Neptus – A Framework to Support Multiple Vehicle Operation

Paulo Sousa Dias
Rui M. F. Gomes
José Pinto
Sérgio Loureiro Fraga
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,slfraga,gil,jtasso,flp}@fe.up.pt

***Abstract* - This paper describes the development of a C3I (Communications, Command, Control and Intelligence/Information) infrastructure, taking place at the Underwater Systems and Technology Laboratory (LSTS) of FEUP. This infrastructure, the Neptus framework, goal is to support the coordinated operation of heterogeneous teams, which include autonomous and remotely operated underwater, surface, land, and air vehicles and people. People perform a fundamental role, not only in the case of remotely operated vehicles, but also with autonomous vehicles where mix-initiative operation is a requirement. The operational scenarios for these teams are mainly environmental monitoring missions but could also include environmental disaster scenarios, rescue missions, etc. The Neptus distributed architecture is service oriented, which enables high degrees of interoperability between applications, of scalability (number of nodes), and of reconfiguration (number and type of nodes).**

Keywords: **Control Networks, Middleware, Systems analysis and models development, Systems Engineering, Systems networks, Underwater Vehicles.**

I. INTRODUCTION

This paper presents a mixed-initiative environment for the coordination and control of teams of multiple autonomous and semi-autonomous vehicles. This infrastructure (C3I – Command, Control, Communication and Information) is used in the context of the activities of the Underwater Systems and Technology Laboratory (LSTS/USTL) to support the joint operation of multiple autonomous and semi-autonomous vehicles. In the context of this work, operation means the wide variety of possible interactions between the pilot (human or automated) and the vehicles including: pre-mission setup and preparation of a vehicle (or multiple vehicles) mission; real-time data acquisition and visualization; pilot intervention during the mission execution (mixed initiative operation); coordinated control of multiple vehicles (fleet control); and post-mission review and data analysis.

The LSTS is currently performing operations with two vehicles from the laboratory, one underwater ROV (Remotely Operated Vehicle) and one AUV (Autonomous Underwater Vehicle). In the short run, within the framework of a collaborative research project, joint operations will include an UAV (Unmanned Air Vehicle) from the Academy of Portuguese Air Force. There are also two ongoing projects (KOS – Kits of underwater operations and PISCIS – Prototype of an Integrated System for Coastal waters Intensive Sampling) whose main targets are the design and construction of one ROV [1] and two AUVs, respectively. This leads to a fleet of 6 vehicles that have to be operated simultaneously. 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 a framework to support the coordinated operation of multiple vehicle.

The Systems Engineering Process [2] was used to guide the development of this infrastructure. 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 infrastructure is currently in the final phase of the "Subsystem definition" stage, where all subsystem are integrated and system wide tests are performed.

This paper is organized as follows. Section II presents the motivation and state of the art of the 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 prototype implementation of the framework. Finally, in section VI, conclusions and an outline of future work are presented.

II. MOTIVATION AND STATE OF THE ART

Typically there are many different software applications to support vehicles operations. Those applications take the form of vehicle consoles. Consoles are designed and upgraded when new small requirements arises. Normally these applications are not designed in an integrated manner. There are several examples of these types of applications, one of which is the Remus AUV [3] console. Its main objective is the display of vehicle trajectories by means of a replay. During the replay, data gathered is shown on a 2D graphics. There is also the ability of defining new missions. No support is given in communications with other devices like computers, databases or even other vehicles. It also lacks monitoring/controlling services and mission simulations based on the vehicles models.

Another example is the ROV hardware based console from Deep Ocean Engineering [4]. This type of console without any kind of interface makes it difficult to integrate the ROV mission with other vehicles or consoles in a cooperative manner. USTL started to work with ROVs with a Phantom class vehicle and fully develop its hardware from

scratch. This major modification was motivated to achieve a way to interface with the vehicle. The developed software had the form of a simple operating console. Today we have a main software console for each vehicle. In the near future we will have single console generating mission files and viewing data for every operated vehicle.

One case where this level of integration has been done is at Naval Postgraduate School (NPS) with the AUV Workbench [5]. AUV Workbench allows the visualization of vehicle behavior by means of simulation of AUV physical equations. It also helps the user to see collected data during the mission. Vehicle behavior visualization is ensured by 2D or 3D (VRML) screen and can be monitored in a standard web browser. With this system it is very easy to replay missions or to do mission rehearsals.

This framework is useful for control algorithm development and testing because of its physical model implementation. Another important feature is the mission planning. This module allows the user to define an underwater mission in 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. Automatic logging of all communications that facilitates data retrieval for post-mission-analysis and mission reconstruction is another standard feature.

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 [7]. The main target here is to design a framework that allows cooperative mission planning and visualization.

Some effort is also being done to integrate sensor networks in future in such a way that missions can be on-line re-planned with 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 are 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 chosen from AUVs and ROVs (also UAVs and ASVs; Autonomous Surface Vehicles; in the near future). The mission execution with these classes of vehicles requires four main steps [9].

Operational setup. The mission starts with a recon-naissance of the site. This includes a preliminary study of the bathymetric profile in order to place the acoustic transponders (used for vehicle navigation) in an appropriate location. Moreover, the site must be accurately studied to detect natural obstacles. This is crucial for a correct mission programming. The position of transponders and the initial position of the vehicle may be determined by using a GPS or using triangulation with natural references. Another important stage in the operational setup is the determination of the data to be measured. Missions may be designed for getting bathymetric profiles, collect CTD (conductivity, temperature, depth) data, detect temperature gradients or inspect underwater structures.

Mission programming. The path of the vehicle(s) is defined and a mission is specified by selecting a pre-defined set of maneuvers and tasks. Depending on the vehicle class, tele-operation is also a selectable maneuver. At this stage, the vehicle(s) are linked with a support laptop, where an initial diagnostic is conducted. Obviously, if the underwater vehicle is an ROV the communication link is always plugged.

Mission execution. In AUV missions, during the execution period, the support computer is disconnected. Then, the AUV is positioned at the initial point, where it starts its mission. The vehicle is able to navigate completely autonomously. In the end, the AUV stops at a pre-defined position and is recovered. Since, there is always a connection between the operator and the on-board computer of the ROV, the, user may send direct commands through a joystick and can always visualize real-time data acquired by its sensors [10]. It is also possible to run a pre-programmed mission, as in the AUV case.

Mission analysis. In the end of the mission execution, the data acquired by the vehicle's sensors has to be analyzed.

The communication with AUV during mission execution is not easy since the acoustic modems have low bandwidth. The amount of data stored by the on-board computer may be large and it is not viable to send this data through acoustic modems.

The Neptus framework [11] should be designed to fulfill the operational mission requirements described above. Thus, the top level requirements are listed below and are represented in the use case diagram of Fig. 1:

- An application to define the environment of the operational mission. Including navigation references, bottom profile, obstacles, and mark points;
- An application for the mission programming;
- A console that establish the link between the support computer and the on-board computer. In the ROV class, the console should enable users to send joystick commands and visualize sensors' data in real-time;
- A simulation platform to allow users 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 of sensors data;
- A repository for gathered data with associated query services.

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

IV. ARCHITECTURE

A. Neptus Components

We can regard the Neptus architecture as organized in several applications/modules that, together, create the Neptus environment satisfying the requirements described in the previous section. In Fig. 2, the deployment diagram presents the five main applications and their connections.

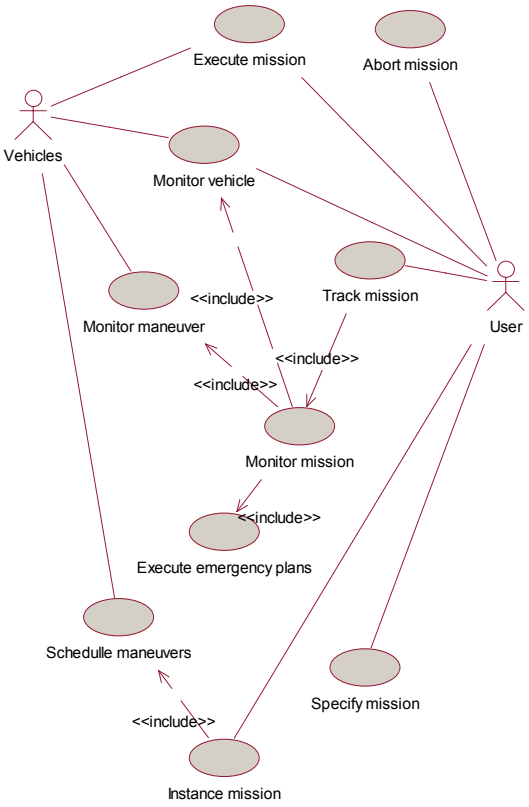

Fig. 1. Requirements use case diagram

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 is intended to be used in the mission preparation and setup. This activity includes the generation/edition of missions (by specifying a hybrid automaton), 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. 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.

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

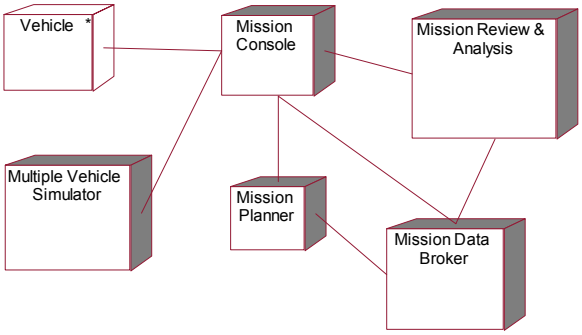

Fig. 2. Deployment diagram

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.

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.

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.

B. Data Structure

There are 3 main data elements that will be used to store data: the vehicle; the mission; and the map.

The vehicle XML file describes the main characteristics of a vehicle. Some of the properties stored are the dimensions of the vehicle and several pictures in selected angles in order to be represented in the MC, MP and MRA (a 3D view is also available). Additional important information included in this file is the specification of the maneuvers that the vehicle is able to perform. Also present is the information about sensors and actuators available on the vehicle, available communications channels and some configuration and transformation files.

The mission XML file will be used to describe a mission. As seen in Fig. 3, a mission is organized in four elements. The header represents generic information like a name, a type, a description and some notes. The definitions contain information of the vehicles that will be used in the mission, also a list of maps used and the home referential. The home referential is used to set a point and a coordinate system axis to be used for the data produced in the mission. The labels element will serve to set teams of vehicles that can be referenced in the plans.

The mission plans will reside in the body of the mission XML file. In this framework, a mission is defined as a set of tasks that a set of vehicles must perform in order to accomplish an objective. In complex missions, vehicles may cooperate to complete a specific task and humans can intervene in real-time in order to change some tasks so that the desired goal is better attained. The tasks will be defined in one or more plans.

The map XML file represents the map of the area where the mission will be executed. It will have information of

objects like: marks, transponders, geometrical shapes and paths.

The main advantage of the use of XML is to be able to use its schema (XSD – XML Schema Definition Language [6]) to check its validity using a common validating parser. There are several available parsers as open source projects that can be used. Additionally it is also a cross platform format. Other advantage of using XML as the data format is the dissemination. That is, data can be shared with partners and easily integrated/used in other applications because:

- Grammar is known by means of the XSD;
- Data can be transformed/manipulated using XSLT (eXtensible Stylesheet Language Transformations); and
- XML documents can be filtered.

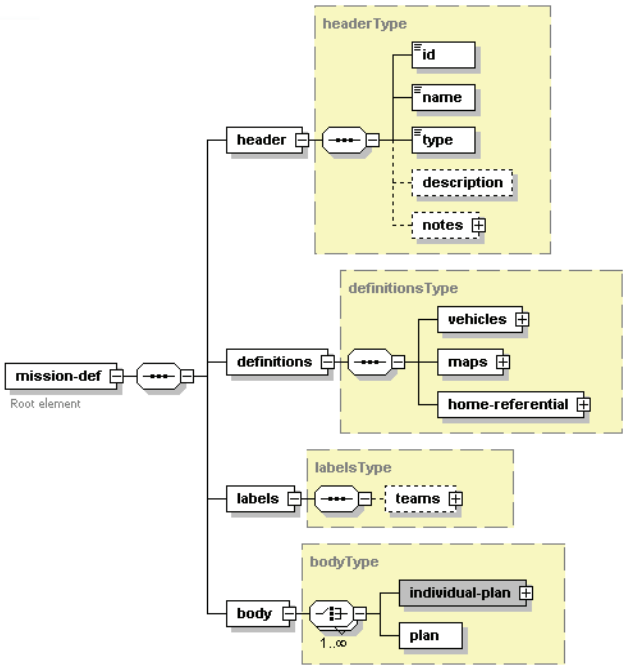

Fig. 3. Mission schema

In the near future a XML schema for the mission execution data will be developed from the basis of the binary data already being produced by the vehicle and the input format for the MRA.

C. Scenarios of Operation

There are two scenarios that had to be considered when developing the Neptus architecture. One is the local use of this framework. By local use, it is meant the use without any other link except with the vehicles. This is mostly what happens in operation scenarios where there is a PC, usually a laptop, and a cable, or radio link, connecting the PC to the vehicle.

The other scenario emerges in the pre or post mission state. Here, we have an open environment with Internet connections available.

Naturally we should also think in the more complex instance of the first scenario. That is one with a small local network that will make possible to interconnect both the vehicles and several PCs allowing cooperation between them.

In the first (and also the extended) scenario the MP, MC and MRA modules will be available. Additionally, in the extended first scenario, several MCs may be interconnected.

The MCs interconnection will allow the dissemination of information. This dissemination might be for passive following of the events of a running mission, or to be used for an active intervention allowing a coordinated operation on a high level.

On the second scenario all the modules will be available. Here, there is an additional concern related to data dissemination, namely, to partners or clients. In this situation, the data to be released may contain sensitive information. This may result in a denial of useful information to co-operating partners/clients. Filtering information into small, coherent, discrete packages (views) makes it easier to control and thus distribute to other partnership members.

To achieve this, new paradigms have to be used. One of them is described in [12] or [13]. Such approach enables to:

- Filter information objects from their sources;
- Publish, subscribe, query, and transform data objects; and
- Specify the policy governing how to disseminate and access data objects.

This would be a system of systems that manages, integrates, aggregates, filters and distributes information to cooperating partners.

D. The Mission Planner

As seen above the Mission Planner (MP) will be used to plan and evaluate a mission. This tool will be mainly used in a pre mission execution stage. In Fig. 4, is depicted the simplified use case that was used in development of this module.

The MP may be decomposed into several sub-components; each of them may be used inside other components. One of them is the mission editor. This will be the MP core component. With this editor, we are able to define a mission with all of the elements (Fig. 3) that were described above.

Another important subcomponent is the Map Editor Module (MEM), which is used to define maps. These maps will essentially be a set of objects, each with various characteristics like size, position or shape. The user will be able to insert any amount of objects on the map. The map objects can be marks, transponders, geometrical shapes (that can be parallelepipeds or ellipsoids), and free-hand and computer generated drawings (or paths).

Another module is the Mission Graph Editor Module (MGEM). This will be used to create the plans that are modeled as hybrid automatons. This editor will allow the visual construction of these automatons in which each node is a maneuver.

In the MP, we will have also the possibility to make a quick preview of the mission just to see a rough execution of the plans. This will be done by the Preview State Generator Module (PSGM). This module will be used in conjunction with the World State Renderer Module (WSRM), allowing it to perform an animation of the mission plan.

The WSRM will allow the visual representation of the world; and will be used in most of the main applications (MC, MP and MRA). There are several ways of displaying this: as text messages; as 2D view; and as 3D view.

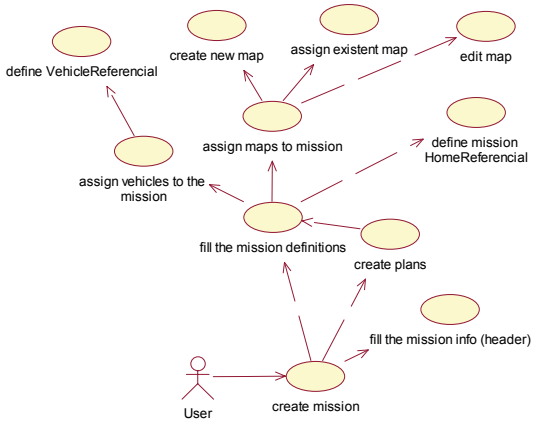

Fig. 4. Create mission use case diagram

V. PROTOTYPE IMPLEMENTATION

Following the Systems Engineering Process [2] an iterative development process was used. Several releases of the software were planned, beginning with a prototype implementation.

It is expected that there will be a wide variety of users using this software, ranging from scientists (biologists and others), engineers and archeologists. This vastness of users compelled us to create user-friendly software, showing only pertinent information and, whenever possible, providing tools for error detection of user defined parameters. Next, we explain the various approaches adopted to achieve these objectives.

A. Choosing Java as the Programming Language

Java is among the best languages to create visual user interfaces, having multiple APIs (Application Programming Interfaces) for this objective only.

We opted for the Swing API (http://java.sun.com/products/jfc/index.jsp) since it is the most flexible, providing a large variety of pre-made visual components and having a pluggable look and feel.

The ability to execute the software over various hardware platforms was indeed very important, since the end users of the software tend to use different hardware and software platforms.

Java is also a great language for team development and possesses excellent open source Integrated Development Environments (IDE) like the Eclipse™ IDE (http://www.eclipse.org).

B. Multidocument Interface

A mission (in Neptus) consists in various individual elements that have to be defined. To make that explicit for the user, we opted for a multidocument interface with the ability to browse over the various mission elements as seen on Fig. 5. When the user loads or creates a new mission, all the available items (there are some items that are created by default) are exposed in a tree-like interface. This interface allows the user to delete, view and alter previously defined elements along with the possibility to create new ones.

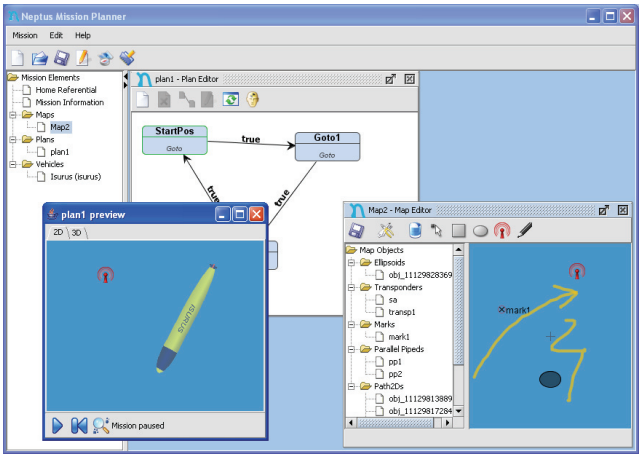

Fig. 5. Mission Planner application screen shot

C. Real World Coordinates

The planned missions will take place in the real world so all the locations defined with this software have to be set in real world coordinates. Since there are many ways of providing this kind of information (and different areas tend to use different ones), an effort was made to make the definition of locations as flexible as possible.

The users can define a location with its absolute Latitude, Longitude and Height/Depth, or may provide an offset from a different location. These offsets can be provided in the form of orthonormal offsets (North/South, East/West, and Up/Down) or spherical coordinates offsets (Distance, Azimuth, Zenith). Once defined, a location may be viewed and altered in the same or any other of the available ways (Fig. 6).

Again, this was achieved by following Object Oriented principles, i.e., by encapsulating a location in a single Object (OO Definition) that provides various ways of initialization (contructors) and, once initialized, its information may be accessed in multiple ways.

D. Definition of an Individual Plan

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 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).

E. Mission Graph Editor

This component represents an individual plan as a graph, allowing the user to insert new nodes (maneuvers) in the graph and to create edges between the existent nodes.

After selecting a node (or edge) for edition, the user is presented with a window where the defined parameters for that item can be viewed or altered.

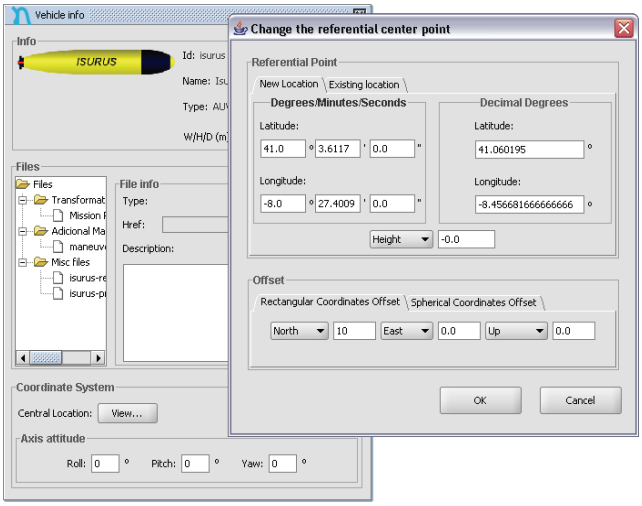

Fig. 6. Coordinate editing panel

To create this component we used the JGraph API (http://www.jgraph.com). This API facilitated the development since it implements all the basic functionalities that a graph editor should have and proved to be very robust and lightweight.

F. Quick Preview of Individual Plans

To prevent possible errors in the definition of individual plans, the framework includes the possibility for the user to preview the execution of the current defined plan.

Currently there is a method for the maneuvers to define its possible implementation. This method consists in the reference of a file URL (Java class) that will generate the different vehicle states based in the current world and vehicle states. In the future, this definition will evolve to be a hybrid automaton definition that will be parsed and interpreted by the Neptus Framework.

This capability showed to be very important because it provides an easy and quick way for the user to completely verify if a mission is defined according to the objectives.

The end user may chose to view the pre-visualization in the form of text messages (being possible to save these messages to a text file) or view a visual animation of the plan. The animations, in turn, can be viewed either in 2 or 3 dimensions. We used the Java2D and Java3D API's to provide these features mostly because they were very easy to integrate with the rest of the application.

G. Creation of Environmental Maps

The MP application also includes a tool that allows the visual definition of environmental maps, the Mission Map Editor (Fig. 5). This editor allows the insertion of different kinds of elements to be possible to recreate the conditions encountered in the real world environment.

Once created, a map can be saved in an individual XML file making it possible the reutilization of its information. When creating a mission, the user can opt to create new maps for that mission or use already existing maps stored in XML files.

VI. CONCLUSIONS AND FUTURE WORK

This paper presented the current state of development of the Neptus framework, a mixed-initiative environment to support the coordinated joint operation of multiple autonomous and semi-autonomous vehicles.

The modules MP, MRA and MC are in advanced staged of developing and already being used (in case of MC and MRA) or in the final stages of development (in case of the MP). In the case of MVS we already have a basic simulator that is being used with the MC for tests. Next steps will include some work in extending the MC to fully support the extended first scenario and also the MDB will start to be implemented.

ACKNOWLEDGMENTS

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

This research has been partly supported by Agência de Inovação under projects PISCIS and KOS.

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" (provisory title), Today's technology for a sustainable future, OCEANS Europe 2005, Brest, France, June 20-23, 2005, in press.

[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/> (April 2005).

[4] Deep Ocean Engineering, <http://www.deepocean.com> (April 2005).

[5] Naval Postgraduated School, <http://terra.cs.nps.navy.mil/AUV/workbench> (April 2005)

[6] XML – Extensible Markup Language 1.0 (Third Edition), W3C Recommendation 4th February 2004, <http://www.w3c.org/TR/2004/REC-xml-20040204/> (March 2005).

[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. Neves, 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] 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.

[11] Neptus, <http://whale.fe.up.pt/neptus> (April 2005)

[12] G. M. Gonçalves, P. S. Dias, A. Santos, J. B. Sousa, F. L. Pereira, "An Implementation of a Framework for Cooperative Engineering", IFAC 2005, Praha, Czech Republic, July 4-8, 2005, in press.

[13] Marmelstein, R., Force Templates: A Blueprint for Coalition Interaction within an Infosphere, IEEE Intelligent Systems, vol. 17, 2002, pp. 36-41.