

A COMMON CONTROL LANGUAGE TO SUPPORT MULTIPLE COOPERATING AUVS

Christiane N. Duarte¹ duartecn@npt.nuwc.navy.mil

Gerald R. Martel¹ martelgr@npt.nuwc.navy.mil

Christine Buzzell¹ buzzellcm@npt.nuwc.navy.mil

Denise Crimmins¹ crimminscr@npt.nuwc.navy.mil

Rick Komerska² komerska@ausi.org

Sai Mupparapu² sai@ausi.org

Steve Chappell² chappell@ausi.org

D. Richard Blidberg² blidberg@ausi.org

Robert Nitzel³ rnitzel@technologysystemsinc.com

1) Naval Undersea Warfare Center – USA

2) Autonomous Undersea Systems Institute – USA

3)Technology Systems Inc. – USA

Abstract

The following paper describes the development and current state of a “Common Control Language” (CCL) for autonomous undersea vehicle (AUV) monitoring and control. The CCL is a protocol that will provide a means for different AUVs and their users to communicate a common set of commands and information between one another. It is specifically tailored for AUV platforms and the underwater domain. Recent efforts in developing the language have consisted of refining the AUV basic behavior interface and implementing an interpreter for CCL on board the Solar-powered AUV (SAUV). The interpreter receives CCL task definitions and translates them into active processes that will execute the directives. These processes are instantiated and maintained using the distributed control environment (DICE). The CCL directives have been highly structured to capture the various aspects of basic behaviors which make sense when interfacing with an AUV. In addition, messaging using CCL has been kept to a minimum reducing the acoustic transmissions to a few bytes

We also summarize the recent AUVFest 2005 operational event where many of the critical pieces of our CCL architecture were field tested. These include a next generation SAUV controller which is being modified to use the DICE framework and the CCL interpreter as well as the

mission planning and monitoring tools required to task and monitor the group of AUVs.

1. Introduction

Unmanned systems are an area of growing war-fighter interest due to their potential as force multipliers, especially against asymmetric threats. Furthermore, swarms or collaborative groups of unmanned vehicles are seen as revolutionary and transformational technologies for war-fighting and homeland defense. Future concepts for autonomous operations are projecting the use of heterogeneous platforms in group operations in order to address various operating regimes and specialized tasks, thus introducing unique issues for coordinated operations. In addition, operator and group members are more effective when some level of communication can exist between agents. This has motivated the investigation [1-8] of a common control language (CCL) to establish both a standard for communicating between different agents (i.e. underwater vehicle, surface vehicle and human operator) during operations and to support the interactions between operator and machine. CCL must meet the needs of an operator managing a group of vehicles or a single vehicle. CCL must allow concise representation of goals or tasks by autonomous agents.

¹ Research partially supported by grants from Office of Naval Research ONR N000140410264 and ONR N0001405WX20076

The paper is divided into the following sections. Section 2 introduces our CCL concept and driving requirements. Section 3 details the basic behavior interface which specifies a generic messaging interface and core behaviors for AUV platforms. Section 4 discusses the higher level planning capabilities of our CCL. Sections 5 and 6 describe the DICE behavior run-time environment and how this environment is implemented on the SAUV. Section 7 provides a deeper look into the DICE environment and shows an example of the CCL language we use. Finally, section 8 describes the AUVFest 2005 mission which we used as a key in-water demonstration of these concepts.

2. What is a Common Control Language?

We can start with our definition of a CCL as:

A concise and robust language to be used by autonomous agents for communicating information and coordinating tasks.

We clarify further the term autonomous agent by defining some minimal characteristics:

- Can “understand” a set of task specifications
- Is situationally-aware
- Can make decisions

An agent can recognize a task as within its repertoire and take responsibility to perform the task or respond by ignoring it or informing others that it is unfamiliar with this task. During a mission, the agent is able to maintain some degree of awareness of its surroundings depending on the accuracy and resolution of its sensor suite. The level of awareness should be detailed enough to recognize key events related to the tasks it can perform. An agent should be capable of making decisions even if rudimentary.

Our approach adheres to the following overall design goals for the implementation of a common control language:

- One vehicle can not look inside another vehicle; coordination is through message-passing
- It should allow for arbitrary execution of behaviors, e.g. repetition, sequential or parallel
- It should be easily extensible to new vehicles and new missions
- It should be possible to use for both real and simulated vehicles
- Users should be able to add their own messages if required but not expect that these new messages will be understood or achievable by all vehicles.

With these definitions in mind we can make an analogy of a group of autonomous agents to a network of computers where the role of a CCL can be compared to a computer

instruction set. There are basic instructions that a computer can support and a programmer combines these instructions in sequential, parallel and iterative constructs. In the computer world, this has evolved from assembly to high-level languages such as C++ and Java. CCL has a basic set of instructions or behaviors that we have defined called basic behaviors. They are the building blocks for autonomous vehicle interactions and we use them in both direct inter-vehicle messaging as well as the atomic units in building mission files.

In reactive-based controllers, behaviors are downloaded and used in a pre-determined fashion. What we propose is a language that will allow autonomous agents, specifically AUVs, to organize and combine these behaviors to task a vehicle or group of vehicles in real-time, based on the situation. Currently there is no standard for such a language. Traditionally, message passing between two or more agents has consisted of pre-defined handshakes of data or triggers specific to a single task.

A key idea in the exchange of commands and information among platforms comes from software agent communications documented in the Foundation for Intelligent Physical Agents Communicative Act Library Specification [9], as well as earlier work in AUV conceptual communication language development [7]. In the present design, our CCL expresses message context in two broad ways:

- **Request** or imperative message (*Request, Urgent Request, Command*) directing the platform to perform a certain behavior, and
- **Inform** message (*Inform, Warn, Urgent Warn*), propagating data, information and knowledge throughout the system.

Each CCL message is composed of an ordered set of structures which include a header section, optional authority, scheduling and looping sections, and a variable sized body section. The grammar is designed so a parser does not need to backtrack.

In addition to the work described in this paper, there is another effort underway at Woods Hole Oceanographic Institute (WHOI) to define a “Compact Command Language” [10]. The WHOI CCL is the basis for acoustic communications with the Remote Environmental Measuring Units (REMUS) AUV and its derivatives, though it has been designed to be generic enough to be applied to other AUVs. This protocol is currently mandated as the standard communication protocol to be used among AUVs within the Office of Naval Research’s (ONR) Very Shallow Water /Surf Zone Mine Countermeasure (VSW/SZMCM) program.

Both our CCL and the WHOI CCL protocols are designed for low bandwidth inter-AUV and AUV-operator communications. Both protocols provide a standard

messaging scheme which addresses various functional capabilities of an AUV. In addition, both protocols provide user support for encoding and decoding of packed message structures. This is particularly useful for our CCL given the variable message size and complexity of the language. Finally, both language protocols claim to be easily extensible to new vehicles and missions.

There are some important differences between the two protocols however. The WHOI CCL is designed around the capabilities of the WHOI Utility Acoustic Modem (UAM) and WHOI Micro Modem, which segment control, status and sensor messages into 32 byte ASCII protocol packets. More importantly, our CCL is being developed in part to enable research of issues pertaining to multiple heterogeneous cooperating AUVs. Our messaging interface in particular draws heavily upon past work in other AUV command languages, as well as work done in intelligent agent communications. Our CCL is explicitly designed to allow for arbitrary execution of behaviors (parallel, sequential, adversary, general choice, cost choice) and, when combined with λ -calculus, allows vehicles to accept a goal, jointly plan how to achieve that goal and carry out the plan.

3. Basic Behavior AUV Interface

Our work draws upon previous research which identified a set of nine generic or basic behavior functionalities common to all AUVs [5,6,11,12]. Figure 1 lists six of these nine categories which we have focused on at the present time.

<p>Maneuver</p> <ul style="list-style-type: none"> • GoTo • Maintain Position • Transit 	<p><i>“Move or relocate yourself in this manner.”</i></p>
<p>Navigate</p> <ul style="list-style-type: none"> • GPS Fix • Avoid Region 	<p><i>“Provide path constraints and position updates.”</i></p>
<p>Communicate</p> <ul style="list-style-type: none"> • Status • Capabilities • File • Parameters • Message 	<p><i>“Tell me something about yourself or here is something you should know about me.”</i></p>
<p>Configure</p> <ul style="list-style-type: none"> • Parameters 	<p><i>“Change some pre-configured aspect of yourself.”</i></p>
<p>Execute Convention</p> <ul style="list-style-type: none"> • SystemAdmin • ModifyBehavior 	<p><i>“Carry out a universally understood action.”</i></p>
<p>Monitor</p> <ul style="list-style-type: none"> • Parameter 	<p><i>“Watch (and record) some aspect of yourself and potentially report these aspects to someone.”</i></p>

Figure 1. Basic Behavior Interface

The maneuver behavior class is the primary interface for commanding the AUV to move or relocate itself to a new position. It contains three specific behavior types: *GoTo*, *Maintain Position* and *Transit*.

In the *GoTo* behavior, the AUV will attempt to maneuver to the specified latitude and longitude or positional offset from the current position or preconfigured origin, and depth or altitude along the specified path at the specified speed. The behavior is complete when the AUV believes it has attained the target destination (within the pre-configured waypoint radius and waypoint depth tolerance settings) or when the specified or default timeout occurs.

For the *Maintain Position* behavior, the AUV will attempt to remain within a circle at constant depth while minimizing energy usage. If the AUV drifts beyond the circle radius or depth bounds, it will attempt to move back to the circle center. If position parameters are specified, the AUV will consider these to be the circle center; otherwise the center is set at the current vehicle position. If orientation parameters are specified, the vehicle will attempt to maintain the indicated orientation during this behavior. The AUV will end this behavior when the specified or default timeout occurs.

In the *Transit* behavior, the AUV will attempt to orient itself to point in the indicated direction and move at the specified speed for a given duration. In addition, the vehicle can also be directed to attain a specific velocity (speed & direction) while maintaining a fixed orientation or rotation rate. This behavior is much more compass driven than the *GoTo* behavior, which is goal point oriented. As with the previous two maneuver behaviors, the transit will end when the specified or default timeout occurs.

When acknowledging a request to maneuver, an AUV will send a message using the informative mode (i.e. *Inform*, *Warn* or *Urgent Warn*). The message contains information as to why it was generated, and a timestamp, if requested.

The navigate behavior class forms the primary interface for specifying path constraints and position updates for the vehicle as it maneuvers. It contains two behavior types: *GPS Fix* and *Avoid Region*. For each of these requests, the requestor may ask for an acknowledgment that the recipient has received and understands the request.

While executing the *GPS Fix* behavior, the platform attempts to update its geo-referenced position using its GPS navigation sensor. The platform should be in a location that allows for the indicated navigation system to work; otherwise a failure message may be broadcast. The activity is complete when the position fix has been acquired or a timeout occurs.

For the *Avoid Region* behavior, the vehicle is requested to avoid entering the designated regions, specified as a spherical or vertically-aligned cylindrical volume with a specified radius. Once set, avoidance regions remain in effect until explicitly cleared.

When acknowledging a request to execute a navigate behavior, the AUV will send a message using the informative mode. In a similar manner to the informative maneuver message, this message contains information as to why it was generated and a timestamp, if requested.

The communicate behavior class is the primary interface for querying the vehicle about itself, and for providing such information to a requesting agent. It consists of four behavior request types; *Status*, *Capabilities*, *File*, and *Parameters*. A fifth type, *Message*, is used as a means for communicating generic unsolicited information.

A *Communicate Status* behavior is used to request that an AUV transmit its current core status block plus up to 15 additional monitorable parameters. This status request can be specified as a one time request (i.e. poll) or as a periodic update. The returned informative status message contains information as to why it was generated, who the requestor is, an optional timestamp, as well as the vehicle's status information block. This status block includes high-level indicators of subsystem state, current behavior, platform orientation, position, speed, navigation strategy, GPS signal type, and available energy. Additional information from requested monitorable parameters is also returned. Figure 2 describes the core status block.

Element	Bits	Data Type / Range	Description
agent_id	6	integer, 0...62	Unique platform identifier.
group_id	2	integer, 0...2	Platform's group identifier.
health_ok	1	CCL_BOOLEAN_TYPE	Indicator of overall vehicle state.
heading	9	integer, 0...360	Vehicle heading [deg].
speed	10	integer, 0...1023	Vehicle speed [cm/s].
current_nav	4	CCL_NAVIGATION_TYPE	Current navigation strategy in use.
maneuver_type	2	CCL_MANEUVER_CONTROL_TYPE	Current maneuver strategy in use.
subtask_id	7	integer, 1...128	Currently executing subtask identifier.
avoiding_obstacle	1	CCL_BOOLEAN_TYPE	Is obstacle avoidance strategy in use?
lat	32	4 B float	Vehicle latitude [decimal deg].
lng	32	4 B float	Vehicle longitude [decimal deg].
depth	32	4 B float	Vehicle depth [decimal meters].
altitude	16	integer, 0...65535	Vehicle altitude [cm]
available_energy	16	integer, 0...65535	Available energy on vehicle [whrs].
time	32	4 B Unix time_t	UTC time since Jan 1, 1970 [sec].
-	6	-	Byte alignment padding.

Figure 2. Core Status Information Block (26 Bytes)

A *Communicate Capabilities* behavior is used to request that an AUV transmit a list of its capabilities. The corresponding informative message contains information as to why it was generated, who the requestor is, an optional timestamp, as well as the platform capabilities. The capabilities information structure includes platform operational role and range, platform type and size, control properties, maximum/minimum/cruise speeds, energy system type, maximum/available/reserve energy, communications and navigation capabilities, networking protocol, and on-board mission sensors.

A *Communicate File* behavior is used to request that an AUV transmit a specific file to the requestor. The file name is provided and, optionally, a directory path where to search

for the file. The corresponding informative message is similar to the preceding capabilities inform message, with the exception that the returned payload is the requested file.

A *Communicate Parameters* behavior is used to request that an AUV transmit the requested parameter values. In addition to honoring request and inform messaging, vehicles are expected to support a well-defined set of state parameters. These parameters provide a support interface for both monitoring and configuring supplementary aspects of the vehicle. The architecture also allows designers to specify additional parameters that are specific to their AUV. Up to 32 monitorable parameters may be requested in a single *Communicate Parameters* message. The corresponding informative message contains information as to why it was generated, the requestor, an optional timestamp, and the requested parameter values.

A *Communicate Message* behavior is used to convey unsolicited information between platforms. A message may contain platform-specific information in ASCII or binary format. The message content size is currently limited to 255 bytes.

The configure behavior class is the primary interface for specifying how the platform should reconfigure aspects of itself. It contains a single behavior type: *Parameters*, reflecting the notion that key internal aspects of the platform are accessed through the parameters mechanism.

For each *Configure Parameters* request, the AUV is requested to modify the specified configurable parameter with the supplied value. Note that configurable parameters are defined in both the well-known parameter list as well as an optional vehicle-specific custom list. Up to 32 parameters may be configured in a single request message. For each of these requests, the requestor may ask for an acknowledgment that the recipient has received and was able to carry out the configure behavior.

The execute convention behavior class forms the primary interface for specifying how an AUV should carry out a well-known system-wide action. It currently contains two behavior types: *SystemAdmin* and *ModifyBehavior*. The platform may also send an acknowledgment message, if requested.

For the *SystemAdmin* behavior, the AUV is requested to alter the specified system administrative aspects. This includes shutting down and restarting the high-level control applications, and shutting down and restarting the platform.

For the *ModifyBehavior* behavior, the AUV is requested to carry out actions which modify its high-level behavior. These include altering the current task and/or mission, as well as switching between different operating modes of the vehicle.

The monitor behavior class is the primary interface for specifying how the AUV should watch some aspect of itself and log and/or report these aspects to others. It contains a single behavior type: *Parameter*, which permits monitoring of a monitorable parameter value. Report generation is triggered at specified time intervals or when a monitorable

parameter value or its derivative crosses a settable upper or lower bound threshold. Reports may be logged and transmitted to other agents. For each *Monitor Parameter* request, the requestor may ask for an acknowledgment that the recipient has received and is able to carry out the monitor action.

4. CCL with an Embedded Planner

Our CCL interpreter is integrated with an embedded planner that will allow optimization of CCL directives to cope with dynamic aspects of the environment. The planner implementation is based on the $k\Omega$ -optimization search method [13]. With the planner, the CCL interpreter creates task managers that adaptively look for the best sequence of basic behaviors for the directives it has received. In many cases, a static plan of action will not work; thus planning for the best actions will be done incrementally. To do this, the task managers will build trees of potential solutions and search for the optimal sequence of actions.

The problem solving works iteratively through the select, examine and execute phases. In the select phase the tree of possible solutions is generated up to k steps ahead, an agent identifies its behaviors of interest for optimization which are included in its Ω set (set of behaviors). The agent may not always have a complete set of behaviors due to limitations on vehicle configuration, controller or tasks. This means that the tree of solutions may not be complete enough in width and depth to deal with complexity. However, incomplete parts of the tree are modeled by silent ϵ -expressions, and their cost estimated (i.e., not all information is lost). The above means that the $k\Omega$ -optimization can be complete and optimal if some conditions are satisfied.

In the examine phase, the trees of possible solutions are pruned thereby minimizing the cost of solutions, and in the execute phase up to n instructions are executed. Moreover, because the ϵ -calculus cost operator may capture not only the cost of solutions, but the cost of resources used to find a solution, we obtain a powerful tool to avoid methods that are too costly. In other words, the ϵ -calculus directly minimizes search cost. This basic feature, inherited from anytime algorithms, is needed to tackle hard optimization problems and allows solving total optimization problems (the best quality solutions with the minimal search costs).

5. DICE Controller Framework

The Distributed Control Environment (DICE), a NUWC group control development tool, is an extension of techniques [14] of behavior based control design utilizing port-arbitrated behaviors. We have extended this approach to support real-time manipulations of behaviors and ports. There are wildcards for connection of behaviors, both those currently running and those that will potentially run. Ports can be created and associated with a behavior during runtime. This is a powerful capability in adapting to changing situations such as unanticipated sensor failures.

The motivation for these extensions is to establish a highly dynamic capability for activation and organization of behaviors to support CCL commands.

An interpreter will need a wrapper so each controller already on the vehicle can understand the CCL. This will have to be a unique wrapper for each native vehicle controller. The CCL interpreter will work through DICE to interface with the native vehicle controller. DICE is the framework for creating a vehicle wrapper.

6. CCL and DICE Implementation on SAUV

In the spring of 2005, the SAUV development team decided that a behavior-based approach would better allow for research into cooperative behaviors and provide enhanced support for future extension of SAUV capabilities. With this in mind and the continued efforts in CCL development, the DICE framework was selected as the overall approach to the next generation SAUV controller.

There were several layers of control which were important to the development step. There was significant legacy software that needed to be recovered. The structure of the previous software was based on a series of software "managers" for the major components of the control system: Communication Manager, Navigation Manager, Data Manager and Mission Manager. Fortunately, this existing structure in the legacy code was conducive to a behavior-based design.

The current SAUV / DICE design consists of a three layer approach, as shown in Figure 3. In the lower layer, all of the original C code based managers as listed earlier with the exception of the Mission Manager were re-used with some modifications to message passing. This also included maintaining the same boundary to low-level processes on the SAUV which interface to the hardware.

The second layer comprises the basic behaviors which form the "common" part that should exist on all AUVs so that they can "talk" CCL. In the diagram, these basic behaviors are the interface to the native vehicles software both in commanding the vehicle and receiving status from vehicle subsystems. On the SAUV, the basic behaviors are implemented using the DICE framework. These behavior processes are activated at system startup and are idle until called by high-level task or direct operator request.

In order to complete the interface between the legacy code processes and DICE behaviors, a "bridge" behavior was designed. The bridge is implemented in DICE but also interacts with the legacy C code of the SAUV controller. The bridge process was required primarily to support the different paradigms for message passing between the SAUV low level (using Universal Datagram Packet) and DICE (using shared memory).

The top layer of this architecture consists of the runtime "aggregate" or task manager behaviors that are created by the interpreter. The code for these behaviors is based on the CCL directives that were written and sent as a task. The

main control functions of a task manager are update, select, examine and execute as defined by the $k\Omega$ -search method. Therefore a task manager is a real-time planner for the specific task at hand.

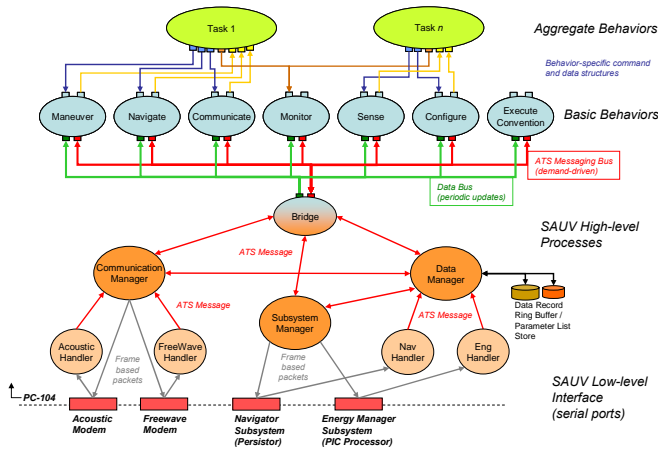


Figure 3. SAUV / DICE Control Architecture

The architecture defines the message passing interface between the aggregate and basic behavior processes. These are categorized as either request or acknowledge type messages. When planning and executing a task, an aggregate behavior will send a request to a basic behavior process for a specific action or status update. The aggregate behavior then receives an acknowledge messages from the basic behavior to complete the handshake. If a vehicle cannot accomplish a particular behavior request, it should report back this lack of understanding.

7. Mission Specification and Execution

Missions are specified in our CCL using a syntax based on that of λ -calculus and conventions taken from the Lisp and C programming languages and is designed to be concise in capturing the interactions of the vehicles. A CCL mission file will contain a mission level task statement and one or more tasks associated with that mission task. The rest of the language contains similar syntax to that of Lisp but uses prefix notation for mathematical and relational functions. Figure 4 shows a hypothetical mission, *mp3*, which incorporates together a rectangular box maneuver and a watch circle maneuver.

We have developed a simple CCL mission editing tool which allows us to create and edit missions. Missions are composed of multiple tasks, which are built upon basic behaviors or other tasks. A task is an aggregate of behaviors and/or complex behaviors. Its definition includes a subtask section that lists the behaviors that need to run. Complex behaviors can also be specified in a subtask section; these are not basic behaviors but rather a task with its own subtask definition. Thus tasks can be nested inside tasks. Part of the definition of a task also includes the

planning operator that should be used such as sequence or parallel. These operators are defined in the language and are used by the $k\Omega$ -planning embedded in a task manager. If sequence is specified then all behaviors listed in the subtask are planned in sequence (i.e. there really is no planning). If both sequence and parallel are specified then plans will be created that run some behaviors in parallel and some in sequence and then evaluate these plans to determine which combination is best. To date, we have used only the sequence operator for our in-water demonstration but have used other operators in land robot experiments.

Let's look at an example using the *mp3* mission where the subtask list consists of box and watch circle (*wc*) tasks. In the *mp3.ccl* file, a *box* task, a *wc* task and an encompassing *mp3* task are defined. A *box* task requires that the vehicle perform a box trajectory with the subtask list defining the waypoints and other maneuvering aspects in the *GoTo* basic behavior.

```

task mp3
{
  (: (init)
    (void)
    (= k 1)
    (= b 1)
    (= n 1)
    (= t 1)
    (= operator_set SEQUENCE))
  (: (subtasks)
    (void)
    (= Complex box)
    (= Complex wc)
  )
  task box
  {
    (: (init)
      (void)
      (= k 5)
      (= b 1)
      (= n 5)
      (= t 1)
      (= operator_set SEQUENCE))
    (: (subtasks)
      (void)
      (= Maneuver CCL_MANEUVER_GOTO CCL_SURF_LOC_LAT_LNG 41.555933
        71.339067 CCL_SURFACE 0 CCL_CONSTANT_DEPTH_PATH CCL_SPEED_MAX 0 )
      (= Maneuver CCL_MANEUVER_GOTO CCL_SURF_LOC_LAT_LNG 41.557000
        71.339067 CCL_SURFACE 0 CCL_CONSTANT_DEPTH_PATH CCL_SPEED_MAX 0 )
      (= Maneuver CCL_MANEUVER_GOTO CCL_SURF_LOC_LAT_LNG 41.557000
        71.330000 CCL_SURFACE 0 CCL_CONSTANT_DEPTH_PATH CCL_SPEED_MAX 0 )
      (= Maneuver CCL_MANEUVER_GOTO CCL_SURF_LOC_LAT_LNG 41.555933
        71.330000 CCL_SURFACE 0 CCL_CONSTANT_DEPTH_PATH CCL_SPEED_MAX 0 )
      (= Maneuver CCL_MANEUVER_GOTO CCL_SURF_LOC_LAT_LNG 41.555933
        71.339067 CCL_SURFACE 0 CCL_CONSTANT_DEPTH_PATH CCL_SPEED_MAX 0 )
    )
  }
  task wc
  {
    (: (init)
      (void)
      (= k 1)
      (= b 1)
      (= n 1)
      (= t 1)
      (= operator_set SEQUENCE))
    (: (subtasks)
      (void)
      (= Maneuver CCL_MANEUVER_MAINTAIN_POSITION CCL_MAINTPOS_LAT_LNG
        41.555933 71.339067 CCL_FALSE 0 CCL_TIME_RELATIVE 10 CCL_TIME_MIN )
    )
  }
}

```

Figure 4. Sample CCL Mission Script File “mp3.ccl”

A watch circle is a special task that the SAUV performs to re-charge itself when on the surface. The vehicle maneuvers to a particular latitude and longitude on the surface and then drifts. If it drifts more than a specified distance or radius away from the center point, it will turn

on its thrusters and motor back to the center point. Its subtask list consists of one maneuver with a *Maintain Position* basic behavior, with the location specified to be on the surface. When *box* or *wc* are nested as subtasks of another task, as they are in the *mp3* task, they become complex behaviors.

Once these tasks are created, they are stored as ASCII text in a file and then sent to the vehicle. This is currently done using file transfer protocol (ftp) over a radio frequency (RF) link while the AUV is on the surface. Once the mission file is onboard the vehicle, an *Execute Convention ModifyBehavior* message is sent instructing the AUV to run the mission.

When the Execute Convention behavior receives a request to begin the mission, it calls on a DICE behavior called *CCLi* (see Figure 5). This behavior controls the process of interpreting the given CCL file.

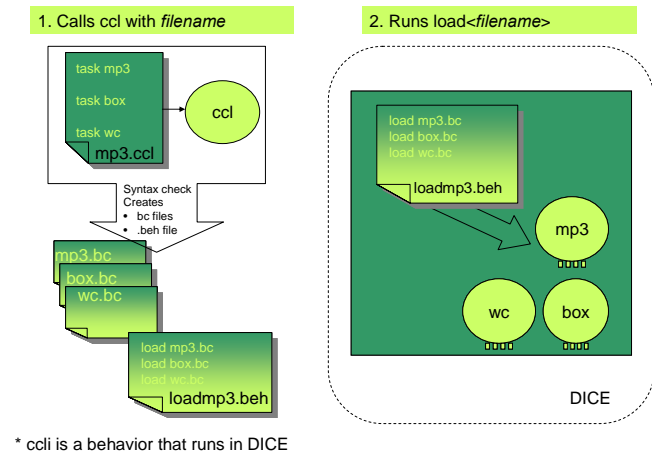


Figure 5. *CCLi* Behavior Role

CCLi first calls on the CCL interpreter process (labeled as *ccl* in Figure 5) which will check the syntax of the task definitions and create new behavior cell (*.bc*) files. The behavior cell format is used to specify all new behaviors. In our example using the *mp3.ccl* file, *ccl* creates the *mp3.bc*, *box.bc* and *wc.bc* files.

The CCL interpreter also creates a DICE script file (*.beh*). This file initiates the load process for each *.bc* file which creates the respective task managers. A task manager controls its own connections between itself and the basic behaviors so that it is able to send requests and receive status to and from the basic behaviors when executing its subtasks. Along with providing the code for real-time planning, a special code library (labeled as *libYada* in Figures 6 and 7) defines the interface that a task manager will use to interact with the basic behaviors. Figure 6 diagrams the connections necessary for the *box* task manager to run a *box* survey. Figure 7 illustrates how multiple tasks use a similar interface mechanism to communicate between each other as they use when communicating with the basic behaviors.

8. AUVFest 2005 In-water Test Plan and Results

We chose the ONR-sponsored AUVFest 2005 event held in June in Keyport, WA as a key goal for in-water testing of our CCL architecture. With access to three SAUVs, we focused our demonstration goals on a multiple vehicle survey with emphasis on persistence. The mission we planned is depicted schematically in Figure 8. The drawing depicts an acoustic network of five nodes. Each of the three SAUVs (2 from AUSI and 1 from NUWC) are individual mobile nodes. The NUWC Mid-sized Autonomous Research Vehicle (MARV) was also factored into the plan as the fourth node, making it a heterogeneous operation. Finally, the operator console application linked to the gateway buoy formed the fifth node. Note that gateway buoys are often used to provide a transparent communications bridge between RF and acoustic communications. Underlying the communications was the testing of two acoustic network protocols (COFSNET & AUSNET) to maintain connectivity of the group.

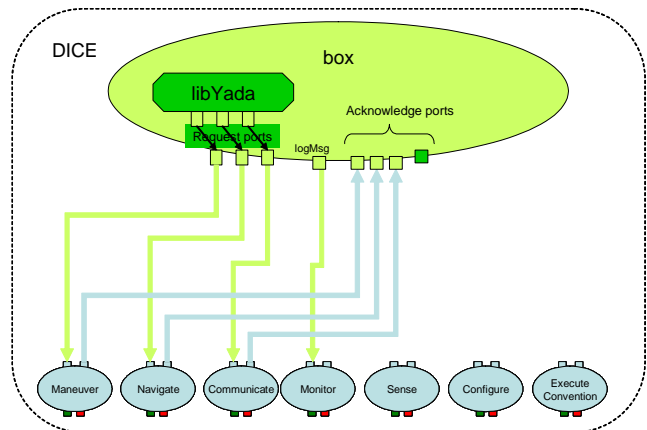


Figure 6: Aggregate and Basic Behavior Interface

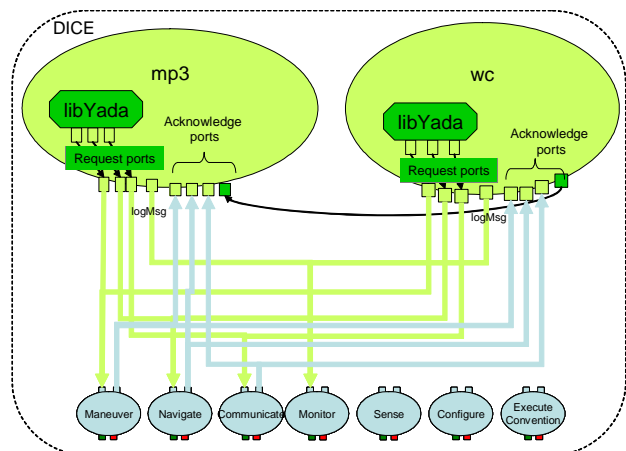


Figure 7. Complex Behavior Interaction

In our scenario, three SAUVs were to be launched from shore and take up their respective positions, each at its own watch circle location, as shown in the diagram. The watch

circle radii would be 100 meters. The survey mission is a rectangular pattern of 2 km by 250 meters (4.5 km distance). The three SAUVs would alternate conducting the underwater survey mission. The vehicle conducting the underwater survey would acquire a GPS fix at the start and end of the rectangular survey pattern and another fix half-way through the pattern. At a speed of 1.5 knots, the time for each SAUV to complete a circuit including acquiring GPS fixes and returning to its watch circle is about 2 hours.

After the initial survey vehicle has completed its circuit of the survey area, it would return to its watch circle. Then a second SAUV would transit to the survey area and continue the survey, again performing a single loop of the area. After completion of the survey, it returns to its watch circle where the next survey vehicle assignment is determined. The intent was to repeat this process for up to four consecutive 24 hour days. During two of the days in operation, the MARV would run a survey on the outer perimeter and provide data that would trigger one of the two SAUV's still in their watch circles to proceed to a smaller box and perform a shorter survey, then return to its watch circle.

It was decided to employ an incremental approach to handling the role switching. Three different options were considered in increasing order of complexity: 1) The operator is in-the-loop for all role switching and tasking such that the operator will select which vehicle takes over the survey box or goes to the MARV specified survey box based on vehicle status updates. 2) The vehicle uses status information from other vehicles, makes its decision and logs it. The operator still picks the vehicle to role swap. 3) The vehicle makes the decision and executes without operator intervention.

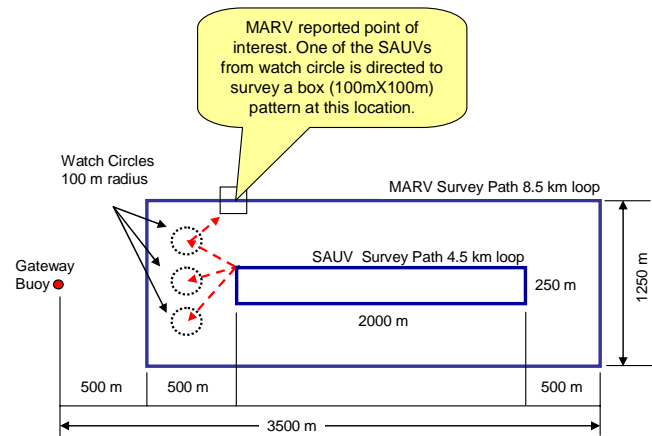


Figure 8. AUVFest 2005 Mission

The mission scenario described above also included a shorter duration operation option. This was presented in the event that 24 hour operations were not possible in our test environment. In this option, the vehicles would go into a

watch circle after the last run of the day. At the end of the day, the vehicles would be tied to the watch circle mooring floats. The floats would have flashers for overnight safety.

The SAUVs would be commanded to report their data in real time via the underwater network at 6 minute intervals to the remote land site. This data would be routed such that it would be sent to the gateway buoy where it would be received acoustically and sent out via RF to the remote land site. This data would be displayed on the operator console screen and on a second laptop for additional data plotting and display. This data would be available to other remote sites also. Additionally, after a SAUV vehicle has returned to its watch circle, having completed a survey of the test area, it would transmit a complete set of all state, sensor, and log data acquired during the survey.

The AUVFest demonstration lasted approximately two weeks. During the first week, the vehicles were unloaded and the systems checked out. One of the vehicles experienced critical hardware failure and was not used during the demonstration. The remaining SAUVs and MARV checked out fine. Following initial delays with the vehicle hardware, there were two environmental issues that impacted our original demonstration goals: currents and acoustic propagation.

We experienced high currents within our operational area during the first week of testing and into the first day of the demonstration. Because these currents prevented the vehicles from successfully achieving the watch circle as planned, the demonstration was changed to the shorter duration operation version.

We also observed asymmetric acoustic communication links within our operational area. These difficult acoustic conditions were credited to the mixing of fresh and salt water. A decision was made to re-design the survey pattern and use a spatial sharing of the survey rather than a temporal sharing of the survey. This would allow the vehicles to have a closer proximity to the gateway buoys.

On the second day of testing, all systems worked well. We were able to achieve many of our objectives and gain good data on the performance of the systems. The following highlights our more salient successes.

The SAUVs were towed to the operational area and although tied down overnight, they were not removed from the water for 72 hours. The extended operational endurance demonstrated that many of the logistical issues of vehicle turnaround that can limit testing can be greatly reduced.

The SAUV missions were designed and then carried out using the CCL architecture described in this paper. The MARV used CCL status packets and CCL status directives during its run to interface with a common operator console. This met our mixed vehicle operation goal. All vehicles were networked acoustically using the COFSNET protocol.

9. Future Efforts

Our original demonstration plan called for autonomous role swapping during the group survey. Unfortunately, we

ran out of time so that is our priority for future efforts. We are looking at different levels of cooperation to achieve this final component of our demonstration and CCL will continue to be extended to support this.

We are also working on reducing the requirement for using ftp over RF for mission file transfer by serializing the mission file in binary format and using the *Communicate File* message to send the information.

Acknowledgments

The authors gratefully acknowledge the support of ONR grants N000140410264 and N0001405WX20076. We would also like to thank the Commander, Naval Meteorology and Oceanographic Command (CNMOC) and the Commander, Naval Undersea Warfare Center (NUWC), as well as the support folks at both Keyport and Newport, for their role in providing the unique test environment of AUVFest 2005.

We would also like to thank Vadiraj Hombal for his continued data analysis and SAUV vehicle operations support.

References

[1] Duarte, C. N., "The Distributed Control of Multiple Autonomous Vehicles for Mine Countermeasure for Littoral Operations", Undersea Defense Technology Europe 99 Conference, July 1-3, 1999, Nice, France.

[2] Duarte, C. N., Werger, B. B., "Defining a Common Control Language for Multiple Autonomous Vehicle Operations", Oceans 2000 Conference, Sept 11- 14, 2000, Providence, RI.

[3] Eberbach, E., Brooks R., Phoha S., "Flexible Optimization and Evolution of Underwater Autonomous Agents", New Directions in Rough Sets, Data Mining, and Granular-Soft Computing, Proc. of the 7th Intern. Workshop on Rough Sets, Fuzzy Sets, Data Mining and Granular-Soft Computing RSFDGrC'99, (Eds. N. Zhong, A. Skowron, S. Ohsuga), Yamaguchi, Japan, LNAI 1711, Springer-Verlag, 1999, pp. 519-527.

[4] Duarte, C., Martel, G., Eberbach, E., Buzzell, C., "A Common Control Language for Dynamic Tasking of Multiple Autonomous Vehicles", Proc. Of the 13th Intern. Symp. On Unmanned Untethered Submersible Technology UUST '03, Durham, NH, August 24-27, 2003.

[5] Blidberg, D. R., "Generic Behaviors: Definition and Structure, An Approach to Modularity in Intelligent System Control Architectures – Volume 1: Technical Proposal" Proposal for SOL BAA #94-19, July 21, 1994.

[6] Blidberg, D. R., "Cooperative Distributed Problem Solving for Controlling Semi-autonomous and Autonomous Oceanographic Sampling Networks - Phase II" ONR BAA #97-021, July 1, 1997.

[7] Turner, E. H. and Chappell, S. G., "Conceptual Communications for Multi-vehicle Systems", Univ. of NH Technical Report #95-08, May 30, 1995.

[8] Buzzell, C. M. "A Common Control Language for Multiple Autonomous Undersea Vehicle Cooperation", M.S. Thesis, University of Massachusetts Dartmouth, October 2004.

[9] Foundation for Intelligent Physical Agents. (<http://www.fipa.org/>).

[10] Stokey, R. P. "A Compact Control Language for Autonomous Underwater Vehicles", WHOI, 16 February 2004.

[11] Komerska, R. J., D. R. Blidberg, S. G. Chappell, and L. Peng (1999) "Progress in the Development and Evaluation of a Standard AUV Command and Monitoring Language," Proceedings 11th International Symposium on Unmanned Untethered Submersible Technology. Durham, NH, August 1999.

[12] Komerska, R., "AUV Common Control Language (CCL) – A Proposed Standard Language for AUV Monitoring & Control", vers. 2.6, AUSI Technical Report # TR-0507-01, AUSI, July 2005.

[13] Eberbach, E., "\$-Calculus of Bounded Rational Agents: Flexible Optimization as Search under Bounded Resources in Interactive Systems", *Fundamenta Informaticae*, vol.68, 2005, pp. 47-102.

[14] Werger, B. B., "Ayllu: Distributed Port-Arbitrated Behavior-Based Control", *Distributed Autonomous Robotics Systems 4*, L.E. Parker, G. Bekey, and J. Barhen, Eds. New York: Springer-Verlag, 2000, pp. 25-34.