NEPTUS – A FRAMEWORK TO SUPPORT THE 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: The *Neptus* distributed command and control framework for operations with vehicles, sensors, and human operators in inter-operated networks is presented. This is done in the context of applications, technologies, and field tests. There are applications for world representation and modeling, mission planning, simulation, execution control and supervision, and post-mission analysis. This is done in a mixed initiative fashion allowing the intervention by experienced human operators. XML abstract data types and XSLT technologies facilitate vehicle-interoperability and the standardization of interactions. A publish/subscribe (P/S) middleware framework for communications in a distributed environment enables the transparent inter-operability of communication networks. A console builder together with the P/S middleware allow the user to configure operating consoles for different vehicles. Results from field tests validate the overall framework and provide directions for future work.

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

1. INTRODUCTION

Researchers and technology developers are devoting significant efforts to the development of concepts of operation for networked vehicle systems. In these systems vehicles come and go and interact through inter-operated networks with other vehicles and human operators. Surprisingly, or not, the role of human operators is receiving significant attention in new concepts of operation for future systems. In fact, this is the reason why researchers and technology developers have introduced the concept of mixed initiative interactions where planning procedures and execution control must allow intervention by experienced human operators. In part this is because essential experience and operational insight of these operators cannot be reflected in mathematical models, so the operators must approve or modify the plan and the execution. Also, it is impossible to design (say) computer controllers that can respond satisfactorily to every possible contingency. In unforeseen situations, these controllers ask the human operators for direction.

The design and deployment of mixed initiative frameworks in a systematic manner and within an appropriate scientific framework requires a significant expansion of the basic tool sets from different areas (computation, control, communication, and human factors) and the introduction of fundamentally new techniques that extend and complement the existing state of the art. The major challenges come from the distributed nature of these frameworks and from the human factors. This is why we need to couple the development of scientific frameworks with field tests with human operators.

At the Underwater Systems and Technology Laboratory (USTL) from Porto University we have been designing and building ocean and air going autonomous and remotely operated vehicles with the goal of deploying networked vehicle systems for oceanographic and environmental applications (Sousa *et al.*, 2003). We have developed a framework for the mixed initiative coordination and control of networked vehicle systems and a tool set for deploying applications. This is done in the framework of dynamic networks of hybrid automata. The concepts for execution control build on experience in the modular design of distributed control hierarchies described in (Sousa *et al.,* 2003). The tool set comprises the *Neptus* command and control framework (Dias *et al.,* 2005; Neptus, 2006) and the *Seaware* middleware publish/subscribe framework for distributed real-time systems (Marques *et al.,* 2006).

Here, we discuss mixed initiative interactions in the context of the *Neptus* framework and report on field tests with network vehicle systems. The *Neptus* design facilitates mixed initiative interactions with heterogeneous vehicle systems over inter-operated networks. First, *Neptus* applications are built around a set of truly reusable software modules with special emphasis on modules for Graphical User Interfaces (GUI) and data management. Second, *Neptus* embodies the abstractions of our command and control framework with XML abstract data types and eXtensible Stylesheet Language Transformations (XSLT) technology; this leads to vehicle-interoperability and to the standardization of interactions with human operators. Third, *Neptus* allows the user to configure operating consoles for different vehicles. Fourth, *Neptus* uses the *Seaware* middleware framework for communications in a distributed environment; this enables the transparent inter-operability of communication networks.

There are not many references on software frameworks for mixed initiative interactions with networked vehicles. Previous work has focused on operational consoles for robotic systems. There are several examples of these of applications; see (Hydroid Inc., 2006) for details on the Remus autonomous underwater vehicle (AUV) console. The Naval Postgraduate School AUV Workbench (Lee, 2004), is capable of managing various vehicles in a cooperative manner. The PLAYER project (Gerkey *et al.*, 2001) developed a communication infrastructure for robotic operations. There is a server relaying data from existing systems (robots, operators, sensing devices …), thus enabling 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.

This paper is organized as follows. In section 2 we present the main concepts behind our control and coordination framework for multi-vehicle systems. We present the *Neptus* framework in section 3 and describe the communications infrastructure in section 4. In section 5 we discuss the configuration of consoles for operations. In section 6, we discuss field tests with multi-vehicle systems. Finally, in section 7, we present the conclusions and discuss future developments.

2. CONTROL FRAMEWORK

We use the concept of maneuver – a prototype of an action/motion description for a single vehicle – as the atomic component of all execution concepts. Thus we abstract each vehicle as a provider of maneuvers and services. A simple protocol based on an abstract vehicle interface governs the interactions between the vehicle and an external controller: the external controller sends a maneuver command to the vehicle; the vehicle either accepts the command and executes the maneuver, or does not accept the command and sends an error message to the controller; the vehicle sends a *done* message or an error message to the controller depending on whether the maneuver terminates successfully or fails.

The abstract vehicle interface is targeted at operations in multi-vehicle control and coordination frameworks. It enables us to decouple the details of vehicle control from the way we organize the external controllers. These are organized in a graph with a tree structure: the nodes are the controllers and the edges are the links connecting them. There are four layers in this tree structure, one layer for each type of controller – task, sub-task, sub-team and vehicle respectively. The root node is the task controller. It is linked to the sub-task controllers. Each sub-task controller is linked to the sub-team controllers. Each sub-team controller is linked to the vehicle controllers in the sub-team. This depends on the task specification. Controllers come and go, but the tree structure is kept. This organization follows from the structure of task decomposition and vehicle allocations. We model this control structure in the framework of dynamic networks of hybrid automata (Sousa *et al.*, 2004).

The design of our control structure allows for mixed initiative interactions at all layers in the hierarchy. This is done at the controller level. The transition structure of the automaton model specifies the conditions under which the operator is invoked, or allowed, and the states where the controller waits for the intervention of the operator. The abstract vehicle interface allows operators to directly interact with each vehicle. This is a first step towards automation since human operators are able to play the role of the external controllers with the help of the network by conforming to the interaction protocols.

3. NEPTUS OVERVIEW

Neptus is a distributed command and control framework for operations with networked vehicles, sensors, and human operators. The interactions with human operators are classified according to the phases of a mission life cycle: world representation; planning; simulation; execution and post-mission analysis. There are applications for world representation and modeling, planning, simulation, execution control, and post-mission analysis.

Neptus uses the *Seaware* middleware framework for network communication (Marques *et al.,* 2006). *Seaware* is a publish/subscribe framework for dynamic and heterogeneous network environments oriented to data-centric network computation. Publishers and subscribers communicate transparently to any node that is registered in the network. Nodes can either be vehicles that publish sensor data and receive operator commands or consoles that subscribe to the data provided by vehicles and sensors and publish operator commands. Seaware uses the RTPS (Real Time Publish Subscribe) protocol and other forms of network transport.

We have adopted XML for data representation in *Neptus*. This enables us to define a grammar for every data file and to specify the exact file format to be expected from potential users. XML can also be filtered and transformed into different formats like text, HTML or any kind of native mission file formats for existing vehicles. A eXtensible Stylesheet Language Transformations (XSLT) stylesheet gives the transformation rules from XML to the vehicle's mission language. This facilitates vehicle inter-operability and the integration of new vehicles. When we add a new vehicle to *Neptus* we must specify the vehicle's command interface in XML format.

There is a set of modular software components – Map Editor, Mission Planner, Mission Processor, Console Builder, Variable Tree, Renderer2D, Renderer3D – which can be used by developers to build *Neptus* applications. This is especially useful when it comes to integrate new vehicles in the framework. The *Neptus* software components and interactions are briefly described next.

The Mission Map Editor (MME) component is a GIS-like application that allows the creation and manipulation of three dimensional world maps. Maps are stored as XML files.

The Mission Planner (MP) component is a top-level application for single and multi-vehicle mission planning. Mission planning is vehicle specific. There is a library of vehicle models and interfaces. Mission plans are stored as XML files. A mission plan is composed of world maps (links to other XML files), vehicle mission plans (a graph with nodes representing maneuvers and transition conditions among them) and additional data like local information, checklists for operations, and specifications for tests.

The Mission Processor (MProc) component translates *Neptus* mission files (XML) to the native formats used by different vehicles. We use this module to generate vehicle-specific mission files. These are then uploaded to a vehicle for execution.

There are vehicle-specific and mission-specific operational consoles. We use the first to supervise single vehicle operations and the latter to supervise multi-vehicle operations. We use the Console Builder (CB) component to build operational consoles and to tailor these consoles to each vehicle and to each operator. Initially the CB application presents an empty window which serves as a canvas for adding various visual components. The visual components are then connected to variables that might be available on the network. These include, for example, the state of the vehicle, or the motor RPMs. The configurations for each console are saved as XML files for reuse.

There is a Variable Tree (VT) module in every console. This module stores the incoming network data and provides generic access to data values. The variables are stored in a tree structure. We use this tree structure to trigger typed events and the updates of dependant variables when the value of a given variable is updated. This simple scheme allows the easy specification of system alerts by defining scripts that run whenever a variable or a variable domain is updated.

The two dimensional (R2D) and three dimensional Renderer (R3D) components are used to visualize the motions of the vehicles and the state of the world. These can be used simultaneously. The Renderer components are connected to VT module in each console to subscribe to the data for visualization. The R3D version proved extremely useful to support the human operator in remotely operated vehicle (ROV) operations. This is because video from the vehicle does not provide enough visual clues for tele-operation in low-visibility areas. The R2D module is quite useful to supervise operations that take place over a large area. Additionally, R2D is also used for map edition, allowing the user to interact with the existing objects (images, paths, marks, etc.).

The Mission Review and Analysis (MRA) component provides support for the analysis of mission data. This includes provisions for replaying missions in a virtual world and also to graph mission variables.

Together, these modules enable the specification of abstract missions with coordinated vehicle plans and world maps (Dias *et al.,* 2006a, b).

The *Neptus* design supports concurrent operations. Vehicles, operators, and operator consoles come and go. Operators are able to plan and supervise missions concurrently. Additional consoles can be built and installed on the fly to display mission related data over a network. Fig. 1 depicts multi-vehicle interactions under *Neptus* and *Seaware*. There is one operational console for an autonomous surface vehicle (ASV) and another one for a remotely operated vehicle (ROV).

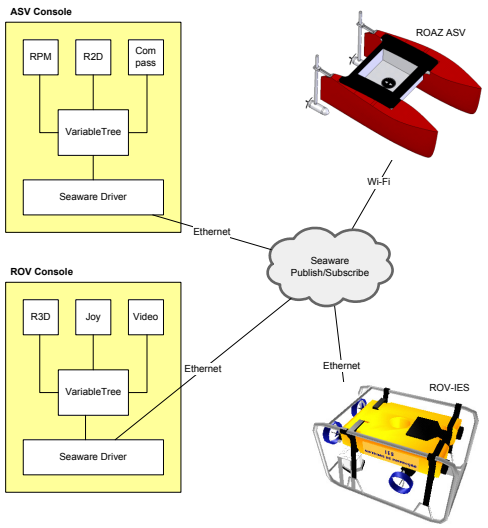

Fig. 1. Interactions under *Neptus*

4. COMMUNICATION INFRASTRUCTURE

We have used XML as the base for all data transfers, storage and manipulation in Neptus. We use XML for the messaging services between Neptus components and external ones. There is a base set which is easily extended.

In order to add a new type of message to the *Neptus* framework we just need to add a few lines to a XML file. That file contains the definition of all messages that will be able to be read by Neptus. There is a fixed message header; the body can be composed of several fields from the set of provided types (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, plaintext (first 2 bytes indicates the length)	minimum 2 and maximum $2 + 2^{16}$ (65537)

Fig. 2 depicts 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 other fields. The Id must be unique for all the components using the *Neptus* messaging service.

```
<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

Seaware 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 two sets of published topics and of subscribed topics which correspond to the message exchanges required for vehicle control.

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 generalizations to many-to-many (consoles/vehicles) communication.

5. CONFIGURABLE CONSOLES

We have developed the Console Builder (CB) application to facilitate the addition of new vehicles with new sensor suites to *Neptus*. The operator uses CB to build and configure vehicle consoles.

There are two important aspects to console configuration: visual components and event communications.

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

There are two states in the Neptus generic console builder application: Editing and Operational. 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. When all components are ready, correctly placed and connected to the system variable tree, the user can switch the state of the application to the operational mode where the components become fixed in the console and start to respond to the user interactions (mouse clicks, key presses …).

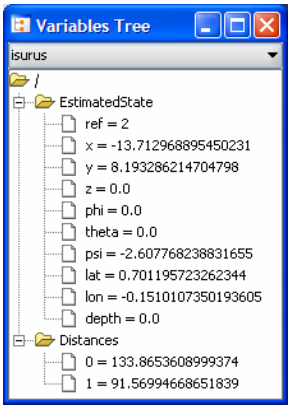

Fig. 3. Neptus variable tree for the Isurus AUV

Currently, there is one specialized console for every vehicle operated under *Neptus*. But we can do more. We can use CB to build multi-vehicle operational consoles for a variety of operational scenarios, even to supervise and control several systems simultaneously, as shown in Fig. 4.

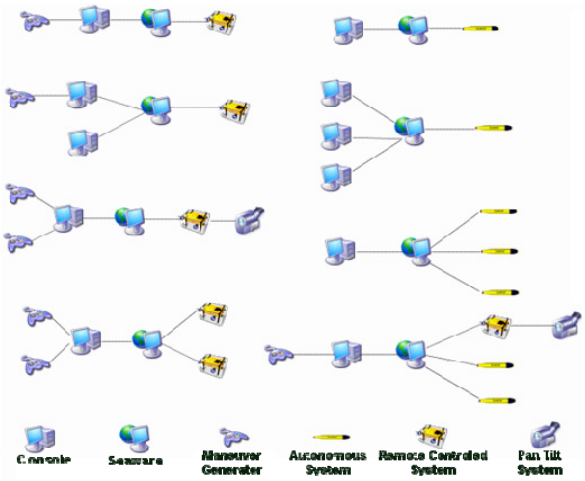

Fig. 4. Operation scenarios allowed by Neptus Console Builder

The architecture behind 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 to be added to operational consoles. The visual components can be extended for scripting, which uses the variable tree repository information. This endows a specific interface with scripting support. With this scripting interface, components can run JavaScript code that accesses the system's variable tree. 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 the *Neptus* framework, makes visual components extremely configurable to the final user.

Another important subject that is established in CB, for easy reconfiguration, is the alarms treatment. Alarm components are hierarchically connected, resulting in a tree structure where the root node is the console itself. The alarm messages travel up the alarm graph structure to the root. Alarms are classified into several levels. A graphical representation of the alarms is displayed on the operational consoles together with a window for error messages (see the LED bar and window for error messages window in Fig. 5). 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).

The use of alarms and scripting interfaces leads to a flexible and organized way to handle malfunctions at mission execution time.

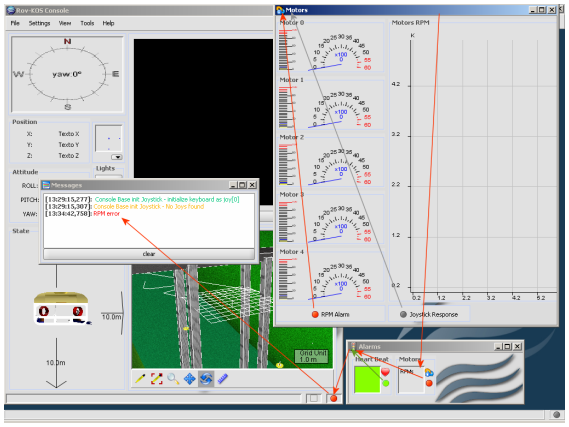

Fig. 5. KOS ROV console and the graph of alarms.

We have used the *Neptus* Console Builder extensively to configure operational consoles for ocean and air going vehicles. CB enables us to configure a new console rapidly and with all the required operational ingredients.

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

6. FIELD TESTS

Field tests are essential to validate developments and to evaluate mixed initiative interactions. We have field-tested *Neptus* in our operational deployments.

We started field tests with single vehicle operations and recently moved into multi-vehicle operations with wireless sensor networks (WSN). In the first field test we used Neptus for mission planning and control of the IES ROV in the inspection of an underwater pipeline (Fig. 7). The use of the same map for mission planning and execution greatly reduced the number of human errors. We were able to visualize the mission in simulation and to use the experience acquired in simulation to operate the vehicle in real time. The 3D visualization of the real motions in a virtual world proved quite useful for operations in waters with poor visibility. We tested the mission planning GUI and the generation of mission files through XSLT in operations with the Isurus AUV (Fig. 7) which took place in the Montemor-O-Velho nautical center. This represented a great advance since we used to edit Isurus native mission files by hand. The number of planning errors was greatly reduced with the help of

the 2D/3D maps and of the visual aids of our planning GUI. We have also built a new console to track the motions of Isurus with the help of data provided by the acoustic localization system. This console enabled us to evaluate mission performance in real-time. We had to provide consistency checks for displaying data coming from different sources.

We used Neptus to operate two Wireless Sensor Networks and two vehicles (Isurus and Roaz) in the NATO Swordfish exercise which took place in May 2006 in Tróia (Portugal). This was done in cooperation with the Portuguese Navy. There was one operator per vehicle and multiple consoles to subscribe to the data published by the vehicles and the sensors. Data was published live to the Internet.

Fig. 7. Isurus AUV (top right), IES ROV (top left) and Roaz ASV

7. CONCLUSIONS AND FUTURE WORK

The *Neptus* framework has already proven invaluable in operational deployments with ROVs, AUVs, ASVs, and WSNs running different operating systems and using inter-operated communication networks (Wi-Fi, wired Ethernet, acoustic modems, ZigBee, etc.). This is in part because of its modular design and of the underlying communications infrastructure. The ability to create new specialized applications through the reutilization of existing components is very appreciated by developers. Heterogeneous vehicles and sensors are easily integrated into the *Neptus* framework and data is transparently shared across operational consoles. The ability to define an abstract mission and to translate the resulting XML by using XSLT is also a much appreciated feature because it allows the integration of new vehicles without changes to the *Neptus* code. In the same manner, the ability to build operating consoles with a GUI is quite important for anyone trying to use *Neptus* to interact with a new vehicle in a new operational scenario.

Neptus is a work in progress. New releases incorporate lessons learned from operational deployments. The available functionality is being extended and improved. This includes: a simulation service to support operator training and validation of mission specifications for generic vehicles (currently this is restricted to one ROV); GUI for mission specification in the framework of hybrid automata (currently mission plans have a linear structure); data logging onto a central database which will be accessed by the MRA application for mission revision or through a web page. This will allow to display data gathered anywhere in the world by any vehicle connected to *Neptus*.

ACKNOWLEDGMENTS

This research was partly supported by Agência de Inovação under project PISCIS. Paulo Dias would like to thank the financial support of Fundação para a Ciência e Tecnologia.

REFERENCES

Dias, P. S., 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, P. S., 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, P. S., 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/>

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.

Sousa, J. B., T. Simsek and P. Varaiya (2004). Task planning and execution for UAV teams. In Proceedings of the Decision and Control Conference, Bahamas.