A modular approach to system integration in underwater robotics

Tomislav Lugarić, Đula Nađ and Zoran Vukić

University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia

Abstract — **Underwater exploration is experiencing an ever increasing usage of both autonomous and remotely operated robotic vehicles. Vehicles are typically equipped with several sensors depending on the mission type being carried out. Integration of all these systems together, as well as obtaining data from them presents a significant challenge. This article describes a software architecture which represents all entities in the system as equivalent modules, while hiding their specifics from the user. Each module in the system is fully defined only by the data it produces and the commands it accepts. Modules are coupled together using a communication interface which again hides the underlying protocol from the user, and is based solely on message exchange. This architecture allows quick reconfiguration of the vehicle, easy integration of various sensor systems and provides the application developers with a higher level of abstraction.**

I. Introduction

SOFTWARE interfaces are a common way of improving potential for code reuse, simplifying cooperation between multiple programmers on the same project and making code more readable. Classes used in object oriented programming have the benefit of grouping data together with the methods that utilize it. At the same time, classes improve the security of the code because they can prevent illegal operations on data. When designing classes that will be used in the system, the public methods define how its functionality will be exposed. For good modularity, the public functions should all have the same level of abstraction [1]. Class interchangeability is achieved using interfaces. Interfaces specify the methods a class should implement without providing any implementation details. A class which implements an interface guarantees that it can be used in a context where such an interface is required. The class can implement more than one interface, and the interface that is used depends on the context [2]. Code that relies on interfaces instead of classes is more flexible because it relies only on data that is passed to an operation and received as its result. The improved modularity of the code reduces the effort needed to integrate different pieces of code and enables utilization of various programming languages in the same project. The higher level of abstraction allows developers to focus on solving problems at a higher level without worrying about low level problems such as pointers or protecting critical data.

Quality of Experience (QoE) measures how satisfied a user is while using a system. Unlike Quality of Service (QoS), which measures the technical aspects of the system such as network throughput, availability and speed, QoE involves more psychological measurements. A typical QoE measurement will involve statistics such as the number of users that try the system out and continue to use it, time needed to perform a particular task and the number of mistakes the user makes before completing the task. A system with a high QoS does not necessarily have a high QoE, but QoS is a necessary prerequisite for QoE. Usefulness and usability [3, 4] are additional requirements for good QoE. Usefulness implies the potential of a resource to provide useful data and/or services. Usability shows the resource's actual ability to provide it. The measures of usefulness and usability are typically used in web design, but most of the principles can be applied to hardware/software architectures. In the context of hardware/software architectures, QoE implies ease of hardware deployment, reconfigurability and expandability. Usefulness is the architecture's capability to incorporate an array of sensors and acquire data from them. Usability is the flexibility of data presentation, capability for custom processing and a measure of the effort required to reconfigure the system.

This article describes the requirements for improving the QoE for integration of multiple sensors on underwater vehicles and development of underwater vehicle control systems.

II. Problem statement

A typical underwater mission includes multiple sensors installed on a single vehicle or multiple cooperative vehicles. Multiple sensors allow the vehicle operators to obtain a better quality of data as well as to perform multiple tasks in one dive. Vehicles may be used in missions that require different sets of sensors. Bottom mapping may require additional sonars to be installed; monitoring marine habitats may require additional cameras or chemical sensors. Sensors installed on the vehicle are typically made by different manufacturers and therefore need different software packages for data acquisition. Before all sensor datasets can be fused together, they need to be transformed into a standard format so they can be processed together. In case timekeeping systems on different sensors are out of sync, special care needs to be taken in order to properly synchronize different logs.

Synchronizing different logs can be solved by developing a central logging application, deployed either on a computer installed onboard an autonomous vehicle or a computer connected to the control panel of a remotely operated vehicle. The central logging application can use its own timekeeping system, eliminating the need for log synchronization. By performing additional calculations and conversions, data can immediately be standardized. The disadvantage of this approach is apparent in case a piece of equipment is installed on a vehicle for the first time. Before it can be used with the vehicle, the logging application needs to be expanded to accept data from the new sensor. This makes transferring sensors between vehicles time consuming and reduces the vehicle's reconfigurability.

The goal of the system described in this article is to improve the reconfigurability of the vehicle and raise the quality of experience (QoE) of its operators without imposing restrictions on the functionalities of the sensors.

III. System architecture

The system described in this article is designed to be modular, loosely coupled and message based. The primary goal while designing the system is to shift the focus sensor specifics to sensor data. Loose coupling is achieved because each entity in the system is completely described by the data it provides (variables) and the data it accepts (commands). Modularity of the software is designed to closely resemble the modularity of the entity's hardware. By modular software we mean that it is easily expandable with new sensors, control panels or additional processing. Loose coupling means that the system is robust because a failure in one of the modules does not necessarily prevent other modules from functioning properly. Since all communication in the system is based on messages, and a message can be read by several modules, it is easy to send the same data to multiple modules at once.

An example system is shown in figure 1. There are three basic types of modules in the system:

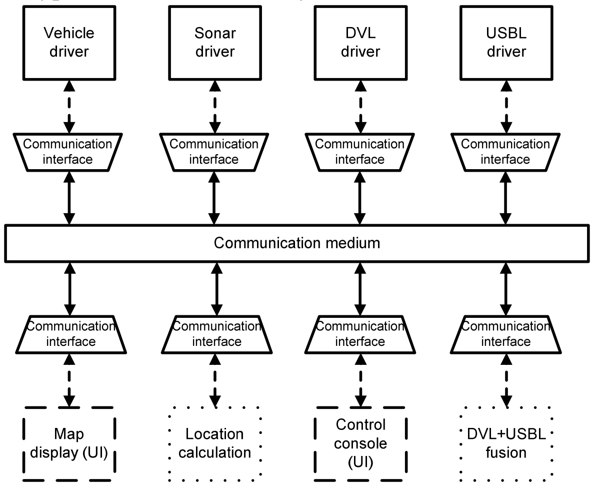

Figure 1: Architecture of system for vehicle system integration

driver modules (represented by solid black boxes), supporting blocks (represented by dotted boxes) and user interfaces (represented by dashed lines). All modules in the system are connected by a communication medium. The communication medium can be implemented using any technology and is used as a logical bus. Communication medium must implement operations specified by the communication interface abstract class.

IV. Communication medium and interface

The communication interface is a purely abstract class, meaning it has no implementations of any communication operations. The communication medium represents the mean through which the communication interfaces communicate with each other. In case of a P2P communication system, the medium will represent the network connecting all nodes together, while the appropriate communication interfaces will take care of routing messages from source to destination(s). In case of a publish/subscribe system, the communication medium will include a repository for messages, while communication interfaces will handle registration for incoming messages and publishing of outgoing messages. The developer of the system does not need to take care of the recipient of a message, and each module can forget about a message as soon as it is passed to the communication interface. The recipient is responsible for receiving the message and handling it.

In the system, communication between communication interfaces and the communication medium is carried out using standardized messages which are best suited for the medium in use. Mediums using textual communication protocols may use XML or similar markup languages to pass messages, while lower level protocols might use arrays of bytes. Communication between modules and communication interfaces is done using appropriate programming functions specified in communication interface abstract class.

The communication interface can have two modes of operation: active and passive. In case of a passive communication interface, the module utilizing it must poll the interface for incoming data when it is ready to process it and pass outgoing data to the interface. In case the interface is active, the module only needs to register event handling functions for incoming and outgoing data. The active interface will poll the module for outgoing data when ready to transmit it and will pass incoming data to the module when it is received. Data can be exchanged between the communication interface and the module in three formats: byte array, named datamap and XML text. When sending data, the interface will accept any of the three formats. When receiving data from the communication interface, the developer of the module must specify which format the module will accept. The developer has no control over the actual format used to transmit messages since this format is determined by the type of communication medium.

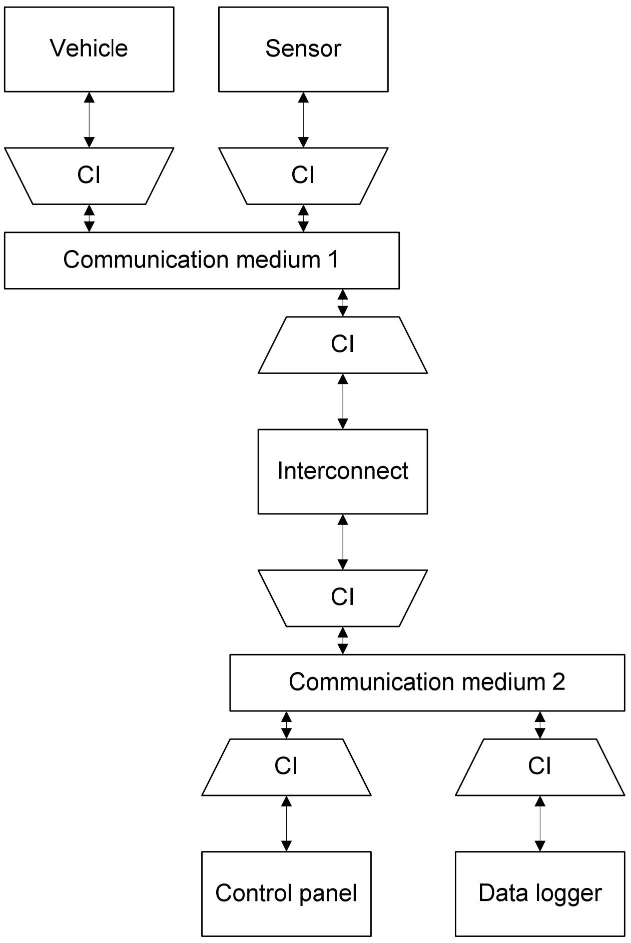

Figure 2: Usage of two communication mediums

In case two connection mediums must be used in the system, a bridge between them should be deployed. Figure 2 shows an example of a system using two communication mediums.

The first communication medium may represent a device specific bus that is used on the underwater vehicle, and the second one may represent a local area network connecting the vehicle's console with computers processing data. Corresponding modules are connected to their respective communication mediums in the same manner as in figure 1. Additionally, an interconnection module is added that is connected to both mediums. Since the data formats the two communication interfaces provide is compatible, the interconnection block only needs to forward data between them. If additional data processing is needed anyway, it can be implemented inside the interconnection block.

V. Driver modules

Driver modules are software abstractions of respective pieces of hardware. Example of a driver module is shown in figure 3.

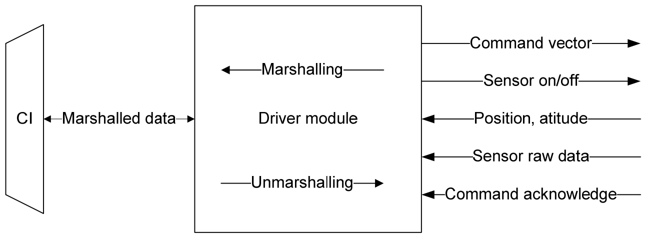

Figure 3: Driver module

Each driver performs marshalling of the data that is sent from the device towards the communication interface and unmarshalling of the data received from the communication interface. Umarshalled data received from the communication interface will include commands for turning various features of the hardware on and off, i.e. thruster power, fin settings and sensor adjustments. Data that is being marshaled prior to sending to the communication interface will include hardware status and raw data.

Driver modules convert device specific commands and data to a standardized format. This has the benefit of allowing devices made by various manufacturers to be easily integrated and swapped. All device driver modules of the same type must send a defined group of datasets for maximum interchangeability. The module using the data from a sensor can only assume that those datasets exist, while additional datasets can be used only after verifying they are present. Additional datasets may be sent if the device provides additional data. For example, every attitude sensor must send yaw, pitch and roll values. It may also send temperature, drift, accelerations in x, y, and z directions. The module that receives this data will rely on yaw, pitch and roll data. The additional datasets may be included in calculations the receiving module performs, but the developer should check if they are provided. If they are not provided, the developer can either make calculations without them, or provide default values that do not impair calculation to be used when no real data is available.

VI. Supporting modules

Supporting modules contain calculations that perform conversions of data, filtering, corrections or sensor fusion. These additional processing operations are kept separate from sensors in order to make the system more modular. If the processing needs data from several sensors, integrating data processing with them would mean coupling the two sensors firmly together, impairing the modularity of the system. In cases where a module needs data from only one sensor, it is still beneficial to separate sensor driver and processing for greater code flexibility. With the processing code in a separate supporting module, the developer has the choice of using the supporting module with any sensor, to combine it with other processing modules or to not use it at all, reading only raw sensor data.

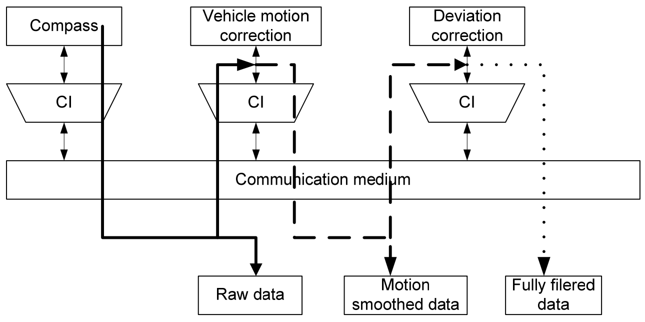

Figure 4: System with support modules

Similarly to driver modules, supporting modules perform unmarshalling of the data that is to be processed and marshalling of the processed data. Unmarshalled data is converted into the format suitable for processing and is either processed by the code directly in the module or is passed to an external application such as MATLAB.

Figure 4 shows an example of a system with supporting modules and different possible data flows. In this example, the driver module for a compass installed on a robotic vehicle provides heading data. Since the compass is prone to magnetic deviation, and is affected by vehicle motion, two modules that correct those errors have been added into the system. The raw heading data is marshaled, sent into the system, and can be used by listening for the corresponding message, indicated by the solid arrow. The motion correction module listens for the message and performs processing by stabilizing the reading depending on the movement of the vehicle. For simplicity, messages providing data about vehicle motion are not shown. The message containing the corrected data is indicated by the dashed arrow. The deviation correction module listens for the message providing smoothed data from motion correction module and corrects the heading using deviation data, obtained from a unit such as GPS (message not shown). Corrected data is indicated by the dotted arrow. Messages containing raw data and motion smoothed data are not consumed because two modules read them. All three messages are available to any modules that are eventually added to the system. The developer or the operator may choose to use raw data and filter it in some new module, to use partially or fully filtered data.

VII. User interface modules

User interface modules are designed to be deployed on computers which are used to operate or monitor the system. Typically the user interface modules will have a graphical user interface (GUI) to display sensor data and vehicle status and accept inputs from the user. Interfaces may also have support for other input methods such as joysticks or keyboards and logging functionality. Unmarshalled data received from the communication interface is analyzed and converted to the appropriate format to be read by the GUI. A diagram of the user interface module is shown in figure 5.

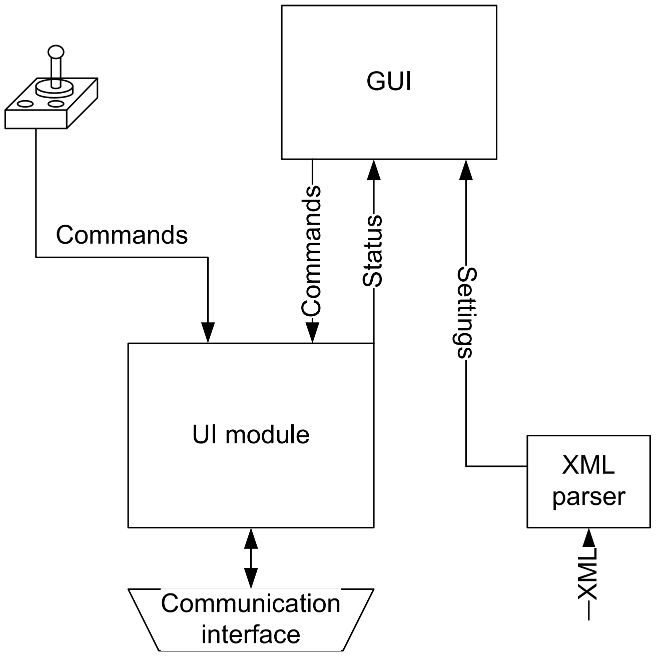

Figure 5: UI module

User interfaces have an XML document parser which is used to load user settings for them. Inside the XML document, the user can specify variables to display, variables to be read from the GUI or from human interface devices (e.g. joystick) and the indicators used to display variables. The XML document can be written manually or automatically generated using a tool for designing user interfaces. When designing the GUI, user selects one of the available GUI elements and specifies its parameters and the variable or variables the GUI element is bound to. If a variable is defined as output, the value of the variable is sent to the GUI element which displays it. If a variable is defined as input, its value is read from the corresponding GUI element. Input of values can be done from keyboard and joystick as well. Joystick and keyboard support can be handled by the same UI module that handles the GUI functionality, or can be handled by a separate module.

Multiple user interface modules may be attached to the system and configured for each user individually. This way, the pilot of the vehicle can configure his display to show navigation data such as maps, obstacles, speed and heading, while other experts observing the same mission can configure their displays to display data relevant to them.

VIII. Example implementation

An example implementation of a modular control system was developed at the Faculty of Electrical Engineering and Computer Science in the Laboratory for Underwater Systems and Technologies. The system is designed for controlling an Iver2 AUV over WLAN while on surface. It can be used to pilot the AUV into position prior to starting a mission, return it from the last mission waypoint to the dock or ship to be retrieved or to monitor Iver's vital statistics while it is performing a surface mission. Two communication mediums are in use: the Mission Oriented Operating Suite (MOOS) [5], and a serial interface.

The driver module for the Iver2 AUV sends messages about vehicle's power system (battery), attitude and position (compass, GPS, IRS, depth) and about the current state of the mission in progress, if any. The commands that can be sent to the driver include thruster/fin settings for direct control, course, speed and depth for a semi-automatic control and load/start/stop mission for fully automatic control. Iver2 has two computers – frontseat and backseat. The frontseat computer controls the AUV, while the backseat computer is used for additional applications.

The MOOS system is a publish-subscribe message exchange environment. It uses a central server on which all clients publish their messages. The clients also subscribe for the messages they wish to receive, and the server forwards the requested messages to them. The MOOS server is deployed on the backseat computer onboard Iver2. Any computers that wish to access data from the system must connect to Iver's wireless network. Figure 6 shows the physical structure of the system.

In the system, the physical communication medium between backseat and shoreside PCs is the MOOS running over Ethernet. Serial interface is used as a communication medium between backseat and frontseat PCs. The backseat PC runs Iver's driver module, a support module and an interconnection module. The vehicle driver module converts data between standard messages that are used in the system and NMEA messages used by Iver's control software on the frontseat computer. The support module logs data about the AUV and stores it on the disk of the backseat PC. The interconnection module connects the MOOS communication medium and the serial communication medium. Since the connection between backseat and shoreside is broken upon diving, it is beneficial to deploy all logging and critical processing modules on the backseat computer and leave only the control modules on shoreside PCs. In this example implementation, only the user interface module is running

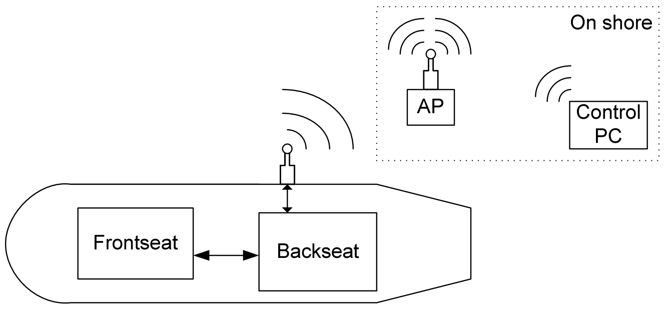

Figure 6: Iver 2 control system – hardware architecture

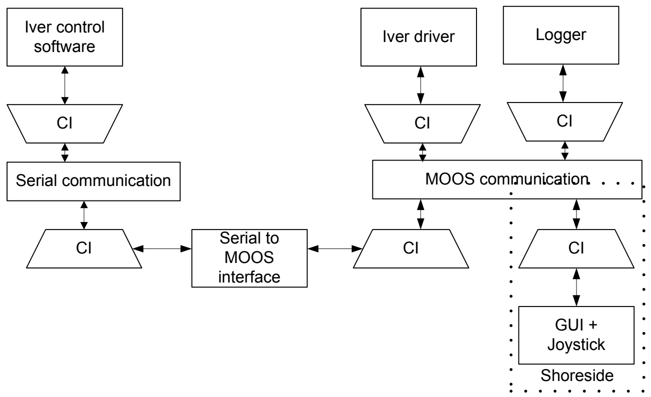

Figure 7: Iver 2 control system - logical architecture

on the shoreside PC. Figure 7 shows the software architecture of the system.

The entire system depicted on figure 7 is physically located on Iver2, except for the part marked by the dotted line. The MOOS communication medium hides the physical separation of backseat and shoreside PCs, making the system easier to expand. The only weak point of the system is the fact that the MOOS server is located on the Iver2 and is unavailable to the shoreside PCs during underwater missions.

In the future, it is planned to use a similar system with other vehicles owned by the Laboratory. Driver modules should be developed for all ROVs, sonars, acoustic modems and sensors available, enabling the deployment of a versatile, expandable and highly flexible robust system.

IX. Conclusion

This article has reviewed a control architecture proposal that offers a polymorphic view of control systems. Devices and protocols can be exchanged without direct intervention into the system kernel. This eases software development and users can focus on developing modules rather than whole systems. With this improvement of the developer's QoE, more time can be invested in testing and research as less is needed for software support.

Passing data encoded in XML is often encountered in web architectures but less so in low level control systems. This is usually because of efficiency concerns. However, if soft real-time operation is enough we find that the additional overhead is negligible compared to benefits. Wide range support exists for XML, therefore, we can easily interface modules written in different programming languages. This increases code reuse.

Increased modularity of the system as well as abstraction of all entities makes reconfiguration an easy and quick process. This in turn improves the end-user QoE because it is easy to quickly remove or add sensors to the system, use a different vehicle or to reconfigure the operating console for a different type of mission.

Future work will continue the development and optimization of the architecture. Focus will be on creating an architecture that would provide support for development of software solutions from low level control systems up to mission control and monitoring systems. The reconfigurability of the system will be facilitated by usage of XML-based configuration files that will define the connections of the entities in the system and allow for quick ad-hoc reconfiguration.

Acknowledgment

The work was carried out in the framework of a Coordination and Support Action type of project supported by European Commission under the Seventh Framework Programme "CURE – Developing Croatian Underwater Robotics Research Potential" SP-4 Capacities (call FP7–REGPOT–2008–1) under Grant Agreement Number: 229553.

References

[1] McConnel, S, Code Complete, Second Edition, Microsoft Press, 2004.

[2] Mayo, J; C# Ekspert; Miš, 2002. Zagreb

[3] Srbljić, S., Škvorc, D., Skrobo, D., Widget – Oriented Consumer Programming, Autimatika 50(2009), 3-4, str 252-264

[4] Škvorc, D., Programiranje prilagođeno potrošaču, PhD thesis, , University of Zagreb Faculty of Electrical Engineering and Computing

[5] MOOS: Main ; Oxford Mobile Robotics Group; http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php; 18. 1. 2011