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
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.
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/editation 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.

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.

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.
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.

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 (constructors) 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.
F. Quick Preview of Individual Plans

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.

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