NEPTUS – A FRAMEWORK TO SUPPORT A MISSION LIFE CYCLE

José Pinto, Paulo Sousa Dias, Rui Gonçalves, E. Marques,
Gil M. Gonçalves, João Borges Sousa, F. Lobo Pereira
{zepinto,pdias,rjpg,edrdo,gil,jtasso,flp}@fe.up.pt

LSTS – Underwater Systems and Technology Laboratory
Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias s/n
4200-465 Porto, Portugal

Abstract: This paper presents the mixed initiative infrastructure that is being developed and used at the Underwater Systems and Technology Laboratory to plan and control missions of cooperating teams of heterogeneous vehicles. The system is composed by various modular components that can be adapted to serve different mission objectives and various forms of operation. This paper also documents how the system has been validated in various test missions that were executed exclusively with this software infrastructure. The system is now evolving to support new mission concepts and added requisites like simulation services and live data publication techniques.

Keywords: Command and Control Systems, Systems Engineering, Communication Networks, Autonomous vehicles.

1. INTRODUCTION

Various ongoing projects at Underwater Systems and Technology Laboratory (USTL) deal with different types of autonomous and semi-autonomous vehicles. Our final objective is the creation of a networked system of vehicles, sensors and human operators that cooperate with one another by planning sequences of actions that allow them to achieve one or various shared goals (Sousa *et al.*, 2003).

We are currently working with two autonomous underwater vehicles (AUV), two remotely operated vehicles (ROV) and a wireless sensor network~~s~~ (WSN) ~~technology~~system. Our collaboration with other institutions, like LSA[1] and the Portuguese Air Force, also adds autonomous surface vehicles (ASV) and unmanned air vehicles (UAV) to this list.

~~Using~~ The use of different kinds of vehicles, belonging to different institutions, certainly leads to various communication protocols, different specialized human operators and multiple integration problems.

~~Having~~ In order to deal with this problem, ~~s in hands,~~ USTL researchers are developing (and testing in real-world missions) a software infrastructure that abstracts concepts like vehicles, consoles, operators, plans, missions and communication protocols in a way that allows the rapid definition of abstract missions which can be executed by the various USTL and non-USTL vehicles. The system is composed by the Neptus framework (Dias *et al.*, 2005; Neptus, 2006) and the Seaware publish/subscribe communication layer (Marques *et al.*, 2006).

The Neptus framework ~~is made up of~~comprises various software modules like an environmental map editor, a mission planner (plan editor), configurable operator consoles (composed by various visual components), 2D and 3D world state ~~-~~viewers, inter-component communications and mission revision. T~~Used t~~ogether, these modules ~~allow the~~enable the definition of abstract missions by planning individual vehicle plans and by specifying environmental maps (Dias *et al.*, 2006a, b). The abstract missions are translated into the vehicles' native languages using a translation service based on eXtensible Stylesheet Language Transformations (XSLT) technology. While a vehicle is executing one of the generated missions, its actions can be monitored and/or controlled by multiple operator consoles

[1] Autonomous Systems Laboratory, Porto's Engineering Institute (IPP, ISEP)

simultaneously using the Seaware communication layer.

The Seaware communication layer is a publish/subscribe system that is used to communicate transparently to any node that is registered in the network. Nodes can either be vehicles that publish sensorial data and receive operator commands or consoles that subscribe the vehicles' data while publishing operator commands.

Since the system is being developed to support real-world missions it has been thoroughly tested not only in its feasibility but also in its efficiency and interoperability. Without such system, various past missions certainly wouldn't have been feasible.

This paper is organized as follows. In section two, related projects are evaluated for its relevance with our system and their solution to similar problems. On section three, the overall system architecture is presented along with various possible operational scenarios. The communications infrastructure is documented in section four and the possibility to easily define new operational consoles is explained in section five. In section six, we show the results of various test missions that were executed using this infrastructure exclusively. Finally, ~~i~~in section seven, we present some conclusions and possible future developments.

2. RELATED WORK

Typically, software applications that support robotic vehicles operations take the form of operational consoles. There are several examples of this type of applications, being one example the Remus AUV (Hydroid Inc. 2006) console. This console is able to display vehicle trajectories and provides the ability of defining new missions, but no support is given in communications with other devices or vehicles.

NPS AUV Workbench (Lee 2004), from Naval Postgraduate School, is capable of managing various vehicles in a cooperative manner. All mission visualization and definition is ensured by 2D and 3D (VRML) visualizations which can be monitored in a standard web browser. With this system it is very easy to replay missions or to do mission rehearsals.

Although AUV workbench was designed for use with multiple vehicles it does not allow cooperative missions using hybrid systems concepts. On-going projects at USTL require vehicle interactions in the context of hybrid systems (Dias *et al.*, 2006a). The main target is to design a framework that allows cooperative mission planning and visualization.

Another project ~~that is~~ trying to create a communications infrastructure for robotic operations is PLAYER (Gerkey *et al.*, 2001). This project provides a server ~~where~~ relaying all data from existing systems (robots, operators, sensing devices …), thus enabling ~~is relayed~~. ~~The objective is to share~~ the entire world state to be shared among the existing systems. Some systems send data as publishers and others (subscribers) get notifications of topic updates. STAGE is being developed in parallel to provide means for visualization of the entire world state.

3. SYSTEM OVERVIEW

The Neptus Framework is composed by various existing applications and modular software components that can be merged by developers to rapidly build new Neptus applications.

Some of the most relevant modules are the Map Editor, Mission Planner, Mission Processor, Configurable Console(s), Variable Tree, Renderer2D, Renderer3D and various other Visual Components. For inter-console and vehicle communication, the Seaware publish/subscribe system is used. ~~Following~~Next, the roles of various Neptus modules are explained, how they are (re)used across different applications and how these applications can be applied for planning and monitoring real-world missions.

The Mission Map Editor (MME) component is a GIS-like application that allows the creation of three dimensional environmental maps, storing these maps as an XML file. One or more of these maps can later be included in an abstract mission, helping considerably in the planning and execution phases. Additionally, the environmental maps ~~are~~ might be sent to vehicles when they ~~know~~ are able~~how~~ to handle this type of information.

The Mission Planner (MP) module is a ~~top~~top-level application that can be used to define the various components of a mission that is to be executed by one or more vehicles ~~in the future~~.

The result of using the MP application is a XML file that holds an abstract definition of the mission plan. A mission plan is composed by environmental maps (links to other XML files), individual plans (a graph with nodes representing maneuvers and transition conditions between them) and additional data like local information and checklists of required material or initial tests.

The Mission Processor (MProc) module knows how to translate mission files (XML) to the different supported vehicles' native formats. This module can ~~though~~ thus be used to generate a vehicle-specific mission file which is then sent to the vehicle for execution.

After a vehicle starts executing a mission, its state can be monitored by using a generic console that shows all information that flows on the network or a specific console that matches the type of vehicle or mission.

Specialized consoles can be defined by using the Configurable Console (CC) module. This module is composed by an empty window that serves as a canvas for adding various visual components. The visual components can be connected to a specific variable that ~~may~~ might be available on the network like the vehicle's orientation~~, vehicle's~~ and position, motor RPMs, etc. At the end, a XML file with all the performed configurations is saved, allowing future reuse.

A Variable Tree (VT) module exists in every console to store the incoming network data and provide a generic access to these values. This module stores the variables in a tree structure, triggering events in every affected branch whenever a value is updated. This simple scheme allows the easy definition of system alerts by defining scripts that run whenever a variable or a variable domain is updated.

To visualize the entire world state, Renderer components are connected to the VT module. Currently, there exists a two dimensional (R2D) and a three dimensional renderer (R3D) that can be used simultaneously. The later is extremely useful in ROV operations when inspecting underwater structures but, on the other hand, the R2D module is also useful for missions that take place over a large area. Additionally, R2D is used also for map edition, allowing the user to interact with the existing objects (images, geometries, paths, marks …).

After executing a mission, the received data can be later revised by using the Mission Review and Analysis (MRA) application. This application can replay the vehicle's actions and display various graphical representations of the sensors data.

An overview of a possible operation scenario, where multiple consoles interact with more than one vehicle using the same communication infrastructure is represented in Fig. 1~~Fig. 1~~.

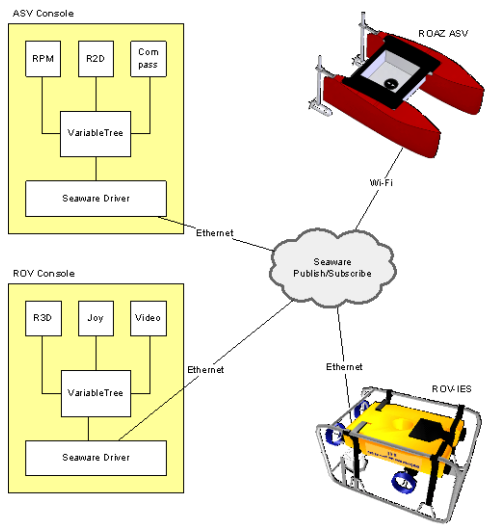

Fig. 1. Possible operational scenario

4. COMMUNICATION INFRASTRUCTURE

To facilitate the integration of this system in several classes of vehicles, ~~though~~ thus minimizing the problems of proprietary formats, we adopted XML for most of the stored information. ~~There are~~One of the several advantages of this option~~in its us~~ eis ~~like~~ the possibility to define a grammar for validating every loaded file. Once we have a grammar~~.~~, we can also ~~say to~~specify ~~potential users~~ the exact file format we are expecting from potential users. XML can also be filtered and transformed into different formats like text, HTML or any kind of mission format of some existing vehicle.

This is exactly what we do for each vehicle ~~that is~~considered ~~added~~ in Neptus. When we add a vehicle, we must provide the list of its feasible maneuvers ~~that it is capable of doing~~ (in an XML format), and, in addition, a XSLT stylesheet giving the transformation rules from the XML to the vehicle's mission language.

XML serves as the base for all data transfers, storage and manipulation in Neptus. We also use XML to define the messaging between Neptus components and external ones. There is a base set already defined, but the extension of this set is very easy.

~~To add~~In order to include a new type of message~~s~~ ~~to~~ in Neptus all ~~that has~~what needs to be done is ~~the addition of~~to add ~~some~~ a few lines to a XML file. That file ~~has~~ constains the definition of all messages that will be able to be read by Neptus. We have a fixed message header and the body can be composed by several fields from the set of ~~several~~ provided types (Table 1~~Table 1~~).

Table 1 Native types

Type	Length (in bytes)
int8, int16, int32 (all signed)	1, 2, 4
uint8, uint16, uint32 (all unsigned)	1, 2, 4
fp32, fp64 (floating point)	4, 8
~~Rawdata~~rawdata, plaintext (first 2 bytes indicates the length)	minimum 2 and maximum $2 + 2^{16}$ (65537)

Fig. 2~~Fig. 2~~ shows an example of a message definition that could be used to communicate the motor state of a pseudo-vehicle. The message has an Id, a name, and some fields. The only thing we must attend is that the message Id must be unique to all the components using Neptus messaging.

```
<message id="3" name="Motor" abbrev="Motor">
    <field name="Identification Number" abbrev="id"
           type="uint8_t" />
    <field name="Pulse Width Modulation" abbrev="pwm"
           type="fp64_t" unit="%"/>
    <field name="Tension" abbrev="u" type="fp32_t" unit="V"/>
    <field name="Current" abbrev="i" type="fp64_t" unit="A"/>
    <field name="Rotations per minute" abbrev="rpm"
           type="fp32_t" unit="rpm"/>
    <field name="Temperature" abbrev="temp" type="fp32_t"
           unit="ºC"/>
    <field name="State" abbrev="state" type="uint8_t"/>
</message>
```

Fig. 2. Message example

The messaging platform that is used is Seaware (Marques *et al.*, 2006), which provides communication between Neptus and vehicles in the networked environment. Seaware is a ~~,~~ publish/subscribe based middleware, which serves IP-based Wi-Fi/Ethernet or underwater acoustic modem communication setups.

The publish/subscribe mechanism allows dynamic pairing of peers in the network according to named message types, known as *topics*. Within Neptus, each vehicle console defines a set of published topics and another of subscribed topics corresponding to the various message exchanges for control of one or more vehicles.

For IP-based communications, integrating new components in the run-time network environment is transparent; Seaware provides that support through a Real-Time Publish-Subscribe (RTPS) protocol back-end, with built-in support for dynamic coupling of peers addressed by topic. It is possible to have distinct Neptus instances interfacing with the same vehicle, with possible generalization to many-to-many (consoles/vehicles) communication.

5. CONFIGURABLE CONSOLES

With the increased variety of vehicles, and its sensors, to be monitored and controlled, it became obvious the necessity to create one application where it would be possible to build and configure consoles. To achieve this goal, the communications process had to be reformulated and ~~one~~ an event communication system, internal to Neptus~~,~~ was ~~demanding~~required.

The internal Neptus event communication system is based on a tree structure, where nodes indicate the subject of data values ~~that are~~ in the ~~,~~ leafs (Fig. 3~~Fig. 3~~). Neptus visual components can become listeners of a single variable (tree leaf) or a defined variable domain (tree branch). Whenever a message arrives from Seaware, ~~,~~ its data is stored into the tree at the right branch according to the XML definition of that message. In t~~T~~his way, all visual components listeners are informed of incoming network middleware data. In a similar way, output data (like joystick) is sent to middleware by Neptus console components through the variable tree. ~~Even between Neptus local components can be applied event communications using~~ T~~t~~he variable tree system can also be used for event communication between Neptus local components.~~.~~

With~~Having~~ an inter-component communication system and several visual components in place, the creation of a visual builder application ~~for creating~~to generate ~~,~~ specialized consoles ~~,i was~~ the next ~~logic~~ natural step.

There are two important state modes in the Neptus generic console builder application: Editing mode, and Operational mode. In editing mode, the palette of available components (compass panel, renderer panel, RPM panel, video panel …) becomes visible, offering the user the possibility to add and place components freely inside console main panel. To configure and connect the panels to the variable tree system, the user can alter the component properties using a dialog box, in editing mode. When all components are ready, correctly placed and connected to the system variable tree, the user can change to operational mode where the components become fixed in the console and start to respond to user interaction (mouse clicks, key presses …).

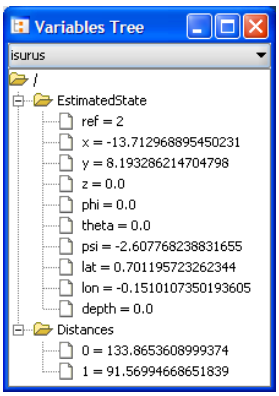

Fig. 3. Neptus Variable Tree used to centralize information related to one system (Isurus AUV in this case)

Currently, every vehicle present, or configured~~,~~ in Neptus has its own specialized console; it is so mainly for commodity. By opening a mission file in Neptus, it is possible to access the matching specialized operational consoles. For instance, when a mission includes a plan for the vehicle Isurus, it is possible to open a TrackerConsole (acoustic tracking of the vehicle) or an IsurusConsole (a console that receives the Isurus state over Wi-Fi). All these consoles are based in the base operational console that receives all network data and presents using a R2D and R3D components. The base console builder is able to respond to a variety of operational scenarios, even monitor and control several systems simultaneously, as shown in Fig. 4~~Fig. 4~~.

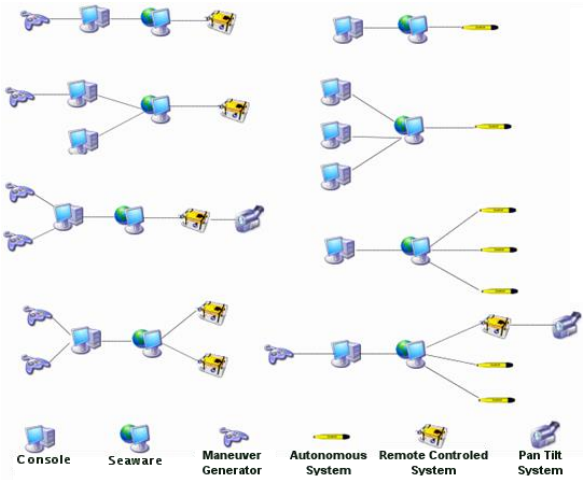

Fig. 4. Possible operation scenarios allowed by Neptus Console Builder

The architecture behind the configurable consoles is simple to use, not only for final users but also for developers. Neptus, as an application framework, can be used to create new visual components (besides the large amount of existing ones) to be added in operational consoles. New visual components must be able to implement some important interfaces, defined by Neptus. Besides variable tree communication interfaces there are other important interfaces to be considered. For instance, the visual components can be extended for scripting, which uses the variable tree repository information, by endowing a specific interface with scripting support. With this scripting interface, components can run JavaScript code that accesses the variable tree system. The script will run whenever some variable of the tree system it uses is updated. The "component developer" only has to process the script's return value and message. This kind of capabilities, easy to implement using Neptus framework, makes visual components extremely configurable to the final user.

Alarms~~,~~ in ~~–~~Neptus base console builder~~, its~~ are another ~~subject~~ topic that requires some special attention to simplify components implementation and final ~~user~~ usability. In summary, alarms work as a tree graph where the root node is the console itself. The alarms, messages and sates, generated by any alarm, travel the alarm graph structure with destiny to the root (console), being displayed in the state bar LED and error messages window (Fig. 5~~Fig. 5~~).

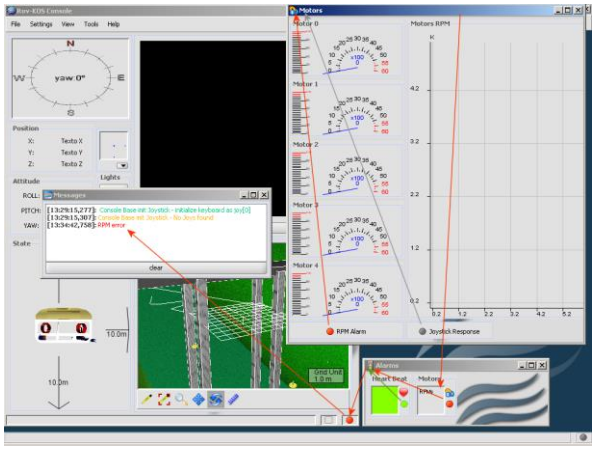

Fig. 5. RO~~V~~V~~–~~ ~~–~~KOS Console and the represented alarm graph.

All the alarm nodes automatically set their state by maximizing the levels of their children recursively. As a result, the root console alarm LED shows the major error occurred in some component (Fig. 6~~Fig. 6~~). The user can find the origin of alarm error by opening the alarms window and following the visual components that indicated failure.

~~Conjugating~~ By conjugating alarm and scripting interfaces we have a complete, flexible and organized way to treat malfunctions at mission execution time. Neptus Console Builder demonstrates to be complete enough for responding to the needs of existing systems in USTL and its abstract base architecture is prepared to be reused in consoles for other types of systems (vehicles).

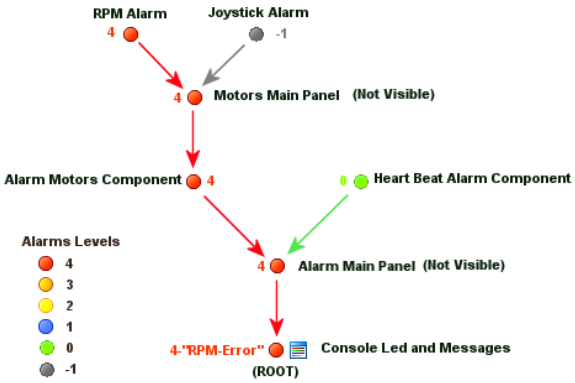

Fig. 6. One Console Alarm graph example with an RPM error message path

6. SYSTEM VALIDATION

The development of Neptus is being closely followed by real world tests. They were the drivinge force for the startup of the first Neptus prototype and continue to validate the recent developments. There were several tests made during the development of this framework but let us focus on several that serve as validation tests.

The first vehicle that used Neptus was the ROV-IES (Fig. 7~~Fig. 7~~). Here was the first debut of Neptus in which it was possible to draw a map, plan a mission

visually, generate the mission file and see its execution. The results were very encouraging because it passed the real-time test of ~~real time~~ following ~~of~~ a mission and contributed~~reduce~~ significantly to the reduction of ~~the~~ mission errors ~~in the mission~~. The mission was the inspection of an underwater pipeline in Douro River in P~~O~~porto. Because the water is not always very clear, a good visual aid is very important ~~because of~~due to the lack of visual landmarks.

The first tests in generation of mission files through XSLT were very good but the real test came with the Isurus AUV (Fig. 7~~Fig. 7~~). Here, because the mission file is a text file very closely resembling REMUS mission files, errors are very likely to happen, especially on site changes. With the visual maps and visual aids in the planning of mission, the errors ~~are~~ were reduced ~~to~~ almost to zero. The tests were made in a Portuguese Nautical Center in Montemor-O-Velho near Coimbra. There, the tests were so good that in fact planners stop editing mission files by hand at all. ~~One~~ Another improvement was the tracker console which allowed tracking the vehicle's movement in order to see if the plan is being followed without errors.

Then, we had the test of tests in the Portuguese Navy and NATO Exercise Swordfish in May 2006. We have simultaneously demonstrated Neptus, Isurus AUV, Roaz ASV (Fig. 7~~Fig. 7~~) and Wireless Sensor Networks, with data being collected and distributed to the various consoles and, at the same time, publishing live data to the Internet. Overall, the~~–~~ results were very positive.

Fig. 7. Isurus (top right), ROV-IES (top left) and Roaz vehicles

7. CONCLUSIONS AND FUTURE WORK

The Neptus framework has given various proofs of its practical utility and extensibility in the past. The modularity of its various components has greatly contributed for the success of this software infrastructure. The task of creating new specialized applications by simple reutilization of existing components is very appreciated by all developers.

The communications infrastructure that Neptus uses has also given proofs for its extensibility. Currently Neptus is being used to plan and control conjoint operations of ROVs, AUVs, ASVs, UAVs and WSNs. Despite ~~these systems~~ having different operating systems (QNX, Linux, TinyOS) and different communication means (Wi-Fi, wired Ethernet, acoustic modems, ZigBee …) all these systems~~, they~~ can ~~all~~ be connected to Neptus ant their data is transparently shared across the connected consoles.

The possibility to define an abstract mission and then translate the~~a~~ resulting XML by using XSLT is also a much appreciated feature because, without touching the Neptus code, support for different vehicles can be added. In the same manner, the ability to define operating consoles visually is of extreme importance to anyone trying to use Neptus to interact with a different vehicle in a possibly different operational scenario.

Currently, Neptus lacks the possibility to define logical conditions for the plans' state transitions. This functionality will be added, allowing operators to define transitions based on the environmental conditions (~~variables~~data ~~existent~~ available in the network). This will add extra flexibility like, for instance, the possibility to execute a maneuver only when a vehicle is nearby or when a sensor network reports a certain average value.
A simulation service is also being developed to support operator training and validation of mission specifications. Currently, the only supported vehicle is one of our ROVs (ROV-IES) but this service will be developed in the same fashion as Neptus, allowing the simple inclusion of different vehicles.

In the future, the data flowing in the network will also be logged to a central database. This database can then be accessed by the MRA application for mission revision or through a web page ~~that~~ displaying~~s~~ the data being gathered by any vehicle using Neptus in ~~some~~ anywhere ~~place of~~in the world.

ACKNOWLEDGMENTS

This research has been partly supported by AdI (Agência de Inovação) under projects PISCIS and KOS. Paulo Sousa Dias would like to thank the financial support of FCT (Fundação para a Ciência e Tecnologia) in his work.

REFERENCES

Dias, Paulo Sousa, R. Gomes, J. Pinto,~~,~~ S. L. Fraga, G. M. Gonçalves, J. B. Sousa and F. Lobo Pereira (2005), Neptus – A framework to support multiple vehicle operation. In: *Today's technology for a sustainable future, OCEANS Europe 2005*, Brest, France, June 20-23.

Dias, Paulo Sousa, R. Gomes, J. Pinto, G. M. Gonçalves, J. B. Sousa and F. Lobo Pereira (2006a), Mission Planning and Specification in the Neptus Framework. In: *Humanitarian*

Robotics, ICRA 2006 IEEE International Conference on Robotics and Automation, Orlando, Florida, USA, May 15-19.

Dias, Paulo Sousa, J. Pinto, G. M. Gonçalves, R. Gonçalves, J. B. Sousa and F. Lobo Pereira (2006b), Mission Review and Analysis. In: *Fusion 2006 The 9th International Conference on Information Fusion*, Florence, Italy, July 10-13.

Gerkey, B. P., R. T. Vaughan, K. Støy, A. Howard, G.S. Sukhatme, R. J. Matarić (2001). Most Valuable Player: A Robot Device Server for Distributed Control. In *Proceedings of the Second International Workshop on Infrastructure for Agent, MAS and scalable MAS*, Montreal, Canada, May 29

Hydroid Inc., <http://www.hydroidinc.com/> (Jul, 2006)

Marques, E.R.B., G.M. Gonçalves and J.B. Sousa (2006). Seaware: a publish/subscribe middleware for networked vehicle systems. To appear in: *7th Conference on Manoeuvring and Control of Marine Craft (MCMC'2006)*, Lisbon, Portugal, from September 20-22.

Neptus, <http://whale.fe.up.pt/neptus> (Jul, 2006)

Lee, C. S. (2004), NPS AUV Workbench: Collaborative Environment for Autonomous Underwater Vehicles (AUV) Mission Planning and 3D Visualization. *MSc Thesis*, Naval Postgraduate School, Monterey, U.S.A., March 2004

Sousa, J. B., F. Lobo Pereira, P. F. Souto, L. Madureira and E. P. Silva (2003). Distributed sensor and vehicle networked systems for environmental applications. In *Biologi Italiani*, n. 8, pp 57-60