Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online
CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 48-2 (2015) 256–261

Towards Programmable Coordination of Unmanned Vehicle Networks [★]

Eduardo R. B. Marques [*] **Manuel Ribeiro** [**] **José Pinto** [**] **João B. Sousa** [**] **Francisco Martins** [*]

[*] *LASIGE/Departamento de Informática, Faculdade de Ciências da Universidade de Lisboa*
[**] *Laboratório de Sistemas e Tecnologia Subaquática, Faculdade de Engenharia da Universidade do Porto*

Abstract: The use of unmanned vehicle networks for diverse applications is becoming widespread. It is generally hard to program unmanned vehicle networks as a "whole", however. The coordination of multiple vehicles requires careful planning through intricate human intervention, and a high degree of informality is implied in what concerns the specification of a "network program" for an application scenario. In this context, we have been developing a programming language for expressing global specifications of coordinated behavior in unmanned vehicle networks, the Networked Vehicles' Language (NVL). In this paper we illustrate the use of the language for a thermal pollution plume tracking scenario employing unmanned underwater vehicles.

Keywords: Unmanned vehicles, Coordination, Cooperative control, Programming

1. INTRODUCTION

Autonomous vehicles are making their way to everyday mainstream use. In particular, many real-world applications can take advantage of multiple autonomous vehicles that operate cooperatively over a networked environment e.g., see (Bellingham and Rajan (2007); Dunbabin and Marques (2012); Sousa et al. (2014)). Large systems are being deployed with a massive integration of unmanned vehicles, sensors, and human user interaction, that can also be spatially distributed across the globe (Isern and Clark (2003); Petrioli et al. (2014)).

[★] The research leading to these results has received funding from the "NETMAR" Project, 2011-1/176, funded by ERDF - Atlantic Area Transnational Cooperation Programme.

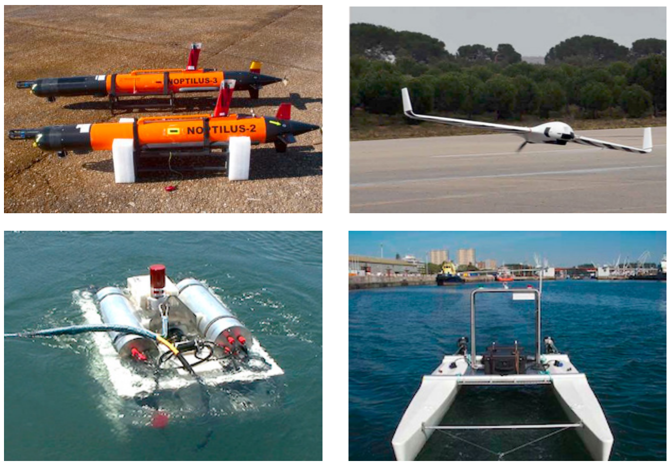


Fig. 1. Autonomous vehicles developed at LSTS

The Laboratório de Sistemas e Tecnologias Subaquática (LSTS) has been building and operating autonomous vehicles for over a decade. Field operations now routinely employ multiple vehicles for diverse uses (Faria et al. (2014); González et al. (2012); Martins et al. (2011); Pinto et al. (2013b); Sousa et al. (2014)), involving a networked environment formed by human operators and heterogenous kinds of vehicles, like unmanned underwater vehicles (UUVs), unmanned air vehicles (UAVs), remotely operated vehicles (ROVs), or autonomous surface vehicles (ASVs). Some of vehicles developed by LSTS are shown in Fig. 1.

A core challenge faced by these deployments lies on the specification of multi-vehicle coordination. Typically, this is accomplished through separate vehicle scripts that informally "glue" during operation, as opposed to global and self-contained specifications of an intended global behavior, which we may call "network programs". With these "network programs" in mind, we have been developing the Networked Vehicle's Language (NVL), introduced in (Marques et al. (2015)).

A single NVL program is able to express an on-the-fly selection of multiple vehicles in a network and their allocation to tasks, subject to several types of constraints. For instance, NVL programs may fire concurrent tasks in distinct vehicles over the same period of time, order these task groups in sequence, and constrain the time bounds for their execution. NVL programs also adapt to the dynamics of the network "cloud", where vehicles come and go, have intermittent connectivity, and change their spatial location over time. The initial prototype of the language is already able to express commonly used patterns in multi-vehicle operations, and some field-tests have already been conducted (Marques et al. (2015)).

10.1016/j.ifacol.2015.06.042

There are several heterogeneous approaches for modelling the coordination of tasks in unmanned vehicle networks, e.g., dynamic and hierarchical hybrid automata networks (Sousa et al. (2004)), “vignette scripts” that map onto abstract state machines (Shahir et al. (2012)), the use of distributed deliberative planning using “timelines” (Pinto et al. (2012)), dynamically-changeable Petri nets (Love et al. (2014)), or combined bigraph/actor models (Pereira et al. (2013)). The concerns of NVL are orthogonal to these approaches. The language focuses on foundational programming constructs that serve as core building blocks for operating unmanned vehicle networks. Beyond direct use, we think of NVL as a backend coordination language for other higher-level modelling frameworks and semantic abstractions in the future.

In this paper, we focus on the practical use of the language. We present a thermal pollution plume tracking scenario using UUVs and the use of NVL to accomplish it. We describe the example scenario in Section 2, and the NVL program for its realisation in Section 3. Section 4 describes the coordination architecture and the software toolchain for the specification and execution of NVL programs. We conclude the paper with some final remarks in Section 5.

2. EXAMPLE SCENARIO

We begin by describing an example application scenario, depicted in Fig. 2. The scenario at stake is adapted from operations conducted by LSTS and partners in Marjan peninsula at Croatia (Sousa et al. (2014)). There, three unmanned underwater vehicles (UUVs) carrying rhodamine sensors were used to detect and sample a simulated pollution plume. During the experiment surface vehicles and aerial vehicles were used to relay information from the AUVs to shore, where the base station was located. In similar spirit, we consider the localisation and mapping of a thermal pollution plume over an area of interest through adaptive data sampling, employing three UUVs. The rationale is that two UUVs track the operation zone into distinct sections to track the origin of the plume. Since the survey area can be too large for direct communication range, an extra UUV serves as a communication gateway in-between the vehicles and between vehicles and also nodes on the shore where human operators may monitor and command the overall operation.

The data sampling is iterative. The survey UUVs initially cover a large area in coarse-grained manner to try to track the probable source of the pollution. Once a survey step is over, the temperature measurements are examined and the zone for the next step is redefined, if found necessary. In that case, the area is typically reduced and shifted towards the hotter area, to get a more accurate temperature mapping of the potential pollution plume. The operation area is reprogrammed (reduced in size and relocated) and again split between the UUV vehicles for a new survey step. The data analysis and area refinement may either be performed by a human user or cooperatively by custom controllers running onboard each of the survey vehicles.

The plot in Fig. 3 illustrates how the scenario may unfold, considering the simulated physical location of Leixões harbour in Portugal. The data at stake was derived from

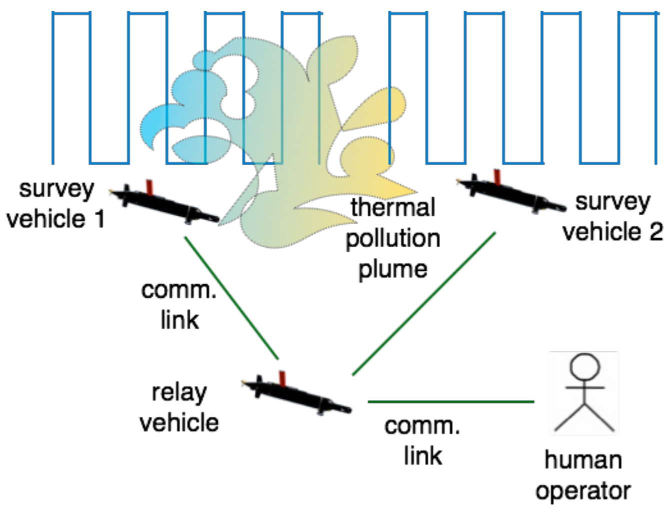


Fig. 2. Example scenario

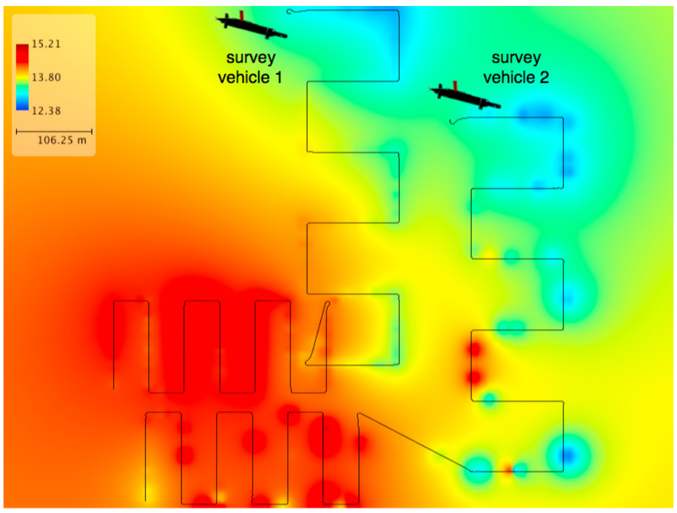


Fig. 3. Thermal pollution plume tracking

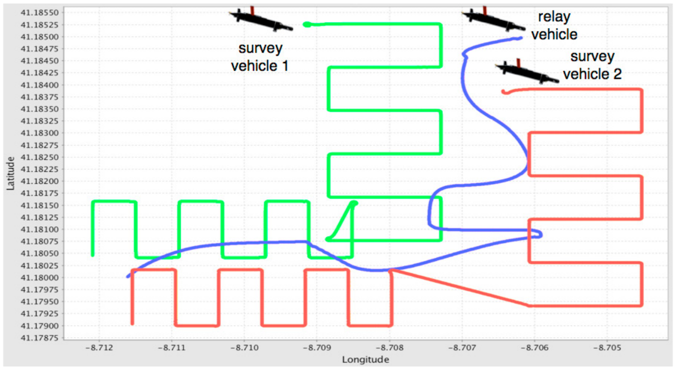


Fig. 4. Vehicle positions

the execution of the NVL program we describe later in Section 3. The paths of the two survey vehicles are shown, along with (also simulated) temperature measurements across the operation area. Higher temperature values indicate a probable source of pollution. The vehicles first scan the area looking for the plume using a coarse-grained row pattern. After this first scan, an human user or an automated software module may analyse the data and derive a new sub-area of interest, attending to the distribution of temperature values and to the zones where these are higher. In the second scan, again using row patterns, the plume is found and mapped in more fine-grained manner.

Fig. 4 displays the vehicle positions for the same data set, but also depicting the movement of the relay vehicle. The aim is that the relay vehicle positions itself such that communications performance and connectivity among vehicles and offshore nodes is optimised. This can be accomplished by a communications relay maneuver that is implemented in LSTS unmanned vehicles (Pinto et al. (2013a)). When executing this maneuver, a vehicle monitors the position of other nodes in the network and moves in order to maintain a certain minimal distance to these nodes, whilst also positioning itself in a manner that optimises communication performance (e.g., by moving to the geometric center of nodes' positions).

3. NVL PROGRAM FOR PLUME TRACKING

Fig. 5 lists a NVL program that realizes the pollution plume tracking scenario. The program declares a set of tasks and a procedure made up of instructions that select vehicles and associate the execution of tasks to vehicles.

```
// Task declarations                                  1
task approachArea1 (vehicle sv)                       2
task approachArea2 (vehicle sv)                       3
task surveyArea1 (vehicle sv)                         4
task surveyArea2 (vehicle sv)                         5
task relay (vehicle relayV,                           6
            !vehicle sv1,                             7
            !vehicle sv2 )                            8
// Main procedure                                     9
proc main() {                                         10
 // Select UUVs.                                      11
 select 5 m {                                         12
  relayVehicle                                        13
   ( type: "UUV" area: 41.18500 −8.70620 1            14
     id: "lauv−seacon−3" )                            15
  surveyVehicle1                                      16
   ( type: "UUV" area: 41.18500 −8.70620 1)           17
  surveyVehicle2                                      18
   ( type: "UUV" area: 41.18500 −8.70620 1)           19
 }                                                    20
 then {                                               21
  do {                                                22
   step 15 m {                                        23
    approachArea1(surveyVehicle1)                     24
    approachArea2(surveyVehicle2)                     25
    relay(relayVehicle,                               26
          !surveyVehicle1, !surveyVehicle2)           27
   }                                                  28
   step 60 m {                                        29
    surveyArea1(surveyVehicle1)                       30
    surveyArea2(surveyVehicle2)                       31
    relay(relayVehicle,                               32
          !surveyVehicle1, !surveyVehicle2)           33
   }                                                  34
   input 20 m userInput                               35
    { "continue" "stop" } default "stop"              36
  }                                                   37
  while (userInput = "continue")                      38
 }                                                    39
}                                                     40
```

Fig. 5. Plume tracking program

3.1 Task declarations

A task is modelled as an indivisible unit of timed computation that requires one or more vehicles in order to execute. Tasks are merely declared by a program, as the purpose of the program is to orchestrate task execution. Thus, the invocation interface of tasks needs to be exposed, but not the actual logic inherent to their implementation. We discuss the actual mechanisms for the implementation of tasks in Section 4.

The plume tracking program declares five tasks, from line 2 to line 8 in Fig. 5: approachArea1, approachArea2, surveyArea1, surveyArea2, and relay. In relation to the target scenario, approachArea1 and surveyArea1 are the tasks to be executed by the first survey vehicle to respectively approach and survey its sampling area, approachArea2 and surveyArea2 play a similar role for the second survey vehicle, and relay is the task to be executed by the relay vehicle. Each task requires only one vehicle to execute, and the relay task additionally takes two vehicle reference arguments, indicated by the ! **vehicle** syntax. The intent of vehicle references is to make tasks aware and "linked" to the presence of other vehicles, even if those vehicles are independently executing other tasks. In the example, the relay task is then supplied with references to the survey vehicles.

Some other NVL task declaration features are not used by the plume tracking program. For instance, tasks may require more than one vehicle resource, when they require the cooperative engagement of vehicles in tightly coupled manner, and they can declare output values that may be used in the program body to govern control flow – see (Marques et al. (2015)) for further details and related examples.

3.2 Procedures

NVL procedures contains sequences of instructions that express selection of vehicles and subsequent allocation of selected vehicles to tasks. The execution of a program starts with a procedure called main. In the plume tracking program there is just the main procedure, lines 10–40).

Vehicle selection. The three UUV vehicles are selected from the network using a **select** instruction at line 12. The vehicles are identified by variables relayVehicle, surveyVehicle1, and surveyVehicle2 declared in the following lines. The instruction also has an associated deadline of 5 minutes, 5 m in the code, and states selection filters per each vehicle.

All three vehicle filters specify the selection of a UUV class vehicle, **type**: UUV in the code, located in a 1 kilometre range of latitude/longitude coordinates 41.18500 N, 8.70620E, as specified by **area**: 41.18500 −8.70620 1. The **id**: "lauv−seacon−3" additional filter used for selecting relayVehicle indicates that the vehicle must be a precise one that goes by the identification of "lauv-seacon-3". The absence of such a filter for surveyVehicle1 and surveyVehicle2 means that the survey vehicles can be any other UUV vehicles in the desired area.

The program stops if (any of) the vehicles cannot be selected within the 5 minute deadline, in line with the default error handling mechanism in NVL. Custom error handling blocks may be specified (Marques et al. (2015)) but are omitted in the example for simplicity.

Task execution. If the UUVs are successfully selected, the **then** block that associates to the **select** instruction in the program is carried out, i.e., the instructions from line 21 to line 39. The **then** block contains a cycle expressed by a **do** { ... } **while** (...) construct (lines 22–38) where each iteration proceeds as follows:

(1) The first **step** instruction (lines 23–28) fires the approachArea1, approachArea2, and relay tasks. These will be executed simultaneously by the surveyVehicle1, surveyVehicle2, and relayVehicle vehicles respectively, with a shared deadline of 15 minutes. The program advances to the next instruction only when all tasks for the step complete. Observe that when a vehicle completes its own task, it may have to execute some default maneuver for some time, if there are tasks in the same step still completing in other vehicles. The **step** instruction will fail if some of the tasks do not complete in the specified deadline. In that case, similarly to vehicle selection, the default error handling mechanism is to stop the program.
(2) The second **step** instruction (lines 29–34) proceeds in analogous manner to the first one. It has a deadline of 60 minutes, and fires the surveyArea1 and surveyArea2 tasks for surveyVehicle1 and surveyVehicle2, plus the relay task again for relayVehicle.
(3) An iteration ends with an **input** instruction (lines 35–36). This asks the human operator or a software module that commands the NVL program for a value in 20 minutes, which should be either "continue" or "stop". The intent is to give time for data analysis to take place to decide if another thermal plume sampling iteration should execute ("continue") or not ("stop"). The instruction's time frame is used to perform the necessary analysis on the thermal plume data, and, in case another iteration is required, also to reprogram the survey vehicles' thermal plume sampling areas/tasks on the background.
(4) When the **do** { ... } **while** (...) loop is finished, so is the **then** block and also the program, since there are no more instructions in sequence. At this point the vehicles are released from duty to the network by the program back to the "network cloud".

4. IMPLEMENTATION

4.1 Overview

The architecture for the execution of NVL programs is illustrated in Fig. 6. The overall aim of the architecture is that NVL programs interact with multiple unmanned vehicles and possibly also human operators for a coordinated behavior amongst all participants.

The main aspects of the architecture are as follows:

- NVL programs are written and validated using an integrated development environment (IDE) by an user. A validated program can then be executed by an NVL interpreter.

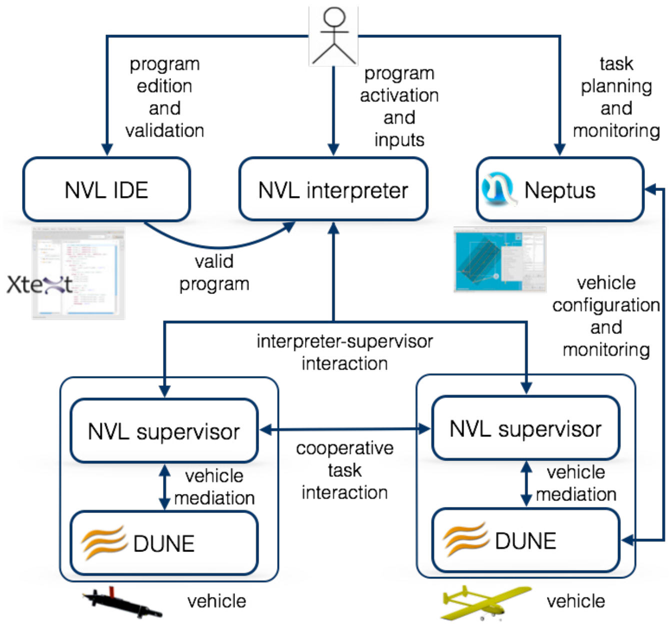

program edition and validation
program activation and inputs
task planning and monitoring
NVL IDE
NVL interpreter
Neptus
Xtext
valid program
vehicle configuration and monitoring
interpreter-supervisor interaction
NVL supervisor
cooperative task interaction
NVL supervisor
vehicle mediation
vehicle mediation
DUNE
DUNE
vehicle
vehicle

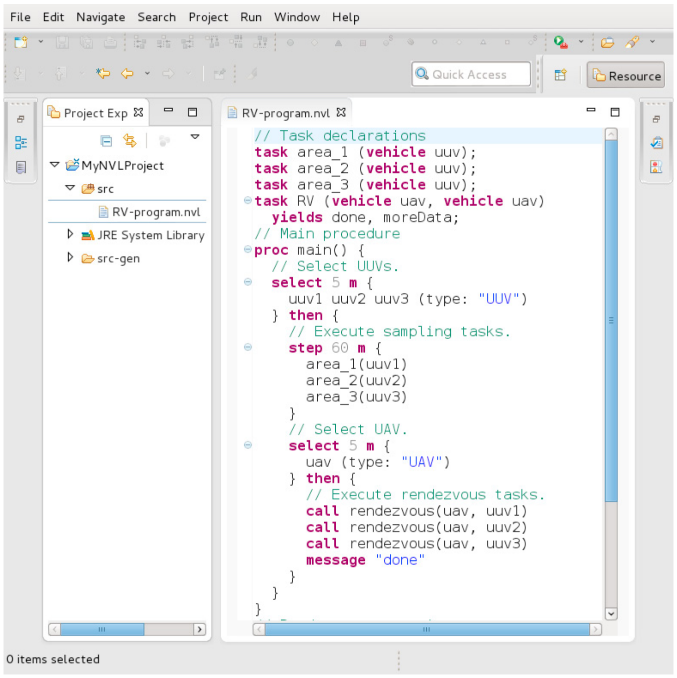
Fig. 6. Coordination architecture

- The execution of NVL program within the interpreter relies on the interaction with multiple vehicles. This is accomplished through communication with NVL supervisor modules that ran onboard each vehicle. NVL supervisors handle and fire tasks that execute on the vehicle they are responsible for.
- NVL programs also interact through human-user inputs, whenever user intervention is necessary for coordination, i.e., through the **input** instruction as in the example program.
- Three components from the LSTS open-source software toolchain (Pinto et al. (2013a)), available from `http://github.com/LSTS`, aid the execution of NVL programs: (1) the IMC interoperability protocol is used for communication amongst all components during the execution of a program; (2) NVL supervisors interact with DUNE, the onboard software system that directly controls the vehicles; and (3) the Neptus system is used to encode vehicle tasks in the form of IMC plans and monitor their subsequent execution.

To derive the plots of Section 2, we deployed this architecture in a simulation environment. The difference from an actual NVL field-test deployment (Marques et al. (2015)) is that DUNE runs with a simulation profile, employing an UUV physics simulation engine (da Silva et al. (2007)) and sensor/actuator simulator task drivers. We next describe the role of the architectural components and their interaction in more detail.

4.2 NVL program specification

To specify NVL programs, a user employs the NVL IDE. The purpose of the tool is to write and validate a program, offering an user-friendly GUI interface that integrates with the popular Eclipse programming environment. A screenshot of the IDE is shown in Fig. 7. The tool is developed in Xtext, a popular open-source toolkit for implementing domain-specific languages (Bettini (2014)), available from `http://eclipse.org/Xtext`. Xtext supports the traditional tasks of typical language design and implementation, such as the definition of the language grammar, syntactic parsing or semantic validation.



```
// Task declarations
task area_1 (vehicle uuv);
task area_2 (vehicle uuv);
task area_3 (vehicle uuv);
task RV (vehicle uav, vehicle uav)
  yields done, moreData;
// Main procedure
proc main() {
  // Select UUVs.
  select 5 m {
    uuv1 uuv2 uuv3 (type: "UUV")
  } then {
    // Execute sampling tasks.
    step 60 m {
      area_1(uuv1)
      area_2(uuv2)
      area_3(uuv3)
    }
    // Select UAV.
    select 5 m {
      uav (type: "UAV")
    } then {
      // Execute rendezvous tasks.
      call rendezvous(uav, uuv1)
      call rendezvous(uav, uuv2)
      call rendezvous(uav, uuv3)
      message "done"
    }
  }
}
```

Fig. 7. Program specification using the NVL IDE

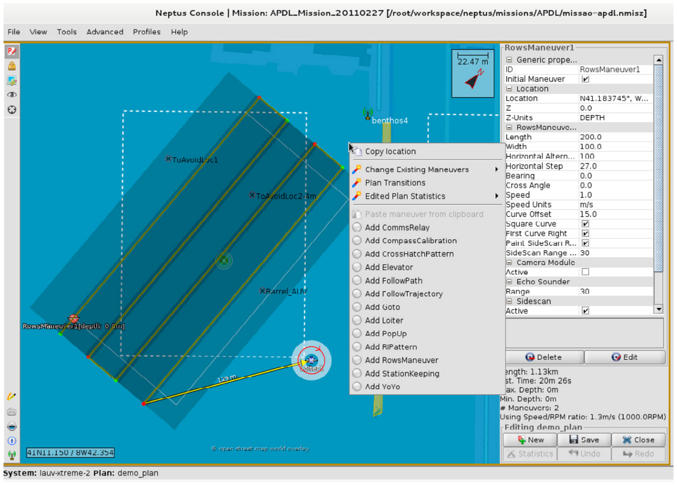
4.3 Task planning

The simplest implementation for NVL vehicle tasks is provided by maneuver plans, which can be edited in Neptus, serialised to IMC format, and then directly executed within a vehicle by DUNE. These plans are composed of sequences of maneuvers that range from simple waypoint tracking or loitering to more more complex maneuvers, like data surveys over a region with a variety of spatial patterns. As an example, the edition of a row-pattern maneuver for data sampling in Neptus is shown in Fig. 8.

An NVL task may be solely defined by an IMC plan, that is simply passed on from NVL supervisor to DUNE, or, in more elaborate form, be implemented by custom controller code that runs within a supervisor. Custom controllers are used to implement behavior beyond the level of abstraction of IMC plans, particularly cooperative tasks, e.g., multi-vehicle adaptive data sampling or vehicle rendez-vous tasks for data transfer/muling (González et al. (2012); Marques et al. (2007, 2015); Pinto et al. (2013b)). These controllers may fire one or more generated IMC plans to DUNE, that can either be pre-programmed or generated on-the-fly.

4.4 Program execution

The execution of a program comprises the interaction between the interpreter and the supervisors onboard each NVL-enabled vehicle. The code of both these components is written in Java, and executes using the low-footprint Java SE Embedded runtime environment. Supervisors attend to the interpreter's orders for task execution, and report back related state, mediating access to the local



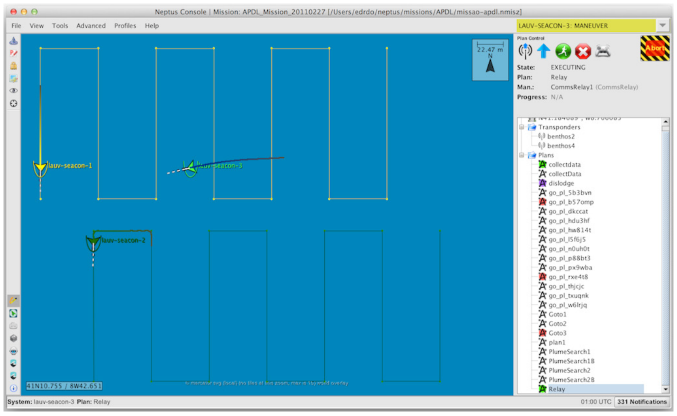
RowsManeuver1

Generic prope...	
ID	RowsManeuver1
Initial Maneuver	✓
Location	
Location	N41.183745°, W...
Z	0.0
Z-Units	DEPTH
RowsManeuve...	
Length	200.0
Width	100.0
Horizontal Altern...	100
Horizontal Step	27.0
Bearing	0.0
Cross Angle	0.0
Speed	1.0
Speed Units	m/s
Curve Offset	15.0
Square Curve	✓
First Curve Right	✓
Paint SideScan R...	✓
SideScan Range ...	30
Camera Module	
Active	☐
Echo Sounder	
Range	30
Sidescan	
Active	✓

```
ength: 1.13km
st. Time: 20m 26s
ax. Depth: 0m
Min. Depth: 0m
# Maneuvers: 2
Using Speed/RPM ratio: 1.3m/s (1000.0RPM)
Editing demo_plan
System: lauv-xtreme-2 Plan: demo_plan
```

Fig. 8. Edition of IMC plans using Neptus

DUNE instance that directly controls the vehicle. When cooperative task controllers run embedded in NVL supervisors, controllers in distinct vehicles also interact among themselves through supervisor-to-supervisor communication. During execution, the progress of IMC plans executed by DUNE can be monitored using Neptus. A Neptus screenshot during execution of the plume tracking scenario is shown in Fig. 9.



```
State:     EXECUTING
Plan:      Relay
Man.:      CommsRelay1 (CommsRelay)
Progress:  N/A
System: lauv-seacon-3 Plan: Relay
```

Fig. 9. Task monitoring in Neptus

4.5 IMC and networking

IMC (Martins et al. (2009); Pinto et al. (2013a)) defines an extensible interoperability protocol for data exchanged in networks composed of unmanned vehicles, human operators, or environmental sensors. All components in our architecture communicate through IMC messages, covering the general aspects of as vehicle interface, maneuver plan specifications, and node announcement/discovery. We extended IMC with a few NVL-specific messages for the interpreter-supervisor interactions plus supervisor-to-supervisor interaction during cooperative maneuvers.

5. CONCLUSION

We presented the use of the NVL coordination language in an example scenario of thermal pollution plume tracking using a team of UUVs. The language is being actively developed further to accommodate to new requirements. Future work directions include:

- Adding more expressiveness to the language, e.g., extensions for supporting vehicle "teams" and associated dynamics (Shahir et al. (2012); Sousa et al. (2004)).
- Further integration with the LSTS toolchain, including the development of a Neptus NVL plugin, and the use of NVL as a backend language for deliberative planning Pinto et al. (2012, 2013a).
- Formal analysis of a NVL programs, in order to establish a priori guarantees on program execution. For instance, the following general problem of feasibility may be stated: can program P accomplish its tasks in t time with a vehicle set V under constraints C ? Basic timing properties may naturally be analysed owing to the inherently timed nature of NVL programs, e.g., estimation of a program's execution time under given resource and input assumptions. More generally, we are interested in the emerging approach of contract-based design for cyber-physical systems (Derler et al. (2013); Sangiovanni-Vincentelli et al. (2012)).

REFERENCES

Bellingham, J. and Rajan, K. (2007). Robotics in remote and hostile environments. *Science*, 318(5853), 1098–1102.

Bettini, L. (2014). *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing.

da Silva, J., Terra, B., Martins, R., and de Sousa, J. (2007). Modeling and simulation of the LAUV autonomous underwater vehicle. In *Proc. MMAR*. IFAC.

Derler, P., Lee, E., Tripakis, S., and Törngren, M. (2013). Cyber-physical system design contracts. In *Proc. ICCPS*. ACM.

Dunbabin, M. and Marques, L. (2012). Robots for environmental monitoring: Significant advancements and applications. *IEEE Robotics Automation Magazine*, 19(1), 24–39.

Faria, M., Pinto, J., Py, F., Fortuna, J., Dias, H., Martins, R., Leira, F., Johansen, T., Sousa, J., and Rajan, K. (2014). Coordinating UAVs and AUVs for Oceanographic Field Experiments: Challenges and Lessons Learned. In *Proc. ICRA*.

González, J., Masmitjà, I., Gomáriz, S., Molino, E., Del Río, J., Mànuel, A., Busquets, J., Guerrero, A., López, F., Carreras, M., Ribas, D., Carrera, A., Candela, C., Ridao, P., Sousa, J., Calado, P., Pinto, J., Sousa, A., Martins, R., Borrajo, D., Olaya, A., Garau, B., González, I., Torres, S., Rajan, K., McCann, M., and Gilabert, J. (2012). AUV based multi-vehicle collaboration: Salinity studies in Mar Menor Coastal lagoon. In *Proc. NGCUV*. IFAC.

Isern, A. and Clark, H. (2003). The Ocean Observatories Initiative: A continued presence for interactive ocean research. *Marine Technology Society Journal*, 37(3), 26–41.

Love, J., Jariyasunant, J., Pereira, E., Zennaro, M., Hedrick, K., Kirsch, C., and Sengupta, R. (2014). CSL: A Language to Specify and Re-Specify Mobile Sensor Network Behaviors. In *Proc. RTAS*. IEEE.

Marques, E., Pinto, J., Kragelund, S., Dias, P., Madureira, L., Sousa, A., Correia, M., Ferreira, H., Gonçalves, R., Martins, R., Horner, D., Healey, A., Gonçalves, G., and Sousa, J. (2007). AUV control and communication using underwater acoustic networks. In *Proc. IEEE Oceans Europe*. IEEE.

Marques, E., Ribeiro, M., Pinto, J., Sousa, J., and Martins, F. (2015). NVL: a coordination language for unmanned vehicle networks. In *Proc. 30th ACM/SIGAPP Symposium On Applied Computing*, SAC'15. ACM.

Martins, R., Dias, P., Marques, E., Pinto, J., Sousa, J., and Pereira, F. (2009). IMC: A Communication Protocol for Networked Vehicles and Sensors. In *Proc. IEEE Oceans Europe (OCEANS'09)*. IEEE.

Martins, R., Sousa, J., and Afonso, C. (2011). Shallow-water surveys with a fleet of heterogeneous autonomous vehicles. *Sea Technology*, 52(11), 27–31.

Pereira, E., Kirsch, C., R.Sengupta, and Sousa, J. (2013). BigActors - A Model for Structure-aware Computation. In *Proc. ICCPS*. ACM.

Petrioli, C., Petroccia, R., Potter, J., and Spaccini, D. (2014). The SUNSET framework for simulation, emulation and at-sea testing of underwater wireless sensor networks. *Ad Hoc Networks and Physical Communication.*

Pinto, J., Dias, P., Martins, R., Fortuna, J., Marques, E., and Sousa, J. (2013a). The LSTS Toolchain for Networked Vehicle Systems. In *Proc. Oceans*. IEEE.

Pinto, J., Faria, M., Fortuna, J., Martins, R., Sousa, J., Queiroz, N., Py, F., and Rajan, K. (2013b). Chasing fish: Tracking and control in a autonomous multi-vehicle real-world experiment. In *Proc. Oceans*. MTS/IEEE.

Pinto, J., Sousa, J., Py, F., and Rajan, K. (2012). Experiments with deliberative planning on autonomous underwater vehicles. In *Proc. WREM/IROS*.

Sangiovanni-Vincentelli, A., Damm, W., and Passerone, R. (2012). Taming Dr. Frankenstein: Contract-based design for cyber-physical systems. *European Journal of Control*, 18(3), 217–238.

Shahir, H., Glässer, U., Farahbod, R., Jackson, P., and Wehn, H. (2012). Generating test cases for marine safety and security scenarios: a composition framework. *Security Informatics*, 1(1), 1–21.

Sousa, J., Simsek, T., and Varaiya, P. (2004). Task planning and execution for UAV teams. In *Proc. CDC*. IEEE.

Sousa, J., Ferreira, F., Costa, M., and Oliveira, M. (2014). Building oceanographic and atmospheric observation networks by composition: unmanned vehicles, communication networks, and planning and execution control frameworks. In *AGU Fall Meeting*.