications in case of changes in the communication protocol.

B. *Networking layer*

Every framework with components distributed across different machines or even loosely coupled on a single machine exhibits a need for message transfer and communication. The networking layer in HRS-OpenBrain has been designed as shown in figure 3 below.

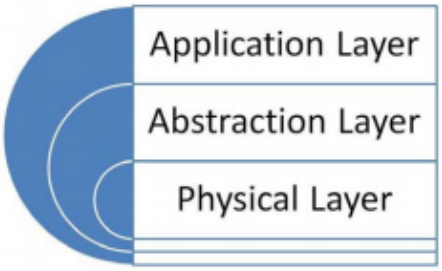

Figure 3: The Networking Layer

The Physical layer handles the system calls, and communicates the messages over the physical network. The Abstraction layer, named JAUS Message processor Utility (JMPUtil), acts as the interface between components and message sending lower physical layer. Currently, User Datagram Protocol (UDP) is used for communication and message sending, and a changeover to Transport Control Protocol (TCP) needs changes only in the physical layer. The components and JMPUtil will remain unaffected. Similarly new technological advances in network industry can be smoothly integrated, without requiring any rework. Message Queues are implemented in the abstraction layer, and JAUS requisite formatting takes place here. Changes, if any, in the JAUS message transfer standard will be incorporated here.

C. *JAUS Component Layer*

This layer contains JAUS defined components, and their common functionalities. All components are inherited from the HrsJausComponent base class and exhibit JAUS compliant characteristics and behavior. This layer also includes the JAUS framework defined message routers to ensure valid and accurate message transportation between components.

1. **Routers**: There is a need for an independent entity, which maintains and updates all subsystem configurations. This is to minimize the requirements for any a-priori information such as source and destination addresses for JAUS Messages. This need is addressed with the Node Manager and Communicator components. Transportation of messages through the JAUS defined topology is handled by these two components. A Node Manager acts as a common entry point on each JAUS Node and is responsible for a seamless communication link between working components on the relevant node and any other active nodes. Any outgoing message from a component, gets routed according to, the JAUS addresses present in the header of the message by the Node Manager. Apart from routing, Node Manager also performs JAUS defined roles like informing other Nodes about any changes in its configuration, Dynamic discovery of new Nodes, monitoring each component's state, handling of broadcast messages, service connection management, and Event processing etc. A special case of the Node Manager is the Communicator, which acts as a common entry point for a Subsystem and performs the above mentioned, roles at Subsystem level. Every Subsystem has a specific node, which has a unique Node Id. Every instance of this specific Node has a Communicator component along with the Node Manager, and all messages to or from the subsystem are routed to other Communicators through this component.

Mission Planner and Mission Spooler: The mission and capabilities of the robotic system are defined in a simple XML file. The Mission Planner (MP) and Mission Spooler (MS) components work in synchronization for completion of the mission. MP confirms the validity of mission statement in XML file, refers a capabilities matrix to validate it, and subsequently converts it into a set of mission commands transferred to Mission Spooler.

D. *Algorithms Repository*

This Feature of the Software comprises of various implementations of standard AI Algorithms for Integrated Robotic Systems development. Specific examples include Path Planners, Waypoint Drivers, Motion planners, Computer Vision, Localization, mapping and SLAM Techniques etc.

The Software design ensures that these algorithms are completely modular to allow seamless integration and easy substitution of one specific implementation with another. For example an A* path planner can be substituted with a D*[17] or a WaveFront planner with a few lines of change in a configuration file. This substitution is also possible in a running system so that the system administrator does not have to restart the entire system for just replacing the path planning algorithm. All three implementations of the Path Planner are inherited from the same Parent and therefore equipped with identical interfacing behavior.

The component behaviors themselves are designed and

implemented to exhibit the behavior of JAUS Components and talk in JAUS specified language. This also makes it possible for robotic system developers to integrate their own algorithm implementations into HRS-OpenBrain with very little thought given to software relevant issues such as interfacing, synchronization or scheduling. The approach aims at reducing the time spent to take an algorithm from the "Proof of Concept" stage, to the deployment on various robotic systems.

E. Hardware Interaction

This layer directly communicates with the hardware components that are part of the robotic system. These hardware components may be sensors, actuators, computational or communication devices that communicate in their unique formats. The Hardware Interface Layer is responsible for correctly reading, writing or controlling these devices as JAUS Components. A detailed example explaining the functionality is discussed below.

HRS-OpenBrain provides range sensors like Sick LMS200 [22] and Hokuyo URG[21] as JAUS components. *SickLMSComponent* and *HokuyoLaserComponent* lie in core layer, and both of these inherit communication interface from the *RangeSensor* class in the component layer. RangeSensor in turn inherits from base *HrsJausComponent* class in the JAUS Interface layer providing the SICK and Hokuyo components the ability to be able to register them as JAUS Components with the routing component, the Node Manager from the component layer and act as a JAUS Entity. The base HrsJausComponent class provides functionalities common to all JAUS components and the RangeSensor class facilitates the common interface for communication with any sensor, which provides range data type JAUS message. The implementations SickLMSComponent and HokuyoLaserComponent use the hardware drivers (*sickLMSDriver* and *hokuyoURGDriver* classes respectively), which interact with the hardware and give Range Data. This data is then sent across using the networking layer to its destination.

This segregation of functionality provides modularity and extensibility to the framework by easy integration/addition of new technologies. For example, to add another Range Sensor to the software, only the code, which interacts with the new sensor hardware, has to be placed in a new inherited class from the RangeSensor Component and the sensor data needs to be packed in the inherited JAUS Message, everything else is taken care by the software architecture.

V. TESTING AND RESULTS

The software system is currently being tested to control various robotic bases. An experimental navigation mission was tested on a Pioneer-P3DX robot (Mobile Robots Inc.) equipped with a Hokuyo Laser sensor. The Mission was to await the specification of a goal point in the world model, refer to the a-priori map information, plan a path and then execute it while maintaining ego localization and safety distance from obstacles. The algorithms used as waypoint drivers were VFH+[15] and dynamic window [16]. Changes in the configuration file were the only tasks required, to complete this specific implementation for navigation. Similarly, A*, D* and Wavefront planners, were used interchangeably for path planning to complete the experimental mission.

Another platform, a proprietary robot base, an AGV manufactured by Hitech Robotic Systemz (HRS) Ltd. was also used to test the ease of integration of a new platform into HRS-OpenBrain. We had to create a class, HrsRobotComponent inherited from PrimitiveDriver base class. The experiment was a success and the same VFH+ and Dynamic window components were used to control the robot base.

Observations taken to calculate latency for message transfer between different components running on same Node and on different Nodes are submitted below in Table 1.

The Mission required the presence of nine different components, before it could be validated. The Components interacting for the duration of the data collected were Node Manager, Primitive driver, global planner (Waypoint Source), HCI (System Commander), local Planner(waypoint Driver), Monte Carlo Localization (Local Pose Sensor), Range Sensor Component, Mission Planner, Mission Spooler. The System Commander was located in a different node than the rest of the components.

S no.	Time Period	Mean (milliseconds)	Standard Deviation (milliseconds)
1	Cycle Time: Range Sensor	99.608400	2.740200
2	Cycle Time Primitive Driver	99.978700	2.914500
3	Travel time Laser Sensor to Waypoint driver (local)	4.8600082	2.9734925
4	Travel time : Laser sensor to System Commander (remote)	19.301434	11.206346
5	Travel Time : Local Pose Sensor to Waypoint Driver(local)	9.8555179	2.2459904
6	Travel Time : Local Pose Sensor to System Commander (remote)	35.249981	13.152864

Table 1

In reference to Table 1, Cycle Time refers to the time period between any two sets of readings from a component. Travel time is the Latency between message transmission and reception for any 2 components.

VI. CONCLUSION AND FUTURE WORK

The HRS-OpenBrain as a software has succeeded in solving some of the identified problems. Development time for various robots at Hi-tech Robotic Systemz Ltd. India, has been successfully reduced. The inheritance based implementation makes it readily usable in robotics research, allowing the researcher to focus on robotics, rather than developing new software or understanding all layers of

HRS-OpenBrain in detail. Extensible design allows easy integration of new code as a component. Division of components in child tasks and parent tasks helps formulate complex mission behaviors using the mission planner. Sophomore students who have been given the software, along with a software reference manual, and an A* implementation, were able to integrate it as a component within a couple of hours.

Our intention is to make HRS-OpenBrain de-facto software of choice for robotic application development. For that considerable scope for further work on addition of Real Time hardware control components still exists. Addition of components for the control of a manipulator arms, augmenting the Algorithm repository with newer and better performance algorithms for any and all types of robotic problems forms a large part of our to do list.

ACKNOWLEDGMENT

We thank the team at, Hitech Robotic Systemz Ltd for funding this dream project and all their support in infrastructure, SDLC knowledge for its development. Especially, Dr. Anirban Das and Dr Madhusudan for their valuable suggestion.

We would also like to thank, Debadeepta Dey for his valuable inputs without which our dream software would not have become a reality.

REFERENCES

[1] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In Proc. of the Intl. Conf. on Advanced Robotics (ICAR), pages 317– 323, Coimbra, Portugal, July 2003.

[2] Noriaki Ando, Takashi Suehiro, Kosei Kitagaki, Tetsuo Kotoku, and Woo-Keun Yoon. RTMiddleware: distributed component middleware for rt (robot technology). In IEEE/RSJ International conference on robots and intelligent systems, pages 3555–60, Edmonton, August 2005.

[3] Brian P. Gerkey, Richard T. Vaughan, Kasper StÃ¸y, Andrew Howard, Gaurav S Sukhatame, and Maja J MatariÃ´c. Most Valuable Player: A Robot Device Server for Distributed Control. In Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), pages 1226–1231, Wailea, Hawaii, October 2001.

[4] R. T. Vaughan, B. P. Gerkey, and A. Howard. On device abstractions for portable, reusable robot code. In Proc. IEEE International Conference on Intelligent Robots and Systems, volume 3, pages 2421–7, October 2003.

[5] OROCOS. Open Robot Control Software Open Robot Control Services. http://www.orocos.org, 2005.

[6] Michael Montemerlo, Nicholas Roy, and Sebastian Thrun. Perspectives on standardization in mobile robot programming: The Carnegie Mellon Navigation (CARMEN) toolkit. In Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), volume 3, pages 2436–2441, Las Vegas, NV, October 2003.

[7] Carle C¨ot´e, Dominic L´etourneau, Franc¸ois Michaud, Jean-Marc Valin, Yannick Brosseau, Cl´ement Ra¨ıevsky, Mathieu Lemay, Victor Tran "Code Reusability Tools for Programming Mobile Robots" In Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), 2004.

[8] Microsoft (2006): Microsoft Robotics Studio Runtime–An Introduction. http://msdn.microsoft.com/robotics/getstarted/runtime/default.aspx

[9] JAUS, Joint Architecture for Unmanned Systems, http://www.jauswg.org/

[10] JAUS Reference Architecture V3.3 Part-1, June, 2007, http://www.jauswg.org/archive/ref_architecture/ref_architecture.shtml

[11] JAUS History and Domain Model, As-4 Unmanned Systems Committee, http://www.sae.org/technical/standards/AIR5664

[12] Mobile Robotics, History, http://en.wikipedia.org/wiki/Mobile_robot

[13] Eve Coste-Maniere, Reid Simmons "Architecture, the backbone of Robotic Systems" In Proc. of the IEEE Intl. Conf. on Robotics and Automation,April 2000.

[14] JAUS Reference Architecture V3.3 Part-2, June, 2007, http://www.jauswg.org/archive/ref_architecture/ref_architecture.shtml

[15] I. Ulrich, J. Borenstein, "VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots", Proc. 1998 IEEE Intl. Conf. on Robotics and Automation (ICRA98), pp. 1572 - 1577, 1998

[16] D. Fox and W. Burgard and S. Thrun, "The Dynamic Window Approach to Collision Avoidance", Technical Report, University of Bonn, IAI-TR-95-13, 1995.

[17] A. Stentz. "Optimal and efficient path planning for partially-known environments". In Proceedings of the IEEE International Conference on Robotics and Automation, May 1994.

[18] OpenJAUS v3.3.0a,2008, http://www.openjaus.com/

[19] OCU and Payloads Committee, OPC 3.0, Experiment Overview, November 2004, www.jauswg.org/archive/other_docs/OPC%203.0%20Overview.pdf

[20] Prof Schmidhuber's highlights of robot car history, http://www.idsia.ch/~juergen/robotcars.html

[21] Hokuyo-URG Range sensor, http://www.hokuyo-aut.jp/02sensor/07scanner/urg.html

[22] Sick LMS 200, http://www.sickoptic.com/home/factory/catalogues/auto/laser_measurement_systems___robotics/lms-400/en.html