Mission Planning and Specification in the Neptus Framework

Paulo Sousa Dias, Rui M. F. Gomes, José Pinto
Gil M. Gonçalves, João Borges Sousa, Fernando Lobo Pereira

LSTS – Underwater Systems and Technology Laboratory
Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias s/n
4200-465 Porto, Portugal
{pdias,rgomes,zepinto,gil,jtasso,flp}@fe.up.pt

***Abstract* - The C3I (Command, Control, Communication and Information) Neptus framework which is being developed at the Underwater Systems and Technology Laboratory (USTL/LSTS) is presented. Neptus is a modular mixed initiative framework (human operators in the control loop) for the operation of heterogeneous teams of vehicles such as autonomous and remotely operated underwater, surface, land, and air vehicles. Neptus is composed of mission and vehicle planning, supervision, and post-mission analysis modules which are provided as services across a network. This paper focus mainly on the mission definition module with the presentation of MDL – a XML based language for mission definition.**

Index Terms* - *Control Networks, Middleware, Systems Engineering, Systems networks, Underwater Vehicles.

I. Introduction

This paper presents the Neptus C3I (Command, Control, Communication and Information) infrastructure for the coordination and control of teams of multiple autonomous and semi-autonomous vehicles. Neptus is a mixed initiative environment (operators in the control loop) which is being developed to support operational deployments of autonomous and remotely operated vehicles from the Underwater Systems and Technology Laboratory (USTL/LSTS) from Porto University. Operational deployments involve a wide variety of interactions between operators (human or automated) and vehicles which include: mission setup and vehicle preparation (extensions for multi-vehicle operations are under development); real-time data acquisition and visualization; operator supervision; coordinated control of multiple vehicles (fleet control); and post-mission review and data analysis.

The USTL is currently operating three types of ocean going vehicles: ROV (Remotely Operated Vehicle), ASV (Autonomous Surface Vehicle) and AUV (Autonomous Underwater Vehicle). These capabilities will be extended in 2006 with the addition of UAVs (Unmanned Air Vehicle). The ROVs are being developed under the KOS (Kit of underwater operations) project [1] and the AUV and the ASVs are being developed under the PISCIS (Prototype of an Integrated System for Coastal waters Intensive Sampling) project. These joint operations, with heterogeneous multiple vehicles, and the need to control the complete fleet in a coordinate manner set the motivation and the requirements for the development of Neptus.

The development of the Neptus framework followed the Systems Engineering Process [2]. In this process, each stage of the systems life cycle is divided in three activities: Requirements analysis; Functional analysis; and Synthesis and design. System life cycle stages range from "System definition" to "Costumer support", including "Subsystem definition" and "Production". The Neptus framework is currently in the final phase of the "Subsystem definition" stage: the subsystems have been integrated and system wide tests are being performed. Neptus has been deployed along with ROVs, ASVs and AUVs in operations which took place in 2005 in Portugal.

This paper is organized as follows. Section II presents the motivation and state of the art of existing tools. The requirements that guided the development of the Neptus infrastructure, the definition of subsystems, components and architecture are presented in section III. Section IV describes the architecture and the common operational scenarios of the tool. Section V presents the Mission Planner application focusing on the mission definition. Section VI illustrates the use of Neptus in an operational deployment with the ISURUS AUV. Finally, in section VII, conclusions and an outline of future work are presented.

II. Motivation and State of the Art

Currently there are multiple software applications which support the operation of autonomous and remotely operated vehicles operations. Most of these applications are basically operational consoles. Consoles tend to be designed and developed as new requirements arise. Most of these consoles lack the modularity needed to extend its functionalities to other vehicles and types of operations. Examples include the REMUS AUV [3] console. The main objective of this application is to display the REMUS AUV vehicle trajectories by means of an animated replay. During replay, data gathered is shown in the form of 2D graphical plots. There is also the possibility to define new missions through the specification of waypoints. However, little support is given for communications with other devices like computers, databases or other vehicles and there is no possibility to simulate future missions. Another example for this kind of applications is the ROV hardware based console from Deep Ocean Engineering [4]. This console has no visual interface what makes it

difficult to integrate the ROV mission with other vehicles or consoles in a cooperative manner.

In the early days of the USTL laboratory we operated one Phantom class ROV vehicle and there was no need for a modular console. Currently, we are working with different types of vehicles which are designed for cooperative missions in a mixed initiative environment where vehicles and systems come and go. This requires a different approach to the development of consoles. In this approach we have designed and developed a modular framework which can be configured for different vehicles and interactions.

One case where this level of integration is being done is at the Naval Postgraduate School (NPS) with the AUV Workbench [5]. AUV Workbench allows the visualization of the vehicle's behavior by means of simulation of the AUV physical equations. It is possible to view the vehicle's behavior in 2D or 3D recurring to VRML technology, being easy to do revision and rehearsal of AUV missions. This application proves to be very useful for control algorithm development and testing because of its physical model implementation. The user also has the possibility to do mission planning and save this plan in the form of a XML (eXtensible Markup Language) [6]. Then by means of eXtensible Stylesheet Language Transformation (XSLT) the mission can be translated to several different types of vehicles. This workbench also provides reliable data transfer between AUVs, other vehicles, server agents and human controllers and automatically logs all communications what facilitates data retrieval for post-mission-analysis and mission reconstruction.

Although the AUV workbench was designed for use with multiple vehicles it lacks distributed hybrid systems control concepts which facilitate the development and formal verification of cooperative missions. On-going projects at USTL require vehicle interactions in the context of dynamic networks of hybrid systems [7].

Neptus is also being integrated with sensor networks in such a way that missions can be re-planned in real time based on the information gathered by these networks.

III. The Requirements

This section describes system requirements for the Neptus systems from the user's point of view. The exposition is partially based on IEEE Std. 1362-1998 [8], which is a standard for system characteristics description based on the definition of Concept of Operations (ConOps).

The execution of operational missions with the USTL underwater vehicles is the main motivation for the Neptus framework. Missions can be performed with either a single vehicle or with a set of vehicles, depending on the mission objectives. The vehicles can be of various types, including AUVs, ROVs, UAVs and ASVs. The mission execution with these classes of vehicles requires four main steps ([9] and [10]): operational setup, mission programming, mission execution and mission analysis.

The *operational setup* phase deals with the reconnaissance of the operational site and identifies the mission objectives. In the *mission programming* phase, the path (or area of operation) of the vehicle(s) is defined and a mission is specified by selecting a pre-defined set of maneuvers and tasks. In the *mission execution* phase there are several types of interactions depending on range and communication bandwidth. Typically communications with underwater vehicles are either slow (through acoustic modems) or non-existent, This may also be the case with other vehicles when operating out of communication range. In this case the data gathered by these vehicles can only be seen in the mission analysis phase. In the case of ROVs, since there is always a connection between the operator and the on-board computer of the ROV, the user may operate the vehicle through a joystick and has continuous real-time access to the data acquired by its sensors [11]. The *mission analysis* phase is the last of the mission phases and concerns the post-mission analysis of an operational deployment.

The Neptus framework [12] was designed to fulfill the operational mission requirements described above. Thus, the top level requirements are listed below:

- An application to define the environment of the operational mission. This includes navigational references, bottom profiles, obstacles, and landmarks;
- An application for the mission programming;
- A console to establish a link between the support and the on-board computers. In the ROV class, the console should enable users to send joystick commands and visualize data in real-time;
- A simulation platform to allow the user to verify the conformity of the mission program. This tool will give the user the chance to debug its own mission plan;
- A tool for mission review and analysis; and
- A repository for the gathered data with associated querying services.

Neptus is intended to be used in several scenarios [10] with different types of vehicles. Thus, the framework has to be designed in a way that some portion of the framework (modules) may be used separately from the other modules. These modules must be designed to be easily integrated in the already existing software.

IV. Architecture

This application is organized into several applications/modules that, together, compose the Neptus environment. The deployment diagram depicted in Fig. 1, presents the five main applications and connections.

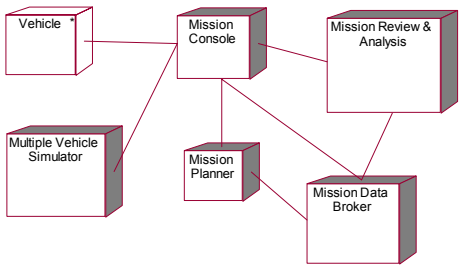

Fig. 1. Deployment diagram

The five main applications/modules are: Mission Planner (MP), Mission Console (MC), Mission Reviewer & Analysis (MRA), Mission Data Broker (MDB) and the Multiple Vehicle Simulator (MVS).

The Mission Planner (MP) application (see section VI) is intended to be used in the mission preparation and setup. This activity includes the generation/edition of missions, and map generation with the ability to prepare visual aids of the mission site and some additional; minor but helpful; functionalities. For this, it is necessary to define a language/syntax to describe the mission, the allocated vehicles and the individual and coordinated mission plans. For this purpose, we use XML (eXtensible Markup Language) [6]. The output of the MP will then be used as an input to the Mission Console (MC).

The Mission Console (MC) is the application in charge of the mission execution (partially presented in Fig. 2). As stated above, the inputs of this application are created in the MP. This application must also provide all the required functionalities to control and operate the vehicles, such as, visualization of the vehicle(s) state, and interfaces to send commands to the vehicle(s). In some circumstances, such as the operation of an ROV, the motion commands may be sent through a joystick. The vehicles may also be controlled by a hybrid automaton controller residing in this application. Naturally, several sub-modules of this application will run inside the vehicles being operated. This application will also be in charge of the translation of the Neptus mission language to the vehicles language.

The post-mission analysis is supported by the Mission Reviewer & Analysis (MRA) application (section VI). This will deal with the compilation and treatment of the collected data. It will also provide the support to replay the mission under analysis.

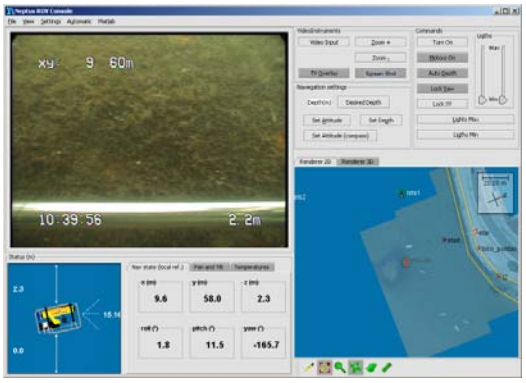

Fig. 2. ROV Mission Console

The central repository where all the gathered data can be stored, properly organized, controlled and published is the Mission Data Broker (MDB). This module/application will be based on a SOA (Service Oriented Architecture) with connection with other application/modules supported by web services. This application is not yet built but will allow the future web based access to the data collected in the missions. For this purpose several services have to be available. One of them will be the authentication and authorization service. This will control who is able to access to the data and to what data. Another service will be the mission requester that will allow people to request some service from our vehicles. Other services are being considered.

The Multiple Vehicle Simulator (MVS) will provide the functionalities needed to pre-execute a mission and assess its viability. This application will provide the service of several simulated vehicles that can be used with the MC or with the MP for a more accurately mission preview. At a moment only a basic simulator is implemented and connects to Neptus with the same interfaces as the actual vehicle. One future evolution will be the support of hardware in the loop simulation.

All the data that is produced in one of the described applications is then consumed by another one in a different stage of a mission. For the data representation XML [6] was chosen.

V. The Mission Planner

The Mission Planner (MP) is used to plan and evaluate a mission. Fig. 3, depicts the use case for the development of this module.

The MP is decomposed into several components: Map Editor Module (MEM); Mission Graph Editor Module (MGEM); Preview State Generator Module (PSGM) and World State Renderer Module (WSRM). Each component is also used in other Neptus modules.

The mission editor is the main component of the MP And serves as the mission graphical editor, defining all the mission elements (Fig. 4) described in [10].

Another important component is the Map Editor Module (MEM). Each map is basically a set of objects with attributes such as size, position or shape. The user is able to insert any number of objects on the map. Typical objects are marks, transponders, geometrical shapes (like parallelepipeds or ellipsoids), and free-hand and computer generated drawings (or paths) and images.

The Mission Graph Editor Module (MGEM) is used to create the plans which are modeled as hybrid automata. This editor allows the visual construction of automata in which each node is a maneuver and the transitions describe the maneuver logic.

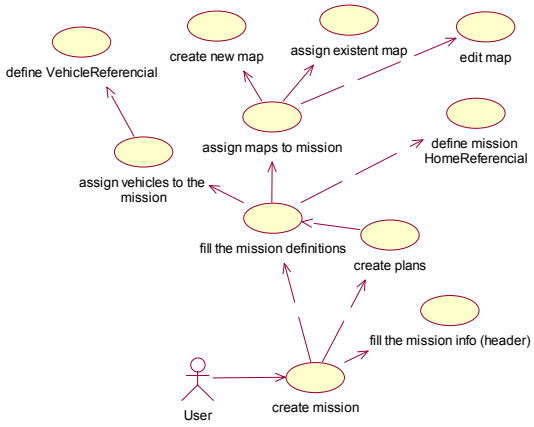

Fig. 3. Create mission use case diagram

The Preview State Generator Module (PSGM) is used in conjunction with the World State Renderer Module (WSRM) to perform the animation of the mission plan. , This module enables the user to preview the mission before it is actually executed.

The WSRM module supports the visual representation of

the world; it is used in most of the main applications (MC, MP and MRA). There are several ways of displaying this: as text messages; as 2D; and as 3D visualizations.

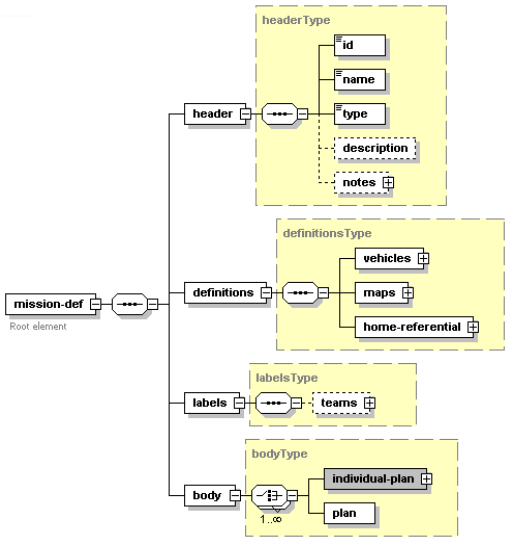

Fig. 4. Mission schema

A. Control Architcture

The mission plan is uploaded to each vehicle. The control architecture ensures that the plan is actually executed. The reference USTL control architecture [13] is depicted in Fig. 5.

At the bottom we have the low level controllers. This layer abstracts the interactions with the vehicle sensors and actuators in a modular interface.

The concept of maneuver plays a central role in the USTL control architecture: it facilitates the task of mission specification, since it is easily understood by a mission specialist; it is easily mapped onto self-contained controllers, since it encodes the control logic; and is a key element in modular design, since it defines clear interfaces to other control elements. Depending on the type of vehicle we can find maneuvers like: *Hover*, *FollowTrajectory*, *FollowWall*, *Surface*, *Goto*, *Rows*, *Tele-operation* and others. Let us take a closer look to the *Goto* maneuver. The maneuver is represented in a labeled transition system. Each transition is labeled with a guard, the condition under which the transition can take place, and an event, the message sent out when the transition is taken; the two are separated by a / in the figure.

The *Goto* maneuver starts in the Init state. If it receives the start event from the vehicle supervisor, it goes to the *O2Txy* state (move to the target on the *xy* plane). In this state the vehicle heading controller is switched on with a calculated heading reference. If the vehicle reaches the desired orientation within some bound, the maneuver controller goes to the next state *G2Txy* (go to target on the *xy* plane). In this control location the *xy* low level controller is switched on and the vehicle moves toward the final position. If it reaches the final position, the controller changes again to the *G2Tz* (go to target on *z*). If there were no problems in the previous three control locations, the maneuver is completed successfully. A timeout occurs if, for example, the vehicle gets stuck and execution does not advance. In this case the maneuver controller goes to the error state and an error code is sent to

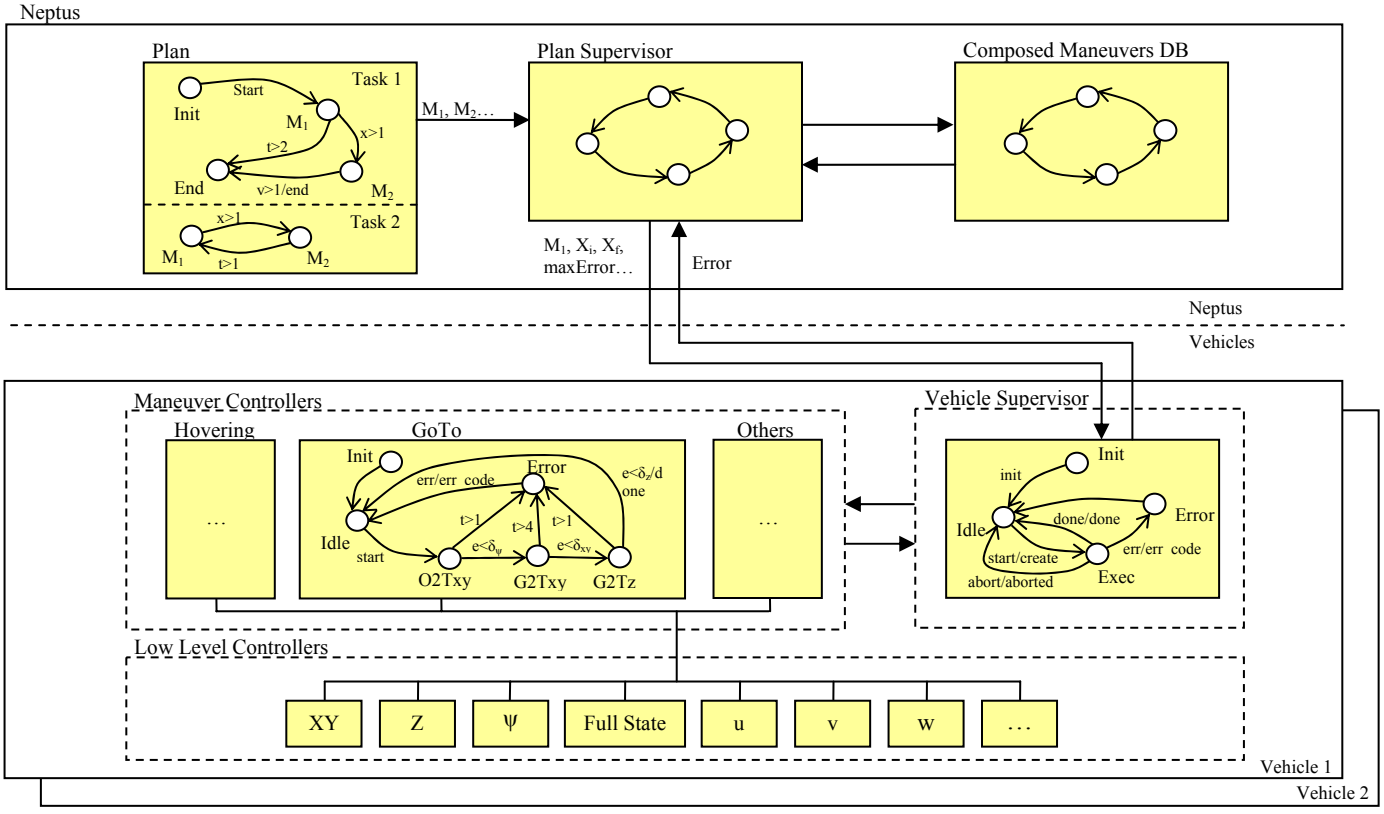

Fig. 5. Control Architecture

the vehicle supervisor.

Next, we have the vehicle supervisor which has 4 states: *Init*, *Exec*, *Error*, and *Idle*. The vehicle supervisor is initially in the state *Idle*. Upon the reception of a maneuver specification it creates a maneuver controller if the enabling condition is true. When the maneuver is completed it goes to the *Idle* state again, otherwise, the transition to the *Error* state is taken, and it sends an *err_code* event to the plan supervisor, and the plan fails.

On top of this architecture we have the plan and the plan Supervisor. The plan Supervisor commands and controls the execution of the mission plan. It commands the vehicle supervisor to trigger the execution of a maneuver specification and waits for the acknowledgment of its completion, or for an error. When it receives the acknowledgement, the plan supervisor selects the next maneuver to be executed. The process is repeated until the plan is successfully terminated, or it fails.

Neptus allows the definition of maneuvers other than the vehicle elemental maneuvers. These are libraries of composed maneuvers which can be re-used in other plans.

B. Mission Planning

In order to make the planning of a mission several tasks have to be engaged. As we saw in section III there are several things that have to be prepared in order to achieve a successful mission. From the reconnaissance of the mission site we will be able to construct a digital representation of the site. This will be the mission map. It will have information of objects like: marks, transponders, geometrical shapes and paths. All of these objects can be used as a reference when we are preparing the vehicles plan.

This data will be stored in XML format. The map information is part of the mission planning data. As seen in Fig. 4 the mission planning data contains the information required to guide the execution [10].

As mentioned before, a mission consists of various pieces of information. One of those pieces is an individual plan. An individual plan indicates which vehicle (or class of vehicles) can perform it and the sequence of maneuvers to be executed. This represents a series of maneuvers, commands and conditions of transition between them.

The most appealing way of representing this kind of information is a graph, so we created a component that allows the user to visually define an individual plan in the form of a graph, the Mission Graph Editor (MGE).

C. Mission Tasks

In order to define plans to a vehicle or a set of vehicles we need to define a series of tasks. These tasks can be either simple or they can be composed. Tasks basically involve the execution of maneuvers.

Maneuvers are of two kinds: basic and composed. It can be selectable from a series of pre-defined ones like the *Goto* maneuver. In terms of commands we can think of a *Reset Logger* or perform a *GPS Fix*.

Each type of maneuver (simple or composed) has a common structure: a body; a pre-execution area; an on--execution area; and a pos-execution area. The body is used to add the content of the task. The other three areas define the commands that are to be executed prior to the initiation of the task, during its execution and at the end of its execution. The control logic is encoded as predicates which state the conditions under which a transition occur in the maneuver automaton.

As a summary, we can show the current data structure for the plan:

```
Plan
    -   Task 1:
            o   Vehicle ID
            o   Maneuver1
                    ▪   Characteristics:
                            •   ID
                            •   Elemental Type (vehicle)
                            •   Composed Type (console)
                    ▪   Maneuver Parameters
                    ▪   Commands
                            •   onEntry
                            •   during
                            •   onExit
                    ▪   Next Maneuver
                    ▪   Condition
                    ▪
            o   Maneuver 2:
                    ▪
            o   Commands
                    ▪   onEntry
                    ▪   during
                    ▪   onExit
            o
    -   Task 2:
```

Because we want to allow some flexibility in the tasks definition it is provided, either by programmatic interfaces, or providing a language, the ability to create new tasks.

The mission data is stored in an XML document and complies with the rules defined in the mission schema.

VI. Operations

This session presents a use case of Neptus in a mission performed with the Isurus AUV in the Nautical Center in Montemor-o-Velho in Portugal (Fig. 6).

Fig. 6. Isurus AUV in Montemor-o-Velho Nautical Center, Portugal

Fig. 7. Preparation in MP of Montemor-o-Velho Nautical Center mission

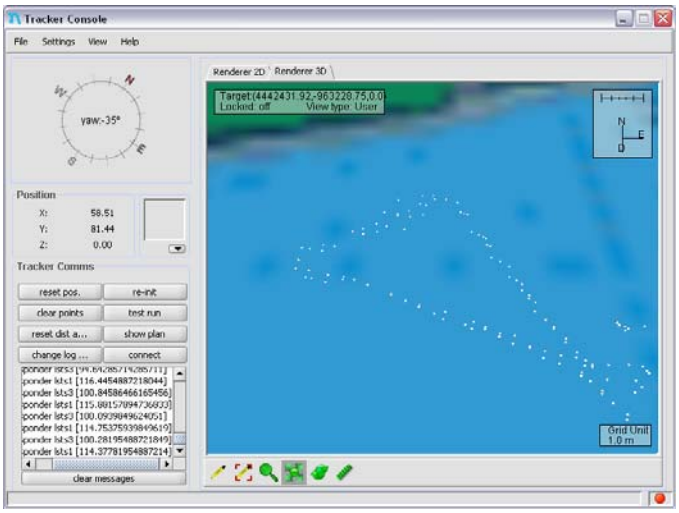

Fig. 8. Tracker Console

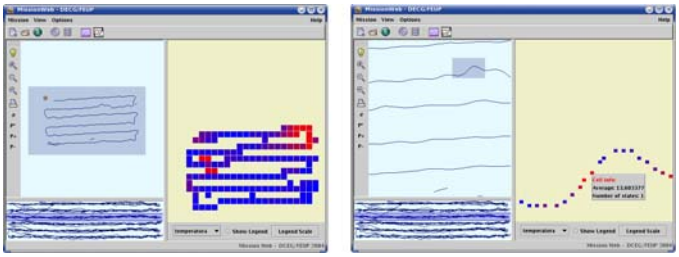

Fig. 9. Mission analysis with MRA

In Fig. 7 we see the MP application with the Nautical Center map and a row maneuver made out of several *Goto*'s. In Fig. 8 we see the MC for the tracker application where the vehicle's position in the mission execution can be estimated by acoustics. In Fig. 9 we see the MRA where the mission executed was reviewed and analyzed.

VII. Conclusions and Future Work

This paper presented the Neptus framework, a mixed-initiative environment to support the coordinated joint operation of multiple autonomous and semi-autonomous vehicles. This framework has been deployed in single vehicle operations. These range from AUVs, to ASVs, and ROVs. We are in the process of adding UAV models to the Neptus database.

The extensions to multiple vehicle operations have been tested in simulations and the first operational deployments will take place during the first semester of 2006. To do this we have used a Publish/Subscribe (P/S) middleware framework to distribute the framework across a local area network. Future work includes transfer of control authority among consoles and operators, and multi-vehicle planning and supervision.

Acknowledgments

The PISCIS project was funded by Agência de Inovação.

Paulo Sousa Dias and Rui Gomes would like to thank the financial support of FCT (Fundação para a Ciência e Tecnologia) in their work.

References

[1] Rui Gomes, A. Sousa, S. L. Fraga, A. Martins, J. Borges Sousa and F. Lobo Pereira, "A new rov design: issues on low drag and mechanical symmetry", Today's technology for a sustainable future, OCEANS Europe 2005, Brest, France, June 20-23, 2005.

[2] IEEE standard for application and management of the systems engineering process, IEEE Std 1220-1998, Vol., Iss., 22 Jan 1999.

[3] Hydroid Inc., <http://www.hydroidinc.com/> (Janeiro 2006).

[4] Deep Ocean Engineering, <http://www.deepocean.com> (Janeiro 2006).

[5] Naval Postgraduated School, <http://terra.cs.nps.navy.mil/AUV/workbench> (Janeiro 2006)

[6] XML – Extensible Markup Language 1.0 (Third Edition), W3C Recommendation 4th February 2004, <http://www.w3c.org/TR/2004/REC-xml-20040204/> (Janeiro 2006).

[7] Sérgio L. Fraga, João B. Sousa, A. Girard, A. Martins, "An automated maneuver control framework for a remotely operated vehicle", OCEANS, 2001. MTS/IEEE Conference and Exhibition, Vol.2, Iss., 2001 Pages: 1121-1128 vol.2.

[8] IEEE guide for information technology - system definition - Concept of Operations (ConOps) document, IEEE Std 1362-1998, Vol., Iss., 19 Mar 1998.

[9] P. Ramos, M. V. Nezves, N. Cruz, F. L. Pereira, "Outfall monitoring using autonomous underwater vehicles", International Conference MWWD 2000 – Marine Waste Water Discharges 2000, Génova, Italy, pp. 321-331.

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

[11] S. L. Fraga, J. B. Sousa and F. L. Pereira, "User-assisted trajectory generation for autonomous underwater vehicles", Marine Technology and Ocean Science Conf. – OCEANS 2003, San Diego, CA, USA, September 22-26, 2003, pp. 1234-1239.

[12] Neptus, <http://whale.fe.up.pt/neptus> (Janeiro 2006)

[13] Márcio Correia, Paulo Dias, Sérgio Fraga, Rui Gomes, Rui Gonçalves, Luís Madureira, Fernando Lobo Pereira, Rui Picas, José Pinto, António Santos, Alexandre Sousa, João Borges de Sousa, "Operations And Control Of Unmanned Underwater Vehicles", Robótica 2005, Coimbra, Portugal, April 29-1, 2005.