Graphical Mission Specification and Partitioning for Unmanned Underwater Vehicles[1]

Gary Giger, Mahmut Kandemir
Pennsylvania State University/Department of Computer Science and Engineering, University Park, PA 16802, USA
{giger, kandemir}@cse.psu.edu

John Dzielski
Pennsylvania State University /Applied Research Laboratory
University Park, PA 16802, USA
jed@enterprise.arl.psu.edu

Abstract - **The use of Unmanned Underwater Vehicles (UUVs) has been proposed for several different types of applications including hydrographic surveys (e.g., mapping the ocean floor and exploring sunken wreckage), mine detection and identification, law enforcement (e.g., enforcing certain fishing regulations), environmental and pollution monitoring, and even performing surveys to find potential drilling locations on the ocean floor for the oil industry. Recently the idea of using multiple, cooperating UUVs to execute these missions has also been proposed. There are two main factors that dictate a particular mission's success. The first factor regards creating a mission that is free from errors, in terms of both syntax and semantics. The second factor deals with properly splitting a mission into a set of sub-missions and assigning each sub-mission to a group of UUVs. Even though tools have been developed to help reduce these potential problems such as high level mission programming languages, compilers for these languages and utilities to automatically split an operator specified mission, the potential still exists for errors when creating a mission (e.g. semantic errors introduced from programming and maintaining the code for existing missions). The goal of this article is to present a programming-free, parallel mission generation utility that uses a series of tools we developed along with a commercially available graphical package. Our utility allows an operator to graphically specify a mission for a group of UUVs and automatically split the mission among the group based on an objective provided by the operator. The main contribution of this tool is twofold. First, it relieves the operator from low-level mission programming including the manual partitioning of the mission across a group of available UUVs. Second, it allows the operator to review the resulting set of generated sub-missions using the graphical interface. Thus, no matter what the particular UUV application is, this tool is another step towards successfully creating missions for UUVs.**

Index Terms – **High-Level Mission Programming, UUV, Graphical Mission Specification, Mission Planning, Compiler.**

1: The preliminary version of this paper appears in the Proceedings of the OCEANS 2007 MTS/IEEE. This paper extends the conference version by (1) providing additional graphical tools eliminating the need for an operator to write any source code for UUV missions and (2) providing two new strategies for generating sets of parallel sub-missions (reducing the number of vehicles and reducing total mission latency).

I. INTRODUCTION

The use of Unmanned Underwater Vehicles (UUVs) has been proposed for several different types of applications including hydrographic surveys for mapping the ocean floor and exploring sunken wreckage [3], mine detection and identification [4], law enforcement such as enforcing certain fishing regulations [16], collecting data on various forms of aquatic life [27], environmental monitoring regarding pollution [17], and even performing surveys to find potential spots on the ocean floor for oil drilling operations and for the inspection of underwater oil pipelines [5][28]. Recently the idea of using multiple, cooperating UUVs to execute these missions has also been proposed [29][30][40]. Whether the application for a UUV is finding potential drilling locations for an oil platform or searching for mines, successfully creating missions (either for a single UUV or a group of UUVs) poses many challenging problems. There are two critical issues in this context. First, attempting to create a mission free from syntactic errors as well as semantic errors and second, if multiple UUVs are used for a mission, one has to properly split these missions into a set of sub-missions and assign them in an optimal way to each vehicle. Both of these factors play a key role in a given mission's success.

Fortunately, tools have been created recently to aid in the resolution of some of these issues. Such tools include mission programming languages (MPLs) [7][12][13] with compiler support that allow an operator to create missions easily (similar to that of general purpose high level languages such as C++ and Java) as well as utilities that generate parallel sub-missions from a given high level sequential mission description [10][11][33]. Even though these tools were developed to make mission creation and maintenance easier rather than using a low level language or writing a set of parallel missions by hand, there are still many potential problems that have to be addressed. For example, if an operator writes a mission using an MPL, the potential still exists for the operator to make mistakes the same as with any programmer writing

source code using a generic high level language. While a compiler typically catches some of these errors (e.g., those related to the syntactic structure of the program), semantic errors are hard to catch and fix at compile time. In addition, when the mission description written in an MPL is to be read by someone who is not the author of that mission text, it may be difficult to understand easily what the mission is supposed to do. As a result, mission text maintenance and update can be very problematic. Last, an MPL is typically specific to a UUV and does not port to other UUVs at all, or requires extensive effort on the programmer's side to port it. While there already exist several attempts at developing a universal MPL [14][18][23][24] that can execute regarding different vehicle types, this is not expected to happen very soon in practice. Even if this is realized someday, such an MPL has to be extended periodically to keep up with new vehicle capabilities and emerging mission requirements.

Therefore, there is a clear need for developing more user-intuitive programming environments for writing and visualizing missions that require multiple, cooperating UUVs. This work is an attempt at this direction, and proposes a programming-free parallel mission generation through a graphical tool. Our tool has five major components. The first component uses a maritime navigational package called Nobeltec [8] and allows the operator to graphically specify the high level mission very easily without indicating explicitly any low level details about the parallelization and workload distribution. The second component provides the ability for an operator to specify additional mission parameters using another graphical interface, which eliminates any source code the operator needs to write for the mission. The third component translates this mission from the graphical description to a mission programming language (MPL) version. The forth component is an automatic mission parallelizer, which takes the MPL description of the mission and parallelizes it across the specified set of UUVs without operator involvement. This parallelization is transparent to the operator and can be targeted at different objective functions such as minimizing total mission latency or reducing total number of vehicles used for a mission. The fifth component of our tool maps these parallel sub-missions back to the Nobeltec representation, again in an operator-transparent manner.

The combined effect of all these components is that the operator defines his/her mission requirements and available resources (UUVs and their sensors) using a graphical interface and the specified mission is automatically parallelized by the tool and displays the resulting set of sub-mission of this parallelization process on screen for the operator to review. The main contribution of this tool is that it relieves the operator from low-level mission programming using an MPL (not to mention all the potential problems that come with it) and the manual partitioning of the mission across the available vehicles. Therefore, this approach does not only eliminate the need for programming but also allows the operator to visualize his/her parallelized mission using our graphical interface.

The rest of this paper is organized as follows. Section II presents background information describing the typical steps involved with a UUV mission. Section III discusses our existing tools including our high-level mission programming language, its corresponding compiler, and a mission splitting utility that includes three different mission splitting strategies. Section IV introduces our new graphical mission specification and partitioning utility for creating a set of parallel sub-missions for a group of UUVs and discusses the details of each of the five components of our tool. Section V offers an experimental evaluation of our tool by illustrating how the different mission splitting strategies are used to allocate vehicles to specific missions and Section VI discusses our future work followed by our concluding remarks.

II. BACKGROUND

UUVs come in many different sizes ranging from a half meter in length up to thirty or more feet. The Seahorse UUV (shown in Figure 1) was designed and built by the Applied Research Lab at the Pennsylvania State University (ARL/PSU) for research purposes including the collection of high quality environmental data in the littoral regions of the world [2]. A typically scenario involving the preparation for a UUV mission includes an operator creating a mission for a particular vehicle, loading the mission onto the vehicle after it has been written, and deploying the vehicle from an oceanic vessel (e.g., T-AGS 60 class oceanographic vessel) in order to execute its mission.

The creation of the mission usually involves an operator typing the mission by hand in the same manner as a programmer would write source code. The mission for a UUV can either be written in the vehicles native programming language or a high-level vehicle language such as the high-level MPL developed for the Seahorse UUV (we will discuss our MPL in more detail in Section III [1]). After the mission is written, the mission is loaded onto the vehicle's onboard control system (e.g., a computer running some type of mission control software [19][21]). Typically these vehicles are equipped with a wireless device such as an 802.11 network card, which allows an operator to transmit the mission to the vehicle's onboard control system via wireless transmission.

When commanded to, the UUV submerges and begins to execute its mission. The execution time of the mission is dependent upon the type of mission. A mission can take anywhere from a few hours to more than a day. A

Figure 1 – The Seahorse UUV courtesy of the Applied Research Lab at The Pennsylvania State University.

hydrographic survey as discussed in [3] may take 8 hours or less to conduct and other mission types such as law enforcement missions or surveillance missions [17] may require UUVs to execute their missions 24/7 with periodic recharging of their onboard power systems. In addition to the nature of the mission, the type of vehicle used also plays an important role in determining the length of the mission. Smaller UUVs may only have enough battery power to last a few hours whereas larger UUVs may have the staying power to last several days. The Seahorse UUV actually has enough battery power to last approximately 100 hours at a rate of 4 knots.

A UUV is usually equipped with one or more sensors that are used to collect data during mission execution. Two different sensor types include side-scan sonars (SSS) and sub-bottom profilers. A SSS is used to map underwater terrain features and sometimes used to search for sunken wreckage (e.g., ships and downed aircraft) that may lie on the ocean floor. A SSS projects a beam from both sides of the UUV and scans the ocean floor as it traverses over a particular region. In the case of the Seahorse UUV, its onboard SSS can scan an area of 100 meters on each side of the vehicle. In other words, the UUV traces out a path 200 meters wide as it moves over the ocean floor. The data collected from a SSS can be reconstructed to provide 2D (and sometimes 3D) images of different features found on the ocean floor. A sub-bottom profiler, on the other hand, is usually located on the bottom of the vehicle and scans for subsurface features, that is, records images of structures and contents below the ocean floor. This data can be reconstructed to form 2D images of these subsurface structures. For more detail regarding SSS and sub-bottom profiler sensors, please refer to [3].

After the mission is finished executing the UUV comes to the surface for retrieval by the oceanic vessel. The larger UUVs are typically fished out, that is, when they surface, a cable is attached to the nose of the vehicle and it is reeled in by a small crane. Smaller vehicles may simply be retrieved by hand. Once the vehicle is retrieved the data it gathered while on its mission can be downloaded and analyzed. In the next section, the existing tools developed at ARL/PSU are discussed that includes the high-level MPL, the supporting compiler, and a mission partitioning utility that automatically generates missions for groups of cooperating vehicles.

III. Existing Tools

This section discusses some existing tools including a high-level MPL, a supporting compiler for this MPL, and a tool that offers three different strategies for splitting an operator specified mission into a set of parallel sub-missions.

A. Our High-Level Mission Programming Language

We developed a high-level Mission Programming Language (MPL) [1] at the Applied Research Lab at the Pennsylvania State University for use with the Seahorse UUV [2] to allow an operator to easily specify missions without having to worry about any details for a particular UUV (similar to a programmer writing a C++ program without having to worry about any of the underlying hardware details for a particular machine's architecture). Our high-level MPL has different types of orders that can be used to specify a mission. Some of these orders include Waypoint Navigation Orders (WNOs) and Survey Orders (SOs). Examples of these orders are shown in a sample mission for the Seahorse UUV (Figure 2). Here the WNO is used to specify a destination latitude and longitude from the UUV's current position. The SO is used to specify an area of the ocean floor to be surveyed or scanned. A SO could be used for different applications such as mapping the ocean floor, mine detection and identification, and exploring potential drilling locations for the oil industry. Next we discuss the supporting compiler for this high-level MPL.

B. Compiler Support for our Mission Programming Language

Each order type supported by our high-level MPL has a corresponding language specification. The language specification for each order type is a rule on how to use each particular order type. The language specifications for the WNO type and SO type are shown in Figure 3. This language has critical elements as well as optional elements. The operator must specify critical elements whereas optional elements may be omitted. Note that if an optional element is omitted then a default value is substituted for this element. Default values are shown in parenthesis. For example, with respect to the WNO, if the *Destination Latitude* and *Destination Longitude* are omitted, the UUV will have no way of knowing where to travel to from its current position. The MPL cannot simply assume a value for these elements. If it does assume a value, the default values for the *Destination Latitude* and *Destination Longitude* could be a position far outside of the operating area for the UUV. On the other hand, if the operator does not specify the optional element *Use_SSS* (which indicates whether to use the SSS), the MPL can assume this is to be used since there are only two options for this element, True and False.

```
{
 Start                    : Waypoint_Navigation_Order
 Destination_Latitude     : 40.0000 Degrees
 Destination_Longitude    : -74.0000 Degrees
 Transit_Mode             : Steer_to_Point
 Transit_Depth            : 10 Meters
 Transit_Speed_In_Water   : 5.0 Knots
 Use_SSS                  : False

 Start                    : Survey_Order
 TopLeft_Latitude         : 40.0000 Degrees
 TopLeft_Longitude        : -74.0000 Degrees
 TopRight_Latitude        : 40.0000 Degrees
 TopRight_Longitude       : -73.9500 Degrees
 BottomLeft_Latitude      : 39.9500 Degrees
 BottomLeft_Longitude     : -74.0000 Degrees
 BottomRight_Latitude     : -39.9500 Degrees
 BottomRight_Longitude    : -73.9500 Degrees
 Survey_Depth             : 30 Meters
 Survey_Speed             : 3.5 Knots
 StartPoint               : TopLeft
}
```

Figure 2 – A sample mission involving a WNO and a SO. A WNO commands a UUV to travel to a destination latitude and longitude. A SO commands a UUV to survey a region of the ocean floor defined by four sets of latitude and longitude coordinates that traces out a rectangular region.

```
Start                   Waypoint_Navigation_Order
Destination_Latitude    Critical Float Deg/Rad
Destination_Longitude   Critical Float Deg/Rad
Transit_Mode            Critical Integer Steer_to_Line/Steer_to_Point
Transit_Depth           Critical Float Meters/Feet/Yards
Transit_Speed_In_Water  Critical Float mps/Knots
Use_SSS                 Optional Boolean True/False(True)
Do_VBSTrim              Optional Boolean True/False(True)

Start                   Survey_Order
TopLeft_Latitude        Critical Float Deg/Rad
TopLeft_Longitude       Critical Float Deg/Rad
TopRight_Latitude       Critical Float Deg/Rad
TopRight_Longitude      Critical Float Deg/Rad
BottomLeft_Latitude     Critical Float Deg/Rad
BottomLeft_Longitude    Critical Float Deg/Rad
BottomRight_Latitude    Critical Float Deg/Rad
BottomRight_Longitude   Critical Float Deg/Rad
Survey_Depth            Critical Float Meters/Feet/Yards
Survey_Speed            Critical Float Mps/Knots
```

Figure 3 – Sample of the language specifications for both WNOs and SOs. Note that each order type has many different order elements such as the *Transit_Depth* element for WNOs that indicates the depth of the vehicle as it travels to its destination and the element *Survey_Speed* for SOs that specifies the speed of the vehicle as it conducts a survey of an area.

To develop the compiler for this high-level MPL, we derived a grammar for the different order types based on the corresponding order type specifications. Once we had the grammar representation for each order type, source code for our compiler was generated using the tools Lex (a lexical analyzer) and YACC (Yet Another Compiler Compiler) [25] that is based on our grammar and set of regular expressions (REs) [26] (REs are used by Lex to identify certain patterns within text strings). An example of the grammar for the WNO is shown in Figure 4. The grammars used in our compiler are in the style of Backus-Naur Form or BNF [6]. Note that some productions in this grammar have a <*no token*> token and other productions do not have this token. The <*no token*> token is used to represent the optional order elements as in the production for *Use_SSS*. Other productions that do not include this <*no token*> token are the critical elements. For a detailed discussion regarding our high-level MPL and its corresponding compiler, we refer the reader to [1].

C. A UUV Mission Partitioning Utility

Since research has been moving towards using multiple UUVs in recent years, we needed a way to add support for this capability to our existing high-level MPL. We added a feature (called the *parallel region* construct) that allows an operator to easily express complex missions involving multiple UUVs. An example of this construct is shown in Figure 5 (Note that this construct is similar to compiler directives used in the C programming language [31]).

```
WNO             -> waypoint_start dest_lat dest_long transit_depth
                   transit_speed transit_mode use_sss do_vbs_trim
waypoint_start  -> Start : Waypoint_Navigation_Order
dest_lat        -> Destination_Latitude : float length_dr
dest_long       -> Destination_Longitude : float length_dr
transit_depth   -> Transit_Depth : float length_mfy
transit_speed   -> Transit_Speed_In_Water float weight_mkk
transit_mode    -> Transit_Mode : steer_type | <No Token>
use_sss         -> Use_SSS : Boolean | <no token>
do_vbs_trim     -> Do_VBSTrim : Boolean | <no token>
steer_type      -> Steer_to_Point | Steer_to_Line
weight_mkk      -> mps| Knots
length_mfy      -> Meters | Feet | Yards
length_dr       -> Deg | Rad
```

Figure 4 – The corresponding grammar for a WNO based on its corresponding language specification from Figure 3. This grammar is in the style of Backus-Naur form (BNF) and is used with the tools Lex and YACC to generate source code for our MPL compiler.

This construct allows an operator to quickly and easily convert operator specified missions for a single UUV into missions involving multiple UUV's by simply enclosing the preferred orders in the parallel region construct. If an operator wants to convert this mission back into a mission for a single UUV, the operator simply removes the parallel region construct. The compiler has access to the available UUVs when the mission is compiled and, based on the number and types of UUVs, the compiler can generate the corresponding set of parallel sub-missions from the operator specified mission.

Our mission partitioning utility contains three different mission splitting strategies, a simple mission splitting approach, an approach to reduce the number of vehicles for a particular mission, and an approach that reduces the total mission execution time or latency. Note that these three strategies assume each vehicle has enough battery capacity to traverse their assigned region of a survey area. Under future work we discuss expanding these strategies to also consider the available battery capacity of each vehicle as a constraint when determining which vehicles are capable of traversing particular regions of a given survey area.

The simple mission splitting approach, strategy one, is as follows. Given an operator specified mission and a set of *n* vehicles (each one with the same capabilities) this simple mission splitting algorithm divides the mission into *n* sub missions. For example, when the parallel mission from Figure 5 is compiled and we have four UUVs available, our mission splitting algorithm will create a set of four sub-missions depicted graphically in Figure 6. Note that this mission splitting algorithm is very

```
...
Parallel_Region_Start
{
  Start                  : Survey_Order
  TopLeft_Latitude       : 40.0000 Degrees
  TopLeft_Longitude      : -74.0000 Degrees
  TopRight_Latitude      : 40.0000 Degrees
  TopRight_Longitude     : -73.9500 Degrees
  BottomLeft_Latitude    : 39.9500 Degrees
  BottomLeft_Longitude   : -74.0000 Degrees
  BottomRight_Latitude   : -39.9500 Degrees
  BottomRight_Longitude  : -73.9500 Degrees
  Survey_Depth           : 30 Meters
  Survey_Speed           : 3.5 Knots
  Start Point            : TopLeft
}
Parallel_Region_End
...
```

Figure 5 – An example mission using the *Parallel Region* construct. Orders contained inside of this construct (such as this survey order) will have a set of parallel sub-missions automatically generated when this construct is encountered by the MPL compiler. The compiler has access to vehicle data including the vehicle types and number of each type that will be available at the time of mission execution.

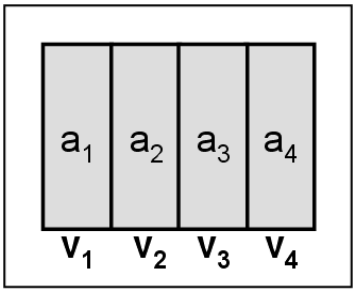

Figure 6 – The set of parallel sub-missions generated from the mission in Figure 5.

simple and assumes all available vehicles have the same capability (i.e., equipped with the same set of sensors and have the same speed). One practical application for this algorithm would be mine detection using a set of identical vehicles for a specified region of the ocean floor.

The second strategy attempts to reduce the number of vehicles for a particular mission. This strategy can work for vehicles that are heterogeneous, that is, vehicles that have different sensor capabilities. For the purposes of explanation we assume the vehicles are equipped with generic sensors s_1, through s_4. The algorithm used for this strategy is a greedy approach that uses an adaptation of the classic set-covering algorithm that can be used to model resource selection problems [32] and is shown in Figure 7.

To briefly describe this algorithm, *X* represents the set of regions contained within the survey area, each one requiring a specific set of sensor readings. Set *V* represents the set of available vehicles that can be used to survey the different regions. Each vehicle in *V* is equipped with one or more sensors (e.g., sensors from the set s_1 , ... , s_4). The regions contained in set *X* must be traversed by one or more of the vehicles contained in set *V* that are equipped with the sensors needed by the particular regions. The idea here is to find the minimal set of vehicles that will satisfy (or "cover") the sensor requirements of each region. Therefore, at a particular instance when a vehicle is about to be chosen (line 4) the next vehicle that can cover the largest subset of regions that are currently not covered (i.e., do not have all of their sensor requirements satisfied by a previously selected vehicle) is selected. It is this line that makes this algorithm a greedy algorithm.

The third strategy (Figure 8) attempts to reduce the total mission latency given the number of available vehicles for the mission and uses a greedy approach similar to the set covering algorithm. To briefly describe

```
GREEDY-SET-COVER ( X, V )
1  U ← X
2  C ← Ø
3  while U ≠ Ø do
4      select v_j ∈ V that maximizes | v_j ∩ U |
5      C ← C ∪ v_j
6      U ← U ∩ v_j
7  end while
8  return C
```

Figure 7 – Setcovering algorithm used to reduce the number of UUV for a particular mission.

```
REDUCE-SURVEY-TIME
1  SA ← get_survey_areas ()        // Provided from user mission
2  VT ← get_vehicle_types ()       // Retrieved from data source
3  Vehicle_List.Types = allocate_vehicle_types ( SA, VT )
4  sort_satisfy_list (Vehicle_List ) // Largest to smallest
5  for j = 1 to size_of ( SA )
6    for i = 1 to size_of ( Vehicle_List_j )
7      num_slices ← get_vehicle_count_for_subregion( Vehicle_List_j.Type_i )
8      slice_width ← calc_width_of_subregion ( SA_j )
9      create num_slices of parallel sub missions for SA_j using slice_width
10   next i
11 next j
```

Figure 8 – The greedy algorithm to reduce the total mission latency for a group of UUVs.

this algorithm, first the vehicle data is retrieved along with the survey area information (lines 1 and 2). Next, the types of vehicles that will best satisfy each region of the survey area is determined, and then this vehicle list is sorted from the largest group to the smallest group (lines 3 and 4). Lines 5 through 11 then iterate through the different survey regions and the vehicle list. During each iteration of the loop structure the next largest group of vehicles is selected and assigned to the next available region of the survey area. It is this behavior that makes this algorithm a greedy approach. We encourage the reader to read [11] for a detailed explanation of all three strategies.

IV. The Graphical Mission Specification and Partitioning Tool

This section discusses our graphical mission specification and partitioning tool and all of the sub components that were integrated to realize this package. We remark on the function of each sub component and explain how it interacts with the rest of the system.

A. The Nobeltec Visual Navigation Suite Package

To create our graphical mission specification and partitioning (GMSP) tool, we integrated a set of independent tools to work together. First, we used the package called Nobeltec's Visual Navigation Suite (VNS) [8] by Jeppesen Marine, which is a commercially available package to aid in the navigation of sea vessels. A screenshot of this package is shown in Figure 9. This package provides an operator with a graphical user interface (GUI) allowing the operator to view various locations in the ocean and any nearby coastal regions.

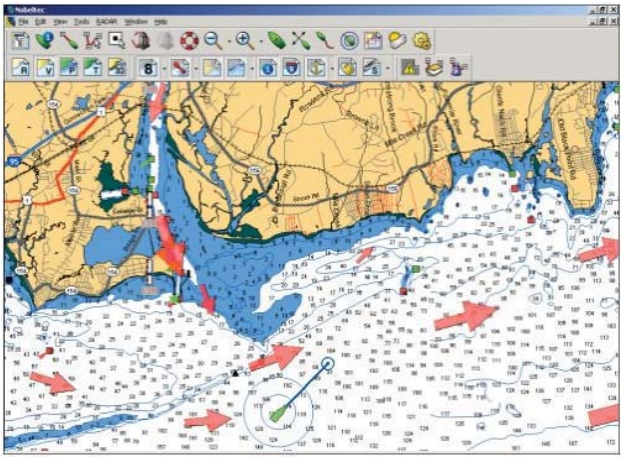

Figure 9 – Screenshot of Nobeltec's Visual Navigation Suite maritime software package.

This tool also lets the operator specify waypoints to visit and areas to explore by using a simple point and click interface. An operator can easily specify a series of waypoints and areas of interest to quickly create a mission for a sea vessel.

We decided to leverage the power of this commercial package for use in creating missions for UUV's. More specifically we wanted a tool that an operator can use to specify one or more survey areas for a group of available UUVs. The idea was to let the operator specify regions that need to be surveyed without worrying about how the survey areas were to be split up among the group of UUVs. Figure 10 shows a close-up of an operator specified mission. Here there is a single SO represented by the rectangle and two WNOs. One WNO order goes from the vessel that will launch the UUVs to the upper left corner of the survey area and the second WNO goes from the lower left corner of the survey area back to the vessel on the left hand side of the screenshot. As the operator specified the mission graphically, the operator places a series of what are known as *Marks* in the VNS. These *Marks* can be used to define a WNO (e.g., two *Marks* connected by a line) or a SO (e.g., Three or more *Marks* that define an area such as a rectangle). Once the operator represents the orders of a mission as a series of *Marks* in the VNS, this mission can then be exported from the VNS to a text file represented using the Open Navigation Format (ONF) [9], which is based on the INI-style ASCII text file format shown in Figure 11. The next section discusses how MPL order elements are added to the ONF representation of the mission using a graphical interface.

B. *Graphically Adding MPL Order Elements*

After the graphical representation of this mission is exported to the ONF representation, the next step involves adding MPL order elements to this ONF representation via another graphical interface. Before discussing the addition of these MPL elements using this graphical interface, note that some data pertaining to the mission such as the *Transit Depth* element for WNOs cannot be directly represented in the Nobeltec VNS package (i.e., there is no concept for this type of data in Nobeltec). Fortunately, the Nobeltec package provides a description field for each *Mark* the operator places on the screen. We use this description field to store MPL specific data such as the *Transit_Depth* and *Use_SSS* elements for WNOs and the *Survey_Depth* and *Survey_Speed* elements used in SOs. In a previous version of our tool, the operator entered these MPL order elements in the description field by hand [35]. A screenshot of this description field along with MPL elements for a SO entered manually by an operator is shown in Figure 12.

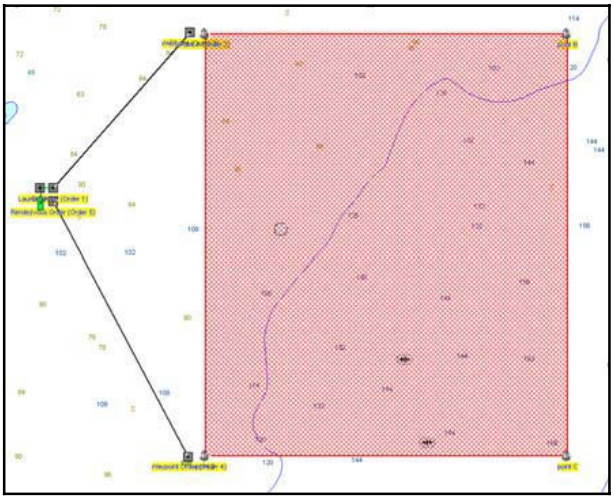

Figure 10 – Screenshot of an operator specified SO in the Nobeltec Visual Navigation Suite maritime software package. This operator specified mission shows a survey area as a rectangular region and two waypoints. One waypoint goes from the vessel on the left hand side that will launch the UUVs to the upper left corner of the survey area. The second waypoint goes from the bottom left corner of the survey area back to the vessel after the mission completes.

```
[++41051bcc-b92e-4f7a-8ade-c4da905ffdac++]
Name = Point A (Order 3)
Type = Mark
CreateTime = 2007-03-09 18:17:30Z
Hidden = FALSE
Locked = FALSE
Desc = {{
(Order 3)
start_order:                Survey_Order
Scheduling_Info_Is_Timed:   True
Survey_Depth:               100 Meters
Survey_Speed:               3.5 Knots
Do_VBSTrim:                 True
SwathWidth:                 200.0 Meters
Overlap:                    0.0
StartPoint:                 TopLeft
}}
Color = 0x000000
LatLon = 37 58.00000 N 074 49.00000 W
```

Figure 11 – A sample of the ONF representation for a SO. Note that the MPL order elements are part of the description field for this particular Mark. This is the result after the GUI used from Figure 13 writes the MPL order elements back the ONF file.

Even though the operator was able to create part of the mission graphically, the operator still had to enter these MPL elements by hand, which allowed for the possibility of syntax errors. Anytime a person writes any type of source code by hand, the potential always exists for errors to be introduced into the source code. This is the main reason for creating an additional graphical interface (Figure 13) to allow an operator to add these MPL order elements graphically rather than typing them by hand. This graphical interface reads the ONF representation of the mission after it is exported and allows an operator to specify MPL elements in a point and click fashion using

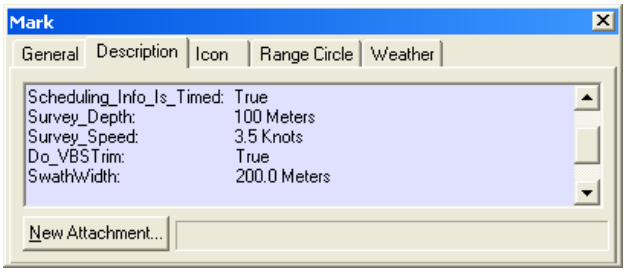

Figure 12 – The description field for a particular mark displaying the MPL order elements entered manually by an operator. This method was required by the previous version of the GMSP utility. In the current version these elements are added using the GUI tool from Figure 13

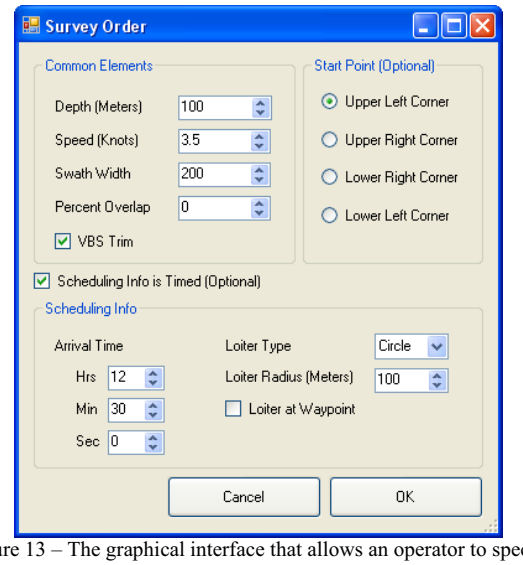

Figure 13 – The graphical interface that allows an operator to specify MPL order elements in a point and click fashion. The method of MPL specification replaces the old method shown in Figure 12, thus further reducing potential errors. Once all MPL order elements are specified, this data is written back to the file containing the ONF representation.

common controls such as check boxes, radio buttons, and spin controls as shown here in Figure 13.

For this particular example, the operator is presented with a way to specify all of the MPL order elements for a SO. A few of the MPL SO elements an operator can configure are the depth and speed of the UUV during the survey, the starting point of the UUV before it traverses the area, and scheduling information such as the arrival time and whether or not the UUV is to loiter after the survey of the area is finished. After the operator finishes specifying the MPL order elements, these elements are written to the description fields of the corresponding orders in the ONF representation. An example of the ONF file with the MPL elements added to the description field is shown in Figure 11. Here the description field for a SO in the ONF representation contains the textual representation of the MPL order elements specified by the operator using the graphical interface from Figure 13. Once all of the MPL elements are written to their respective orders in the ONF file, this file is now ready to be converted to the corresponding MPL representation of the mission.

C. *Converting from ONF to MPL*

After the ONF representation has all of the MPL order elements added to the various orders, it is ready for processing by the Nobeltec to MPL (NT2MPL) translation component. This component reads the operator specified mission represented in the ONF format and converts the mission orders to the corresponding order types in the MPL format. The translation is a simple one-to-one translation. For example, a WNO represented by two *Marks* in the Nobeltec package is simply translated to a WNO in the corresponding MPL format. A *Mark* in the VNS includes the destination latitude and longitude and other data for the display of this *mark* in the Nobeltec environment. When the NT2MPL component translates a particular mission order from the ONF format to the MPL format, specific data relating to the mission order (e.g., a WNO) stored in the description field is added to the corresponding MPL order type. In short, this ONF format captures all of the mission details in a text file similar to the way configuration settings are saved in the INI-style ASCII text file format used by many Windows applications. Once the ONF representation of the operator specified mission is converted to its MPL equivalent, the mission is ready to be split into its corresponding set of parallel sub-missions. We discuss the mission splitting component in the next sub section.

D. *Generating the Set of Parallel Sub-Missions*

After the mission represented in the ONF format has been converted into the corresponding MPL format, the mission is ready to be split into a set of parallel sub-missions. The mission splitting component or Mission Parallelizer (MP) reads the MPL representation of the operator specified mission and begins the mission splitting process. For the purposes of this example, we choose to use strategy one form Section III, the simple mission splitting algorithm. In the next section we present examples that illustrate the use of strategies two and three on a particular set of missions. Since we are using a very simple algorithm, we simply read the dimensions of the entire survey area and divide it (as explained in Section III.C) into n equal sub areas, where n represents the number of vehicles available for the mission. Each sub-mission is then written to a separate file in the MPL format. Each file can then be loaded onto the mission controller of its corresponding UUV when the mission is to be executed as discussed in [22]. Once the set of parallel missions is generated from the operator specified mission, these missions are ready for converting back into the ONF representation so they can be loaded back in the Nobeltec VNS package to be viewed by the operator.

E. *Converting from MPL back to ONF*

After the operator specified mission is split into its corresponding set of sub-missions, this set of sub-missions must be converted back into the ONF representation to be imported back into the Nobeltec VNS package. Again this translation is a one-to-one translation process. The entire set of sub-mission in the MPL format are read by the MPL to Nobeltec (MPL2NT) translation component and translates the contents of each sub-mission file back into its corresponding ONF representation. All sub-missions are then written to one Nobeltec file in the ONF format. Once this file is created it can then be imported back into Nobeltec for review by the operator. Figure 14 shows the corresponding set of generated sub-missions based on the operator specified mission in Nobeltec VNS as depicted in Figure 10. Here the operator specified mission was split into a set of four sub-missions and placed over the original operator specified mission.

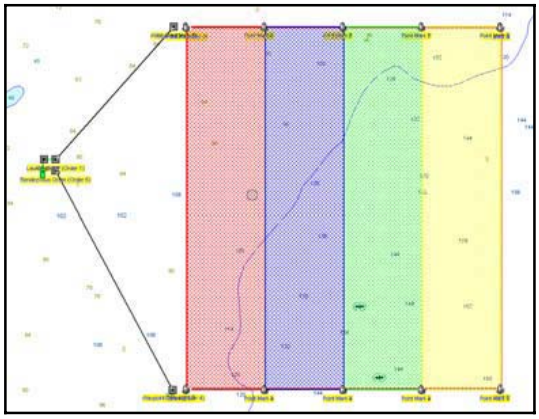

Figure 14 – Screenshot of the set of generated sub-missions after being imported back into the Nobeltec Visual Navigation Suite maritime software package. Note how four different sub-missions are shown based on the original mission from Figure 10

F. *Bringing it all Together, an Operator View*

We have presented all five sub-components of our graphical mission specification and partitioning tool. We now want to give a complete summary of the different steps for this tool from an operator's standpoint. Figure 15 shows a diagram of the entire process of generating a set of parallel sub-mission from an operator specified mission using our graphical mission specification and partitioning tool. Note that our five sub-components are depicted by gray boxes. Here, step 1 involves the operator specifying the mission graphically using the Nobeltec VNS. After the graphical specification of the mission is completed, the mission is exported from the Nobeltec VNS to a text file containing the ONF representation of this mission (Step 2). Step 3 involves the operator using the MPL Tool to graphically specify all the MPL order elements for the mission. Once the operator finishes this task, the MPL order elements are written back to the text file containing the ONF representation. The operator then invokes the compiler (a command line utility) where this ONF representation of the mission is processed by the compiler (Steps 4, 5, and 6). Step 4 converts the ONF representation to the corresponding MPL format using the NT2MPL conversion utility. Step 5 generates the corresponding set of parallel sub-missions in the MPL format. These missions are the ones to be loaded onto the actual vehicles as mentioned before and is discussed in [1]. Step 6 converts the MPL version of the sub-missions back the ONF representation using the MPL2NT conversion utility. Note that steps 4, 5, and 6 all happen automatically. Step 7 involves the operator importing the set of newly generated sub-missions back into the Nobeltec VNS for review. From an operator's standpoint, the only manual steps are step 1 (exporting the mission from Nobeltec) step 2 (launching the MPL Tool), step 4 (invoking the compiler to generate the corresponding set of parallel sub-missions), and step 7 (importing the ONF representation of the set of parallel sub-missions back into Nobeltec for inspection). Steps 4, 5, and 6 are contained in the compiler and are transparent to the operator.

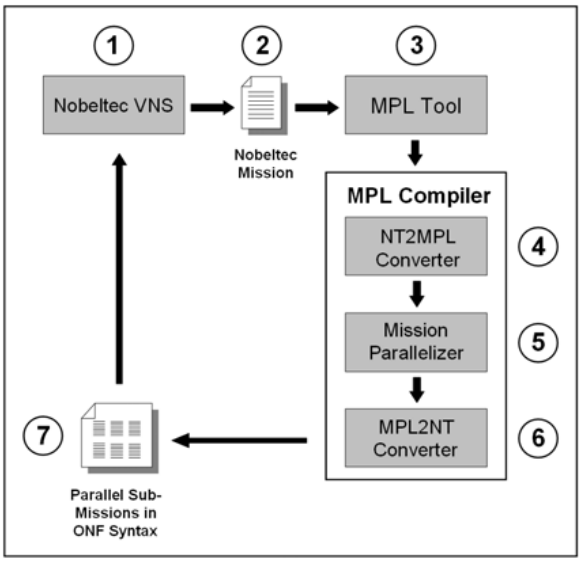

Figure 15 – High-level view regarding all of the steps involved when using the Graphical Mission Specification and Partitioning Tool. Step 1, the operator specifies the mission graphically and exports the mission as ONF representation (step 2). Step 3, the operator adds MPL elements via MPL GUI. The operator invokes the compiler and the ONF mission is converted to MPL, split into set of parallel sub-missions, and sub-missions are converted back to ONF (steps 4-6 respectively). Step 7, operator loads ONF sub-missions into the VNS for review.

V. Experimental Evaluation of Scenarios

This section illustrates how mission splitting strategies two and three from Section III are used by the GMSP utility to allocate UUVs to different examples of missions. First we examine how strategy two (reducing the number of vehicles) is used by the GMSP utility for two different missions. We then examine how the GMSP utility uses strategy three (reducing total mission latency) to allocate UUVs for the same two missions.

A. *Reducing the total Number of Vehicles for a Given Set of Missions*

For this strategy we consider two different missions and for each mission we will use two different groups of vehicles to show how our GMSP tool works under different conditions. The two groups of vehicles used for this strategy are shown in Table 1. The first group (Group A) contains eight vehicles total (four different vehicle types and two vehicles in each type). Each vehicle type in this group (Type 1 through Type 4) is equipped with a unique sensor (s_1 through s_4 respectively). The second group of vehicles (Group B) also contains eight vehicles

TABLE 1
VEHICLES USED FOR THE EXAMPLE MISSIONS

Group A

Vehicle Type	Quantity	Equipped Sensors
Type 1	2	s_1
Type 2	2	s_2
Type 3	2	s_3
Type 4	2	s_4

Group B

Vehicle Type	Quantity	Equipped Sensors
Type 5	4	s_1, s_2, s_3
Type 6	4	s_3, s_4

(two different vehicle types and four vehicles in each type). The first vehicle type in this group (Type 5) contains three sensors (s_1, s_2, and s_3). The second vehicle type in this group (Type 6) is equipped with two sensors (s_3 and s_4).

The first mission used for this strategy is shown in Figure 16. For this mission, there are four separate regions where each region requires a different sensor type. The top left region requires sensor s_1,, the top right region requires sensor s_2, the bottom right region requires sensor s_3, and the bottom left region requires sensor s_4. Note that for the purpose of clarity we added the specific sensor labels to the regions in Figure 16. These labels do not actually appear on the screen when the operator specifies the mission.

Together these regions comprise the entire survey area. This example mission represents a typical mission that an operator creates using the Nobeltec VNS package. After the operator creates this mission, the mission is exported and the MPL elements are then added to the ONF representation (Step 3 from Figure 15). When the operator invokes the MPL compiler, the option to reduce the number of vehicles is selected. For this execution, vehicle Group A is assumed to be the set of available vehicles for this mission. After the compiler generates the set of parallel sub-missions, the ONF representation can be loaded back into the VNS package for review. The resulting set of parallel sub-missions generated by the MPL compiler for the operator specified mission in Figure 16 is shown in Figure 17.

Here a minimum of four vehicles is needed to conduct the survey. Since each vehicle has a unique sensor and since there are four different regions each requiring a different sensor type, one vehicle is needed from each vehicle type from the group to fulfill the sensor requirements of the entire survey area.

If we use vehicle Group B for this same mission, only two vehicles are needed (one vehicle from each of the two different vehicle types). The resulting set of parallel sub-missions generated by the MPL compiler for this group of vehicles is illustrated in Figure 18. Here a single vehicle of Type 5 (containing sensors s_1 , s_2, and s_3) is

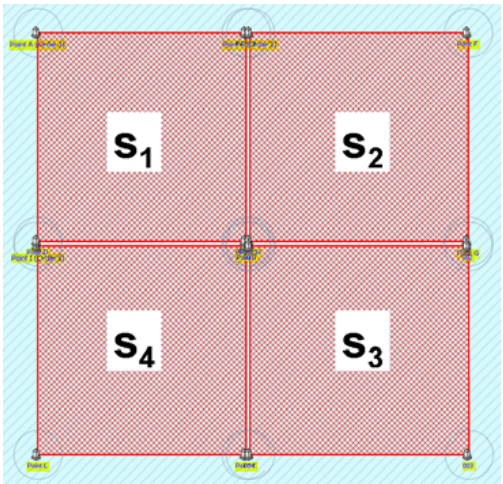

Figure 16 – Example Mission one that will be used for strategies two and three, reducing the number of vehicle and reducing total mission latency.

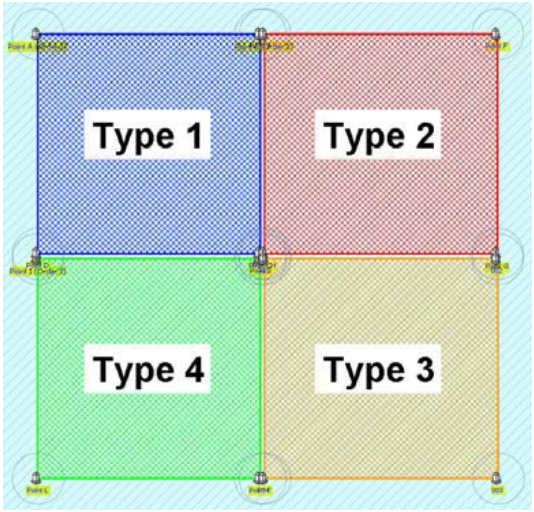

Figure 17 – The set of parallel sub-missions generated by the compiler for vehicle Group A with respect to reducing the total number of vehicles for the mission in Figure 16.

needed to conduct a survey of the upper two regions and the bottom right region. A single vehicle of Type 6 (containing sensor s_3 and s_4) is needed to conduct a survey of the lower left region. The vehicle from Type 5 will use all three of its sensors for its assigned regions. Sensor s_1 will be used for the upper left region, sensor s_2 will be used for the upper right region, and sensor s_3 will be used for the lower right region. Even though the vehicle from Type 6 is equipped with both sensors s_3 and s_4, this vehicle will only use sensor s_4 to survey the lower left region since this is the only sensor required for its assigned region. Note that if strategy three was used to reduce total mission latency, vehicles with multiple sensors will be utilized to aid in reducing the total execution time.

Next we consider both groups of vehicles for our second example mission shown in Figure 19. Here there are three different regions that comprise our survey area. The top left region requires sensor s_1, the top right region requires sensor s_2, and the entire bottom region requires both sensors s_3 and s_4. If we use the vehicles from Group A for this mission, a total of four vehicles are needed to satisfy the sensor requirements of the different regions in

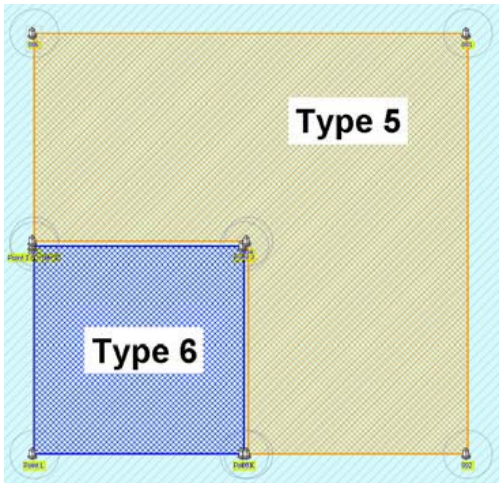

Figure 18 - The set of parallel sub-missions generated by the compiler for vehicle Group B with respect to reducing the total number of vehicles for the mission in Figure 16.

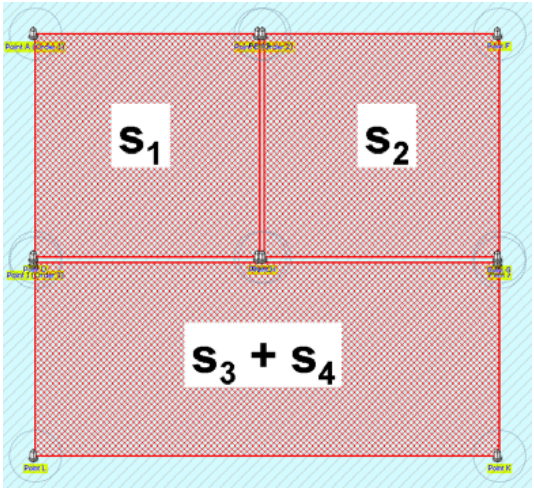

Figure 19 - Example Mission two that will be used for strategies two and three, reducing the number of vehicle and reducing total mission latency.

this survey area (the same as before when these vehicles were used for the mission in Figure 16). A vehicle of Type 1 is assigned to the upper left region (the region that requires sensor s_1); a vehicle of Type 2 is assigned to the upper right region (the region that requires sensor s_2) and one vehicle each from Type 3 and Type 4 is assigned to the bottom region (the region that requires sensors s_3 and s_4).

If we use the vehicles from Group B for this same mission, only two vehicles are required. One vehicle from Type 5 is used to survey all three regions since it is equipped with sensors s_1, s_2, and s_3 and all three regions require these sensor types. One vehicle is used from Type 6 since it is required to patrol the bottom region that requires sensor s_4. Note that even though the vehicle from Type 6 contains sensor s_3, this sensor is not used by this vehicle as it conducts a survey of the bottom region. The resulting set of parallel sub-missions for Group B is shown in Figure 20. Here the task of using sensor s_3 is given to the vehicle of Type 5 since this was the first vehicle assigned to the different regions. If we were to use strategy three (reducing mission latency) instead of strategy two for this mission, the vehicles of Type 6 would be used to satisfy the sensor requirements of the different regions in conjunction with the vehicles of Type 5 in order to reduce the total mission execution time as we will see later in this section.

B. Reducing the total Mission Latency for a Given Set of Missions

For this strategy (reducing the total mission latency) we consider the same two missions from Figure 16 and Figure 19. For each mission we will use the same two groups of vehicle listed in Table 1 (vehicle Group A and vehicle Group B will be used). There are a few additional pieces of information for this strategy. The size of the survey area in each example mission is 800 meters by 800 meters, that is, each survey area is comprised of different regions. Together these regions make a square survey area where the length and width are both 800 meters. Since we are trying to reduce total mission latency we need to take the speed of each vehicle into account. We assume that the vehicles in Group A and Group B all have the same rate of speed of 1 meter per second (3.6 Km/h).

First, vehicle Group A is used for example mission one from Figure 16. After running the option to reduce total mission latency, we get the output as shown in Figure 21 Note how all vehicles are utilized for this mission. Both vehicles of each type are used on a different region in order to satisfy the sensor requirements of that region, but to also help reduce the total mission latency. If one vehicle was equipped with all the sensors and was to traverse all regions of this entire survey area, it would take over an hour at a rate of 3.6 Km/h (This time includes the time it takes for the vehicle to turn as it surveys the area). Using all eight vehicles where each vehicle only traverses its assigned slice as depicted in Figure 21, the total mission time is only less than seven minutes. Note that this time is a theoretical value assuming that the UUVs are traveling in ideal conditions (i.e., no turbulence, drag, or currents that will impede the speed of the vehicle, not to mention the time it takes for each vehicle to reach its assigned region). We discuss improvements to our tool in Section VI regarding these

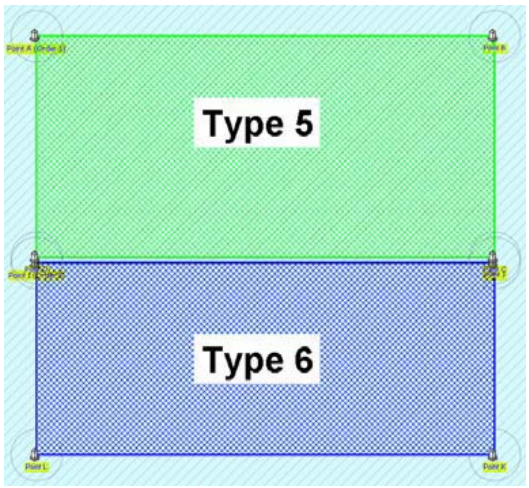

Figure 20 - The set of parallel sub-missions generated by the compiler for vehicle Group B with respect to reducing the total number of vehicles for the mission in Figure 19.

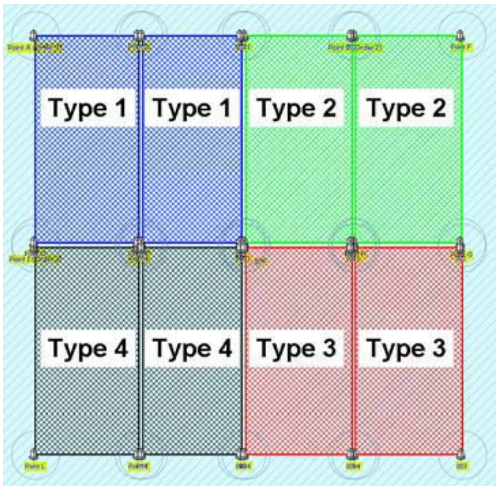

Figure 21 - The set of parallel sub-missions generated by the compiler for vehicle Group A with respect to reducing total mission latency for the mission in Figure 16.

issues.

Next we ran Group B for this same mission in Figure 16 and the results were identical to the results shown in Figure 21. The algorithm to reduce the total mission latency generated the same eight sub missions with one difference, the vehicle assignments. The top four slices were assigned to the vehicles of Type 5 and the bottom four slices were assigned to the vehicles of Type 6. When we ran Group B using example mission two from Figure 19, the results in this situation were also the same. Again there were eight slices and the vehicles of Type 5 were assigned to the top four slices and the vehicles of Type 6 were assigned to the bottom four slices. However, when we ran the vehicles of Group A against the mission from Figure 19, we did have different results than the previous three runs. The set of parallel sub-missions are shown in Figure 22. Here vehicles of Type 1 and Type 2 are assigned to the top regions the same as with the previous runs for this mission, but the vehicles of Type 3 and Type 4 have a different assignment. Since there are only two vehicles of Type 3 (equipped with sensor s_3) and two vehicles of Type 4 (equipped with sensor s_4), both vehicles of each type must traverse the entire width of the survey area in order to satisfy the sensor requirements of the bottom region. The path these vehicles take is the critical path and the total mission latency in this example is slightly less than 14 minutes. Again note that our method does not take the time needed for vehicles to reach their assigned regions. We will discuss this next under future work.

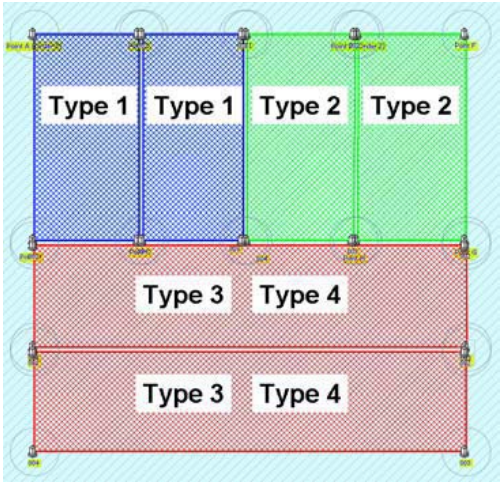

Figure 22 – The set of parallel sub-mission generated using the example mission from Figure 19 and the vehicle from Group A. Note that both vehicles of Type 3 and Type 4 are assigned to the bottom regions. Since there is only two of each of these vehicle types, each vehicle must traverse the entire width of the survey area.

VI. Concluding Remarks and Future Work

In summary, we have presented previous work regarding a high-level MPL, the supporting compiler, and three different strategies for generating a set of parallel sub-missions based on a operator specified mission for a group of cooperating UUVs. We also presented the integration of these different components with a commercially available package called Nobeltec VNS as the basis of our GMSP utility that allows the operator to create missions (both for a single UUV and multiple UUVs) using its graphical interfaces without manually writing any source code. The preliminary experiments involving this tool show much promise as a tool that can aid in the UUV mission creation and planning phases and remove from the operator much of the burden of creating missions for multiple UUVs. Even though our current tool shows much promise as a graphical mission-planning tool, many more improvements can be added to make this utility more versatile and practical.

Several tools already exist that allow operators to plan, rehearse, and replay missions, and even allow real-time monitoring of UUVs as they execute a mission [15][20][36][37]. We would like to extend this previous work by offering our own utilities in addition to these existing tools to automatically generate missions for multiple UUVs. Although our current mission-partitioning strategies may offer only a few objectives (e.g., reduce the total mission latency or reduce the total number of vehicles), more strategies can be added to our current repertoire to expand upon the number of available objectives. We recently developed other mission splitting algorithms that utilize integer linear programming (ILP) formulations that will minimize the number of vehicles, minimize the total mission latency [10], and even an ILP formulation to minimize the power consumed by a group of vehicles when they attempt to communicate data to a surface vessel or land based station [41]. Note that ILP formulations produce optimal results [34] when compared to the greedy approaches from Section III that may produce sub-optimal results in certain instances. We also plan to extend this work and create other objectives that will attempt to minimize the total power consumption of a group of vehicles while they conduct a mission. We have even devised dynamic mission re-planning methods that will reassign the tasks of a failed UUV to the remaining vehicles in a group to successfully complete the mission [38][39]. Since our graphical mission specification and partitioning tool is modular, we can simply add these additional mission-splitting strategies to our existing strategies. We may even be able to offer multi-objective optimization techniques for a particular mission. As we continue this work, our goal is to devise a whole series of mission splitting algorithms that will offer an array of different objectives for our GMSP utility and provide the operator with a complete graphical UUV mission-planning tool while relieving the operator from low-level mission programming tasks.

References

[1] G. Giger, L. Xue, S. Tangirala, and M. Kandemir, “High Level Mission Programming Support for Autonomous Underwater Vehicles,” *AUVSI’s Unmanned Systems North America*, Orlando, FL., 2006.

[2] Systems and Unmanned Vehicle, Applied Research Laboratory at the Pennsylvania State University, “Seahorse - Autonomous Underwater Vehicle (AUV),” 2006, http://www.arl.psu.edu/capabilities/at_suv.html

[3] R. Henthorn, D. W. Caress, H. Thomas, R. Mcewen, W. J. Kirkwood, C. K. Paull, and R. Keaten, “High-Resolution Multibeam and Subbottom Surveys of Submarine

Canyons, Deep-Sea Fan Channels, and Gas Seeps Using the MBARI Mapping AUV," *IEEE OCEANS 2006*, Boston MA., September 2006.

[4] L. Freitag, M. Grund, C. von Alt, R. Stokey, and T. Austin, "A Shallow Water Acoustic Network for Mine Countermeasures with Autonomous Underwater Vehicles," *IEEE OCEANS 2005*, Washington D.C., September 2005.

[5] D. Bingham, T. Drake, A. Hill, and R. Lott, "The Application of Autonomous Underwater Vehicle (AUV) Technology in the Oil Industry - Vision and Experience," *Proceedings of the International Federation of Surveyors' 22nd Congress*, Washington D.C., 2002.

[6] M. L. Scott, *Programming Language Pragmatics*, Academic Press, 2000.

[7] E. Eberbach, C. Duarte, C. Buzzell, and G. Martel, "A Portable Language for Control of Multiple Autonomous Vehicles and Distributed Problem Solving," *Proceedings of the 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems*, 2003.

[8] Products, Nobeltec, "Nobeltec Visual Navigation Suite," 2007, http://www.nobeltec.com/products/prod_suite.asp

[9] Open Navigation Format, Nobeltec, 2007, http://www.nobeltec.com/onf/ONFBody.htm

[10] G. Giger, M. Kandemir, S. D. Lovell, and J. Dzielski, "Automated Mission Parallelization for a Group of UUVs," *Proceedings of the 15th International Symposium on Unmanned Untethered Submersible Technology*, Durham, NH., August 2007.

[11] G. Giger, M. Kandemir, S. D. Lovell, J. Dzielski, and S. Tangirala, "Automated Mission Parallelization for Unmanned Underwater Vehicles," *AAAI Fall Symposium on Regarding the Intelligence in Distributed Intelligent Systems*, Washington D.C., November 2007.

[12] C. N. Duarte, C. Buzzell, G. R. Martel, D. Crimmins, R. Komerska, S. Mupparapu, S. Chappell, D. R. Blidberg, and R. Nitzel, "A Common Control Language to Support Multiple Cooperating AUVS," *Proceedings of the 14th International Symposium on Unmanned Untethered Submersible Technology*, Durham, NH., August 2005.

[13] R. L. Stokey, L. Freitag, and M. Grund, "A Compact Control Language for AUV Acoustic Communication," in *OCEANS 2005 - Europe*, June 2005.

[14] F. F. Ingrand, R. Chatila, R. Alami, F. Robert, "PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots," *Proceedings of the IEEE International Conference on Robotics and Automation*, April 1996.

[15] N. A. Anisimov, A. A. Kovalenko, G. V. Tarasov, A. V. Inzartsev, and A. Scherbatyuk, "A Graphical Environment for AUV Mission Programming and Verification," *Proceedings of the 10th International Symposium on Unmanned Untethered Submersible Technology*, Durham, NH., September 1997.

[16] J. E. Manley, "Multiple AUV missions in the National Oceanic and Atmospheric Administration," *Autonomous Underwater Vehicles, 2004 IEEE/OES*, June 2004.

[17] S. T. Tripp, "Autonomous Underwater Vehicles (AUVs): A Look at Coast Guard Needs to Close Performance Gaps and Enhance Current Mission Performance," Technical Report ADA450814, Coast Guard Research and Development Center, Groton, CT., 2006.

[18] A. Rajala, M. O'Rourke, and D. B. Edwards, "AUVish: An Application-Based Language for Cooperating AUVs," *IEEE OCEANS 2006*, September 2006.

[19] S. Chappell, S. Mupparapu, R. Komerska, and D. R. Blidberg, "SAUV II High Level Software Architecture," *14th International Symposium on Unmanned Untethered Submersible Technology*, Durham, NH., August 2005.

[20] S. S. Mupparapu, S. G. Chappell, R. J. Komerska, D. R. Blidberg, R. Nitzel, C. Benton, D. O. Popa, A. C. Sanderson, "Autonomous systems monitoring and control (ASMAC) - an AUV fleet controller," *Autonomous Underwater Vehicles, IEEE/OES*, June 2004.

[21] J. Borges Sousa, F. Lobo Pereira, and E. Pereira da Silva, "A Dynamically Configurable Architecture for the Control of Autonomous Underwater Vehicles," *IEEE/OES 1994 'Oceans Engineering for Today's Technology and Tomorrow's Preservation,'* September 1994.

[22] S. Tangirala, R. Kumar, S. Bhattacharyya, M. O'Connor, and L. E. Holloway, "Hybrid-Model based Hierarchical Mission Control Architecture for Autonomous Underwater Vehicles," *Proceedings of the 2005 American Control Conference*, Portland, OR., June 2005.

[23] D. Davis, "Automated Parsing and Conversion of Vehicle-Specific Data into Autonomous Vehicle Control Language (AVCL) Using Context-Free Grammars and XML Data Binding," *Proceedings of the 14th International Symposium on Unmanned Untethered Submersible Technology*, Durham, NH., August 2005.

[24] R. Blank, "A Structured Programming Approach For Complex AUV Mission Control," Masters Thesis, Naval Postgraduate School, Monterey, CA., September 1993.

[25] J. Levine, T. Mason, and D. Brown, *UNIX Programming Tools lex & yacc*, O'Reilly & Associates Inc., 1992.

[26] J. Friedl, *Mastering Regular Expressions*, O'Reilly & Associates Inc., 1998.

[27] M. Caccia, R. Bono, G. Bruzzone, and G. Veruggio, "Variable-Configuration UUVs for Marine Science Applications," *IEEE Robotics & Automation Magazine*, June 1999.

[28] G. Conte, S. Zanoli, A. Perdon, and A. Radicioni, "A system for the automatic survey of underwater structures," *IEEE OCEANS 1994 'Oceans Engineering for Today's Technology and Tomorrow's Preservation,'* September 1994.

[29] B. Bourgeois and P. McDowell, "UUV Teams for Deep Water Operations," Technical Report 430774, Naval Reseach Laboratory, Stennis Space Center, 2001.

[30] R. Wernli, "Low Cost UUV'S for Military Applications: Is the Technology Ready?," Space and Naval Warfare Systems Center, San Diego, CA., 2001.

[31] K. A. Barclay, *ANSI C Problem Solving and Programming*, Prentice Hall International, 1990.

[32] T. Corman, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, The MIT Press, 2001.

[33] G. Giger, M. Kandemir, and S. D. Lovell, "Automatic Generation of Parallel Missions for Autonomous Underwater Vehicles," Technical Report CSE 07-006, Department of Computer Science and Engineering, The Pennsylvania State University, 2007.

[34] H. Anton, B. Kolman, and B. Averbach, *Applied Finite Mathematics*, Saunders College Publishing, 1992.

[35] G. Giger, M. Kandemir, and J. Dzielski, "A Graphical Mission Specification and Partitioning Tool for Unmanned Underwater Vehicles," *IEEE OCEANS 2007*, October 2007.

[36] D. T. Davis, and D. Brutzman. "The Autonomous Unmanned Vehicle Workbench: Mission Planning, Mission Rehearsal, and Mission Replay Tool for Physics-Based X3D Visualizations," *Proceedings of the 14th International Symposium on Unmanned Untethered Submersible Technology*, Durham, NH., August 2005.

[37] I. Woodrow, C. Purry, A. Mawby, and J. Goodwin, “Autonomous AUV Mission Planning and Replanning – Towards True Autonomy,” *Proceedings of the 14th International Symposium on Unmanned Untethered Submersible Technology*, Durham, NH., August 2005.

[38] S. Erkan, M. Kandemir, and G. Giger, "Dynamic Fault Tolerant Mission Re-Planning Algorithms for a Group of UUVs," *Proceedings of the 15th International Symposium on Unmanned Unthethered Submersible Technology*, Durham, NH., August 2007.

[39] G. Giger, M. Kandemir, and J. Dzielski, "Reliable Mission Execution Using Unreliable UUVs," *AUVSI’s Unmanned Systems North America*, San Diego, CA., June 2008.

[40] R. Peel and B. Williams, “Bringing a Common Operating Picture to Fleet UUV Exercises," *AUVSI’s Unmanned Systems North America*, August 2004.

[41] S. Erkan, M. Kandemir, G. Giger, and S. D. Lovell, "Energy-Optimal Data Collection and Communication Using a Group of UUVs," *AAAI Fall Symposium on Regarding the Intelligence in Distributed Intelligent Systems*, Washington D.C., November 2007.