Conceptual Communication for Multi-vehicle Systems[*]

Elise H. Turner
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA

Steven G. Chappell
Marine Systems Engineering Laboratory
Northeastern University
Nahant, MA 01908 USA

Abstract

For multiple vehicles to exchange information with and request aid from their collaborators, they must be able to communicate. This communication must go beyond transmitting bit strings. It must allow vehicles to interpret these bit strings as meaningful messages.

In this paper, we argue for the need to provide abstract meaning to the messages based on the vehicle's conceptual representation of the world. We call this *conceptual communication.* We also present a **Co**nceptual **L**anguage for **A**UVs (COLA) that we have developed to support communication between cooperating autonomous underwater vehicles (AUVs).

Elise H. Turner and Steven G. Chappell, 1995. *Conceptual Communication for Multi-vehicle Systems.* Technical Report No. 95–08, Computer Science Department, University of New Hampshire.

[*]This material is based upon work supported by the National Science Foundation under Grant No. IRI-9309703. Steven Chappell's efforts in this work were funded under ARPA contract number MDA972-90-K-0003 and NSF under contract number BES-9222146. We would like to thank the members of the UNH CDPS group for many helpful discussions about this research. In particular, we would like to thank Laura Donia and Scott Valcourt for their help with implementation of COLA; Mike Seymour, Cathy Steffen, Roy Turner and Martin Dempsey for their insights; and Roy Turner for the implementation of Orca. We would also like to thank Dick Blidberg for the initial idea of basing an AUV language on natural languages and for helpful further discussions.

1 Introduction

In complex domains, it is important for agents in a multi-vehicle system to exchange information and coordinate their actions. Communication serves these two important purposes. However, to provide agents with the meaningful messages required for cooperative behavior, we must begin to address issues of *content* in our research on communication. Traditional ocean engineering research in communications, which provides reliable channels for transmitting bit streams, supplies the means for transmitting these messages. We extend the notion of communication in this work to questions of content, focusing on the question of how languages can be constructed to efficiently convey that content.

Like more traditional work, our work must take into account the specific characteristics of the cooperating agents, their tasks and their world that affect their communication. In this work, we are concerned with autonomous underwater vehicles (AUVs) cooperating to achieve some task that is important to their mission. We call such a task a *primary mission task*. Because of the acoustic characteristics of the ocean, the limited resources on-board an AUV, and the complex nature of real-world problem solving, communication between cooperating AUVs must be sensitive to the following:

- The multi-vehicle system may be *heterogeneous*, composed of many types of vehicles.
- Communication is in service of some primary mission task. And, resources used for communication must be added to the cost of achieving that primary task.
- AUVs communicate over a limited-bandwidth channel.
- Due to the complexity of their tasks and the unpredictable nature of their worlds, AUVs cannot have complete information about each other.

All of these problems can be ameliorated if the AUVs communicate about *concepts.* The AUVs' concepts capture their knowledge of the entities in their world and provide the basis for abstract communication. Clearly, all concepts are abstractions of actual entities in the real world. In addition, many concepts capture information about "types" of objects. For example, a particular rock, a rock as a type of object, and a physical object may all be represented as concepts. These types provide additional abstraction for the language. This abstraction makes language processing more efficient by allowing AUVs to communicate important information but to leave out insignificant details which increase bandwidth, increase processing time, and require information about the receiver which may not be known by the sender. We call this type of communication *conceptual communication* to distinguish it from traditional work in communications and from research on biologically-based signal-response communication.

In this paper, we discuss the need for abstraction in communication and present a **Co**nceptual **L**anguage for **A**UVs (COLA). Many features of COLA's design address of the characteristics of AUV communication [20]. Here, we focus on how COLA provides abstract communication. In Section 2, we outline how abstraction ameliorates the constraints on AUV communication. In Section 3, we give an overview of COLA. This section provides enough detail to allow others to implement the language for their multi-vehicle systems. In addition, we discuss how the different constructs of COLA contribute to the abstraction of the language. Many of these constructs are drawn from natural language[1] processing research. In these cases, we discuss how the natural language processing work can be adapted for an AUV language. In Section 4, we present examples of COLA to illustrate how abstract communication can make language processing more efficient.

[1]Natural languages are languages, such as English, which have been created and developed as humans use them to communicate.

One disadvantage of conceptual communication is that it requires a great deal of intelligence on the part of the communicators. Vehicles must be capable of intentional behavior, have a representation of the world, and be able to reason about that representation. In Section 5, we discuss conceptual communication and alternative methods for coordinating the actions of cooperating AUVs, characterizing each in terms of their requirements for agent capabilities and the types of tasks for which they are most appropriate.

2 The Need for Abstraction

Abstraction in communication ameliorates each of the constraints on communication between AUVs listed above. By leaving out the appropriate details:

1. Communication between different types of vehicles is made easy.
2. Computational effort is reduced.
3. Bandwidth is conserved.
4. The receiver is free to exploit knowledge that may not be known to the sender in order to handle communication more efficiently.

2.1 Communication between Different Vehicle Types

Just as programming languages allow the programmer to write code that is independent of the machine on which the code will be run, abstract communication allows vehicles to communicate without knowledge of the internal representations of the other vehicles in the system. This is important because we would like to give multi-vehicle systems the flexibility to utilize a wide variety of vehicles. In addition, we would like to allow vehicles to continue to communicate without changing their language, even when the configuration of the multi-vehicle system changes. This is especially important if the system can be reconfigured during a mission when, for example, a vehicle becomes disabled or a new vehicle becomes available.

Each of the different vehicles in a multi-vehicle system may have a different machine architecture and, consequently, a different way of interpreting bit strings. Different vehicles may also have different representations for their knowledge. If vehicles were to communicate without abstraction, they would have to account for these differences within the bit strings transmitted for each message. Each vehicle would need extensive knowledge of the others' instruction set and knowledge representation at the bit level. Each message would have to be translated from the sender's internal representation to the receiver's. If a new type of vehicle were to enter the system during a mission, the other's could not communicate with that vehicle until all of the necessary information about its internal representation had been successfully communicated.

When communication is abstract, however, it is removed from the details of internal representations. Each vehicle is responsible for translating the shared symbol in the language into its internal representation. Any vehicle which knows the vocabulary and rules of the language can communicate. No additional knowledge of fellow-vehicles is needed.

2.2 Computational Effort is Reduced

By communicating at an abstract level, the overall computational effort required to process the communication is reduced. This is because the abstraction captures the sender's understanding of the communicated information, so, this understanding is communicated along with the information.

For a vehicle to make use of the information contained in raw data, it must “understand” it. In other words, the raw data must be processed so that its meaning can be translated into the vehicle’s knowledge representation language and used for reasoning. For example, suppose a vehicle is on a mission to collect a sample of a radio-active material. An image of the area will only be useful after the vehicle has processed it and has identified the material in that image. Only then will the vehicle be able to go to the material.

When we provide vehicles with a conceptual language, we provide them with a mechanism for transferring knowledge at this level of abstraction. Consequently, once the sender has processed the raw data, it can send the abstracted concept. This spares the receiver from the computational effort required to process the raw data for itself. When processing images, speech, or other complex data, this effort may be substantial.

This principle can be applied all the way up the abstraction hierarchy of the AUV’s knowledge representation. For example, suppose the sender were giving the receiver information to be used for path-planning. It may refer to a rock as an obstacle, communicating information about how the sender classified the rock in this context. Although this form of abstraction may save the receiver processing time in the current context, it may also hide information that will be important in a future context. Consequently, to fully exploit the power of abstraction, we must develop sophisticated language generation tools to help the sender choose the proper level of abstraction for a concept.

2.3 Conserving Bandwidth

Clearly, whenever details are not explicitly communicated less bandwidth is consumed than when all details are transmitted. So, any abstraction will conserve bandwidth. In conceptual languages, the abstraction is based on the concepts in the communicator’s knowledge base. The concept, itself, is an abstraction from the actual entity in the world.

Concepts also promote abstraction in communication because they group the attributes of an object together and associate them with a single name. This name, then, can be transmitted. Although the name is quite short, all of the attributes associated with it are communicated implicitly.

Most standard methods of representing a concept use some form of *slot-filler representation*. *Slots* are the attributes which the entity possesses. For example, the concept **physical object** would include the slots **mass** and **size**. The values of these attributes are called *fillers*. For example, a specific physical object such as a rock might have a filler for **mass** of 5 grams and a filler for **size** of 40cm. Bandwidth is conserved here simply by referring to the concept by name instead of having to list all of its attributes.[2]

2.4 Overcoming Incomplete Information

One of the most difficult problems facing multi-vehicle systems is the problem of incomplete knowledge about other vehicles and the world. This is also the problem on which abstract communication has the most dramatic effect.

On any complex mission, AUVs will face the problem of incomplete information [7]. The underwater environment is unpredictable and it is impossible for any single vehicle to sense enough information to keep a complete world model up to date. We also cannot guarantee that any vehicle will have complete knowledge about another. As the vehicles carry out their assigned portion

[2]If the vehicles that share the language do not share a knowledge representation, their vocabulary can be defined so that such a conceptual hierarchy is established. We discuss this in Section 3.

of a mission, they work in separate areas of the environment and focus on different tasks. This means that each will gain different knowledge from sensing different parts of the world, and each will make different inferences as demanded by its tasks. Since the vehicles will not have the same knowledge, they cannot mimic each other's reasoning exactly. Consequently, no AUV will have complete information about its world or about its multi-vehicle system.

Not all vehicles know the same things, however. The knowledge known by an individual vehicle, but not known by all other vehicles in the system is called *local knowledge*. Not surprisingly, an AUV's most complete and reliable knowledge is often knowledge of itself and its immediate environment. This includes knowledge of the vehicle's own internal states, goals and plans. Consequently, local knowledge is often critical for efficiently performing mission tasks.

Abstract communication allows the message sender to leave out details which rely on local knowledge. By selecting the proper level of abstraction, the sender communicates details which are important to its goals, but allows the receiver to supply all others according to its current situation. This frees the sender from the burden of complete knowledge and permits the receiver to exploit local knowledge to respond to the message efficiently.

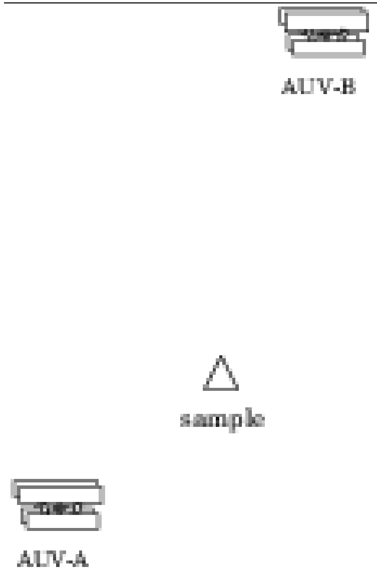


Figure 1: Mission scenario which requires abstract communication.

Abstract communication is particularly important when vehicles give each other commands. Otherwise, incomplete knowledge on the part of the sender could be devastating, causing the sender to expend a great deal of computational effort planning to handle inaccurately described situations and forcing the receiver to carry out commands inefficiently. Consider the mission shown in Figure 1. Vehicle A's task for the mission is to collect the sample. Vehicle A is currently moving to the sample in order to collect it. Vehicle B needs Vehicle A's help for its task, and, so needs Vehicle A to rendezvous with it at its location. We will call this location "X". If Vehicle B knew everything about Vehicle A and the world, it could plan the most efficient path to the rendezvous point. First, it would instruct Vehicle A to collect the sample. Next, it would instruct Vehicle A to go to the rendezvous point. This would increase the bandwidth required for Vehicle B to request the rendezvous because each instruction would have to be explicitly communicated. It would also increase the overall reasoning time for the mission since Vehicle B would not only have to plan the path, but would also need to reproduce all of the reasoning done by Vehicle A throughout the mission to have accurate knowledge of its planned behavior. If we were willing to allow these inefficiencies, however, Vehicle B would be capable of communicating detailed actions to Vehicle A that could be carried out efficiently in the context of the mission.

Remembering that Vehicle B will not have complete knowledge of Vehicle A, suppose Vehicle B did not know of Vehicle A's high-priority task to collect the sample, but does have enough information about the environment to plan a path from Vehicle A's location to X. Vehicle B could command Vehicle A to follow that path by communicating a series of "GOTO" commands by which Vehicle A is sent to the waypoints of the path. However, this will not allow Vehicle A to collect the sample on the way to the rendezvous location. If Vehicle B, instead, gave the more abstract instruction of "GOTO X", Vehicle A would have the flexibility to choose its method for getting to X. In this case, Vehicle A can collect the sample on the way to the rendezvous point and more efficiently achieve these two mission tasks.

Of course, a command can be too abstract. The message must be detailed enough to contain all important information about the sender's goal. In the mission above, for example, Vehicle B must specify that Vehicle A should get to X. In addition, there are circumstances in which a sender should supply as many details as possible. These include: when the sender is managing the activities of several vehicles, when the sender has more information or resources for reasoning than the receiver, when the instruction is urgent and must be carried out immediately, and when the sender is testing and debugging a vehicle. Consequently, there should be a range of abstractions from which the sender can select the proper command for the current situation. The problem of how a sender can determine the amount of information that should be sent in a given message is an open research question. We are currently addressing this question for volunteering information [18; 21] and plan to address the problem for requesting actions in future work. For now, we are interested in building the language tools that allow vehicles to communicate at these different levels of abstraction.

3 COLA: A Language for Abstract Communication between Cooperating AUVs

We have developed a language, COLA, which supports abstract communication between cooperating AUVs. COLA was designed for use by the Experimental Autonomous VEhicles (EAVEs) of the Marine Systems Engineering Laboratory (MSEL) at Northeastern University [6]. Although some details of the language are linked to this specific architecture, the constructs of the language can be easily adapted for other vehicles. However, because of COLA's use of abstraction, vehicles which use COLA must meet certain requirements:

1. The vehicle must represent its knowledge in the form of *concepts* which the vehicle can access. This is necessary because these concepts form the basis of COLA's vocabulary.
2. The vehicle must be able to reason about itself and its environment. This is important because, the vehicle must be able to supply details that have been abstracted out of the messages which it has received.
3. The vehicle must be capable of goal-directed behavior. This is necessary for motivating communication in COLA and for handling received communication within the context of the vehicle's current mission goals.

With this level of intelligence required of language users, we can turn to humans' *natural languages* as our best example of languages for abstract communication between intelligent, cooperating agents. In developing COLA, we have borrowed heavily from the field of natural language processing. The architecture for processing a message and many of the algorithms are borrowed directly from that work. However, the language itself – the words that are used, how those words derive their meaning, and the grammar – have been developed specifically for AUVs. In this way, we can take advantage of techniques that have proven efficient for a specific processing task and increase their efficiency by constructing the language to meet the special needs of underwater communication [22]. In particular, AUV communication differs from human communication because:

- Language designers are not constrained by existing conventions.
- AUVs have only those goals which are directly related to problem solving. They do not, for example, strive to be polite or interesting. This means that their communication can always be straight-forward and literal.
- AUVs have different processing capabilities from humans.
- AUVs communicate under severe bandwidth limitations. This limitation can be relaxed if messages are allowed to contain errors.

In this section, we present an overview of COLA with special attention paid to three contributions of this work. First, COLA supports the abstraction needed to communicate efficiently in the complex environment. We explain how constructs of the language contribute to the abstraction. Second, COLA is an implemented language for cooperating AUVs. The language is presented in enough detail that it can be implemented on other vehicles. Third, COLA shows how natural language processing research can be adopted for communication by artificial agents. In each section that describes concepts borrowed from natural language processing, we point out special considerations for artificial agents.

3.1 Overview of COLA

An utterance in COLA is called a *message*. There are several types of messages. Each message type has a distinctive structure or *syntax*. The message type indicates the goal which motivated the message. We call the AUV which produces a message the *sender*. A *receiver* is any AUV which receives the message, not just the AUV for which the message was intended. The EAVEs communicate via a Datasonics acoustic link [15; 8]. Although a connection must be established between two AUVs for them to communicate, we can simulate broadcasting messages by sending the same message across point to point connections to all other agents. For this reason, the AUV which receives a message does not know if it is the only receiver. So, for the purposes of interpreting messages, we assume that messages are broadcast.

EAVEs have a layered architecture. Each layer handles increasingly complex information and is given more time to respond to input. Ultimately, the top layer of the EAVEs will incorporate a sophisticated mission planner, Orca [24]. Figure 2 shows the architecture of the EAVE's highest layer. It includes a problem solver and a *communication module*. The problem solver is responsible for reasoning about the mission in order to achieve its primary mission tasks and goals. The communication module is responsible for producing and interpreting messages. The communication module works in conjunction with the mission planner to ensure that communication is always processed in the context of the mission. It has the following components:

Lexicon: The lexicon is the dictionary for COLA. It stores all information about the words needed to process a message.

Syntactic processor: The syntactic processor *parses* the message to determine its grammatical structure.

Semantic interpreter: The semantic interpreter builds the meaning of the message from the meanings of the individual words. It passes the interpreted message on to the mission planner.

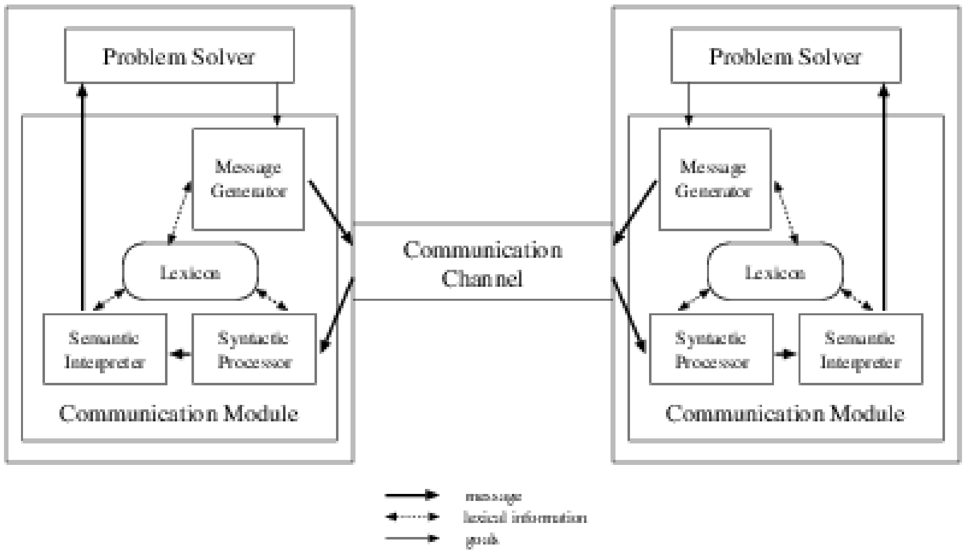


Figure 2: Architecture for inter-AUV communication.

Message generator: The message generator constructs a well-formed message which captures the goal of the sender according to the rules of COLA. It then sends the message to the intended receiver.

In the remainder of this section, we provide details of these submodules and of the specifications of the language itself.

COLA is currently implemented in our Simulator for Multi-Agent Research and Testing (SMART) [23]. Our simulated missions involve two cooperating AUVs. Both of the AUVs have full communication and reasoning capabilities. Both are able to act as either the sender or receiver of a message. Both are able to reason and act autonomously, each having individual goals which may or may not be in service of the cooperative mission tasks. By implementing and testing our language in the simulator, we are able to anticipate capabilities of AUVs that have not yet been tested on actual vehicles. In addition, in our simulator, we are able to isolate parts of the language for testing. Our work in the simulator so far has allowed us to expand the language as we test it on a variety of missions.

3.2 Message Types

It is intuitively appealing to think of the message type as a directive specifying how the receiver will handle the message. Following this way of thinking, if a vehicle were sent a command by another vehicle it would immediately carry out the action. After receiving a statement, it would add the information to its knowledge base. However, it is not always in the best interest of the receiver or in the best interest of the multi-vehicle system as a whole, for the receiver to have such blind obedience.

Instead, it is important for AUVs to have the flexibility to defer or refuse to perform actions associated with a message type. If a command can be delayed, it may be fit into the receiver's plan so that it can be handled more efficiently (see example in Section 2.4). If a statement is not automatically added to the receiver's knowledge base the receiver is free to examine it and decide whether to not the statement should be believed. However, for agents to enjoy this flexibility,

- Inform

  content-roles: situation
  response: add information to knowledge base
  example in English: "There is a rock at location (250 300 450)."

- Warn

  content-roles: situation, *who-warned*, *response*
  features: situation is a threat to the receiver
  response: avoid or eliminate situation
  example in English: "There is a mine in the area; return to the barge."

- Urgent warn

  content-roles: situation, *who-warned*, *response*
  features: requires immediate attention
  response: avoid or eliminate situation
  example: same as for warn, but the mine is in the vehicle's current path

- Request

  content-roles: action, *for-whom*
  response: perform action
  example in English: "Collect samples of radioactive material."

- Urgent request

  content-roles: action, *for-whom*
  features: high-priority action which must be performed immediately
  response: perform action
  example: same as request, but the action must be performed immediately

- Command

  content-roles: action, for-whom
  features: sender has authority over for-whom
  response: perform action
  example: same as request, but task has increased priority because of authority of sender

Figure 3: Message types in COLA.

messages cannot simply mandate actions that the receiver must take.

We turn to the *speech act theory* paradigm [5; 17] of natural language processing to provide this flexibility. In this paradigm, message types convey the *intentions* of the speaker, not a list of actions that the receiver must perform. We can think of this shift from actions to intended actions as a form of abstraction. The important details that are removed are *if* and *when* the actions will be performed. This allows the receiver to reason about the sender's intention in order to decide whether or not to accept the goal and how best to achieve it.

Speech act theory is especially useful for artificial intelligence systems because it fits in well with the formulation of reasoning as planning to achieve one's goals [9]. In this paradigm, utterances are seen as actions. Like other actions, they are motivated by the goals of the actor. Also like other actions, they can fail to satisfy those goals. Consequently, agents can reason about their communicative acts in the same way that they reason about other actions.

This has advantages for both sender and receiver in an AUV language. For the sender, communication is seamlessly motivated by primary mission tasks as a way of achieving subgoals of those tasks. For example, the primary task to collect samples from five different locations may have five subgoals – each to collect a specific sample. A vehicle could achieve one of these subtasks by sending a message asking another vehicle to collect the sample. The receiver benefits by seeing the message not as a mandate, but as an expression of a specific intention of the sender, and, by extension, its higher-level goals and plans. The receiver, then, can decide whether or not to do as the sender intends, making the corresponding actions intended by the sender its own goals.

The receiver can also reason further about the communication as insight into the sender's larger plans [3]. This additional reasoning can require a great deal of knowledge about the sender and can be quite computationally expensive. It is required when plan-based speech act theory is applied to natural language processing to handle a prevalent form of non-literal language called *indirect speech acts*. In COLA we eliminate indirect speech acts simply by requiring senders to choose the form for their utterances which matches their goal. As a result, we eliminate the overhead of fitting each message into a larger plan. However, because we follow the plan-based speech act paradigm, if the receiver needs to recognize the plan of the sender for some other reason, the sender's message can be used to suggest a plan.

COLA's message types are shown in Figure 3. We have drawn these types from the speech act types suggested by Searle [17]. Although the names of these types suggest how they should be used, COLA's message types are actually defined by the sender's intentions and its beliefs about the world. This information is communicated implicitly by the type of message. Information that is explicitly conveyed is called the *content* of the message. There are two broad categories of message types: **informs** and **requests**. Within each category we have the basic message type, and other types which carry information about specific aspects of the action. An **inform** conveys the sender's intent that the receiver add information from the content of the message to its knowledge base. A **request** conveys the sender's intent that the receiver perform some action. A **warn** is an **inform** where the sender is communicating information that may be threatening to the receiver. The structure of the message allows the sender to include information about what the receiver can do to avoid the problem. A **command** is the same as a **request**, except that the sender has authority over the receiver. In addition to those suggested by Searle for natural languages, we include **urgent** message types. Unlike humans, AUVs cannot indicate that a message is urgent by raising their voices or speaking faster. Urgent messages must be given a separate type so that they can be identified. Also, because the urgent message types are explicitly marked as such, they can be identified and handled quickly.

The *content-roles* of each message type specify the function that groups of words will play in relationship to the message as a whole. Content roles in italics are optional. For example, we expect a message of type **request** to include the requested action in its content. If the sender wished to specify an AUV to carry out the action, that AUV's name could be included in the message. If the sender would allow any receiver to perform the action, the sender would not need to include the name of the AUV. *Features* specify features of the world associated with the message types. For example, a **command** is only appropriate if the sender has authority over the receiver. When processing a command, the receiver understands that the sender wishes to exercise that authority. The *response* is the action that the sender intends the receiver to perform as a result of receiving the message. For a **request**, the sender intends the receiver to perform the requested action; for an **inform**, the sender intends the receiver to add the content of the message to its knowledge base.

Of all the information that is associated with a message type, only the content roles need to be explicitly communicated. We give each message type a unique and identifiable structure for this information (as described in Section 3.5). By identifying the message type, the receiver accesses the features and intended response. So, message types allow us to save bandwidth by communicating this information implicitly. Because this information is communicated, definitions of the message types, like those shown in Figure 3, are part of the language and must be part of the linguistic knowledge provided to AUVs communicating with that language. In the current implementation of COLA, this knowledge is represented procedurally in the methods that Orca uses to reason about communication.

3.3 Integration with the Mission Planner

Clearly, to use the intention-oriented interpretation of message types described above, an AUV's communication module must be integrated with its mission planner. The goals which motivate the message originate in the mission planner, and the receiver relies on its mission planner to reason about how the sender's goals will fit with its own. In addition to supporting the abstraction of message types, the mission planner supports abstraction in a conceptual language in two ways. First, it maintains the conceptual knowledge base needed to support the vocabulary of the language. Second, it supplies the reasoning capabilities needed for a receiver to translate abstract messages into detailed instructions. In this section, we discuss the details of the specific knowledge representation and reasoning capabilities currently being implemented for COLA's interface with EAVE's mission planner Orca.

3.3.1 The Knowledge Representation

Orca represents its knowledge using a *frame system* [16]. A frame is a type of slot-filler representation (see Section 2.3). Slots can be filled by pointers to frames or by single values. For example, a rock may contain agate which is represented by a frame giving a description of agate. It may also be at location (670 894 1000) which is represented by this list of numbers. We will call the latter a *primitive value* to distinguish it from values that may be represented as frames. The values of the frame can have *constraints* placed on them indicating the type or range of values that must be used to fill the slot. The frame system reports an error if an attempt is made to fill the slot with a value that does not meet the constraints.

Frames are organized in an *ISA-hierarchy.* The ISA relationship relates subtypes or instances to types. For example, *FIDO ISA DOG* and *DOG ISA ANIMAL.* As with other slot-filler representations, frames bundle information into a single concept. The isa-hierarchy is an abstraction hierarchy which provides the vehicle with a map for reasoning or communication about a concept at a variety of levels of abstraction. Slots can *inherit* default values through the ISA relationship. For example, if dogs are known to bark, Fido inherits this property; if there is not explicit information in the frame representing Fido denying that he does not bark, we assume that he does. Values can also be specified so that they are inherited from other frames within the frame system.

Orca represents its knowledge of events in *schemas.* Schemas are partially ordered sets of *tasks* that can be used to achieve a goal. These tasks can themselves be schemas, goals or *executable actions.* Executable actions are actions which can be carried out by the vehicle.

3.3.2 Reasoning about Communication

The mission planner's reasoning provides the motivation for communication. As it conducts a mission, the mission planner will have some goals that can be satisfied by communicating with other agents. If the mission planner chooses to achieve such a goal through communication, it sends the goal to the communication module. We call this a *communication goal* since it is now the responsibility of the communication module. The language generator constructs an appropriate message and sends it across the communication channel. When the receiver gets a message, its communication module interprets it and passes that interpretation to the mission planner. The mission planner must then decide how to best handle the sender's intentions.

For a **request**, **command**, or **urgent-request** the receiver must decide whether or not to carry out the requested action. This depends on the priority of the requested action relative to the priorities of other actions that the receiver intends to perform. The priority of these actions can be affected by a number of factors, including a value given for the priority slot when the message is sent and the receiver's relationship to the sender.

High priority goals are selected for execution by Orca using the planner's *agenda*. An agenda is a list of goals and tasks that the agent intends to achieve. Requested actions are placed on the agenda when the receiver decides to carry out the action. When an AUV completes one agenda item, it selects the highest priority item still on the agenda and tries to achieve it. This way, the requested goal is handled according to its priority relative to the priorities of the existing goals of the receiver.

When an item is selected from the agenda, it is *expanded.* A goal is expanded by selecting a schema which can achieve that goal and placing its tasks on the agenda. This plan may consist of goals, tasks, and executable actions. Tasks are expanded to less abstract tasks which are placed on the agenda. Executable actions are executed when selected from the agenda. At each step, Orca chooses the expansion which best fits the current situation. Consequently, when the sender gives an abstract goal or action as the content of a **request**, the receiver can expand it so that it is executed efficiently in the current context of the mission.

For an **inform**, **warning** or **urgent-warn**, the sender intends that the receiver add the content of the message to its knowledge base. The receiver must decide whether or not to follow the sender's intentions based on its current beliefs about the world and the and the reliability of the sender. If the message is a **warning** or **urgent-warn**, the receiver must discover what is threatening about the new information and consider removing the threat. To accomplish this, Orca handles the content of an incoming **inform** message as though it were incoming data from a sensor. This way, the information can be evaluated to see if it can be believed before it is added to the knowledge base. **Warnings** and **urgent-warns** can trigger a special mechanism to check for harmful implications from the knowledge.

We chose this approach because we consider adding information to a knowledge base to be a high-priority action that should be taken immediately. Another approach would be to add to the agenda a goal to add information to the knowledge base. This would allow the vehicle to handle even higher priority goals before putting information in the knowledge base, but would incur additional overhead for each **inform**. It is important to note that the priority of adding information could not be set based on the importance of the specific information that is to be added. That could not be known until the information was processed by the mission planner. Instead, all communicated information would need to be given the same priority, or the receiver would have to rely on the sender's assessment of the importance of the information.

Because communication and reasoning are so tightly linked, progress in many current areas of interest for artificial intelligence and intelligent control will improve communication. Specifically, research on evaluating the quality of information and planning in real world domains will be important for communication. Some of these issues are being addressed as part of on-going research on Orca . Others will be addressed by the broader community. For now, the basic architecture for handling messages has been implemented for Orca and is being used to evaluate and test communication in our simulator [23]. As progress on Orca continues, we expect AUVs to make better decisions when handling incoming messages and choosing goals to communicate.

```
WORD       AAQ                            WORD       AXB
CATEGORY   (SLOT)                         CATEGORY   (FRAME OP)
DEF        LOCATION                       DEF        ^GOTO
```

Figure 4: Example lexicon entries.

3.4 The Lexicon

The words of a language provide the mechanism for abstraction at the conceptual level. The meanings of words are linked to the conceptual knowledge of the language user. Since the conceptual knowledge forms an abstraction hierarchy, by providing words for each concept we give the vehicles the ability to refer to entities or actions at several levels of abstraction.

The words of the language also abstract the language from the individual's knowledge base. The *lexicon* is the dictionary each AUV uses to translate a word in the language to a concept, or set of concepts, in its own knowledge representation. The use of a lexicon and the abstraction it provides have three important consequences for communication between AUVs:

1. **AUVs can share the language even if they have different representations for their knowledge.** AUVs can have different knowledge representations for two reasons. First, their knowledge representation languages may be different. This difference could be as comprehensive as using completely different knowledge structures (e.g., frames vs. semantic nets) or as seemingly insignificant as having different symbols for the same values (e.g. "big" vs. "large"). In either case, however, a symbol from one AUV's knowledge base could not be accurately interpreted by another. The knowledge representation of two AUVs could also differ because the AUVs "understood" a concept to mean different things. For example, one AUV might know that ROCK-1 is at location (200 75 500), but another may not know that ROCK-1 exists. If the first AUV refers to ROCK-1, that reference will be meaningless.
   By giving each AUV its own lexicon, the meanings of words can be represented using that AUV's knowledge representation. AUVs then need only know the words and their agreed upon meaning in the language and do not need to know any of the details of the other's knowledge representation.
2. **A one-to-one correspondence between words and concepts is not required.** A lexicon allows a vehicle to distinguish definitions from concepts. The lexicon need not simply point to a concept, instead it may store a definition that is a combination of several concepts or parts of concepts. This ability also allows AUVs to coin new terms to help them communicate more efficiently. The vehicles must agree upon the meaning of the new word, but this meaning may be distinct from any concept in either vehicle's knowledge representation.
3. **Words can be communicated using as few bits as possible.** For humans to be able to read, maintain, and extend the AUV's knowledge base, the names of concepts must be meaningful to them. Similarly, keywords in the grammar should reflect the special structures that they signify. These human-readable symbols require many more bits to be communicated than the minimal number needed to distinguish the words in the language. A lexicon allows us to assign a short bit string to the meaning of the human-readable symbol found in the grammar or knowledge base.

A brief discussion of the lexicon follows. Figure 4 shows examples of *lexicon entries*. Each entry contains three fields of information:

word: is the encoding that will be recognized as a word of COLA
category: gives the syntactic category - the equivalent of a part of speech, such as a noun in English
def: holds the meaning of the word in the knowledge representation of the AUV

The lexicon is accessed for two purposes. During message generation, the lexicon is accessed to find words which capture the sender's intended meaning. When interpreting a message, each incoming word is *looked up* in the lexicon and its syntactic category and meaning are returned. Consequently, lexicon entries are entered into two hash tables. One hash table is indexed by **word** and is used when interpreting messages. The other is indexed by **def** and is used when generating messages.

To simplify communication and cooperation, each vehicle in the system is given a core of knowledge represented by identical frames. This core consists of all prototypical frames, their slots, and the primitive values. Each shared concept corresponds to a word in COLA. There are also concepts in an individual's knowledge base that do not appear in COLA. In particular, when AUVs combine concepts in this core to create a new type or when AUVs add a new instance of an entity to their knowledge base, these concepts do not correspond to words in COLA. The **category** and **def** of a lexicon entry are currently linked to the shared portion of the AUVs' knowledge bases from which COLA's vocabulary is drawn.

Since messages in the language are used to construct a frame that can be handled by the mission planner, the syntax of the language captures the structure of the frames. The syntactic categories, "frame," "slot," and "value," correspond to a symbol's possible use in a frame. Frames are further divided into "op", "agent" and "sit" depending on whether the frame represents an operator (command), an agent, or a concept which is not an operator or agent. Figure 4 shows the correspondence between a symbol's use in a frame and its syntactic category. In Section 3.5, we discuss how these and other syntactic categories were chosen and how they function in COLA's messages.

Currently, in COLA we derive the meanings of our words directly from concepts in the shared knowledge of the AUVs. Each shared frame name, slot name and value provides the exact definition for a word. In addition, there are special keywords in the grammar that are used to mark special structures in COLA. These words have no meaning beyond their use in the grammar, so their **def** slots are filled with the keyword written in English. Finally, AUVs may create new names to refer to entities in the world. We discuss this in more detail in Section 3.6.

In each case, each word has a single meaning that is exactly the same for each language user. This allows us to eliminate the sources of ambiguity that arise in natural languages and concentrate on issues that are more basic to AUV communication. That, in turn, simplifies processing and allows us to reduce the computational effort required to interpret a message. It also allows us to automatically generate the lexicon on a single computer and transmit it to all collaborators before the mission begins. Fortunately, because we are creating an artificial language, we will be able to study the impact of ambiguity on the language as we experiment with COLA and incorporate those types of ambiguity which we see as most beneficial.

It is not necessary for vehicles to share a core knowledge representation, however, because COLA's words are stored in a lexicon. We could relax this assumption simply by creating a vocabulary for COLA apart from any specific knowledge representation and creating a lexicon for each agent.

3.5 Syntax

Syntax refers to the structure of messages. The legal structures for messages are described in the *grammar* of the language. A *syntactic parse* is the process by which the structure of an incoming message is revealed.

Syntax is necessary to COLA because it supports the abstraction to message types. Each message type is given a unique structure which allows receivers to identify it. In addition, syntactic message ::= inform | warning | urgent-warn | request | urgent-request | command

inform ::= situation
warning ::= "warn1" situation agent operator | "warn1" situation operator | "warn1" situation agent | "warn1" situation
urgent-warn ::= "warn" agent situation operator | "warn" situation operator | "warn" agent situation | "warn" situation

request ::= agent operator | operator
command ::= agent operator "comm" | operator "comm"
urgent-request ::= "req" agent operator | "req" operator

situation ::= sit | sit sv | ref
operator ::= op | op p-list | op man-vals sv

sv ::= slot v sv | slot v
v ::= "{" frame "}" | "{" frame sv "}" | val | desname | "{" ref "}" | frame | num | num num num
ref ::= desname | desname sv | desname type class sv
p-list ::= v | v p-list
man-vals ::= v | v man-vals

Figure 5: Message grammar for COLA.

processing can be useful for error detection. The parse will fail if the message contains an error which causes it to have an illegal structure. In future work we will explore how these checks can be exploited to allow linguistic knowledge to complement lower-level algorithms for communication.

Syntax also provides an important abstraction in its own right. The grammar groups symbols of a message into *syntactic constituents.* Within each constituent, words are related in specified ways. This conserves bandwidth because relationships captured in the syntax do not have to be explicitly communicated. The syntactic constituents also create an abstraction with any one of a number of structures serving as the same constituent for a more abstract structure.

At the highest level of abstraction, COLA's constituents are message types (e.g., **inform**, **request**). The constituents of each message type are the content-roles for that type. In order for COLA to use the message, these roles must be filled by Orca's frames. Consequently, to make the translation from a COLA message to an Orca frame more efficient, the structure of the content-role constituents reflects the structure of Orca's knowledge.[3]

COLA's grammar is shown in Figure 5. We use Backus-Naur form (BNF) to specify the grammar [12]. Each line in the figure corresponds to a *re-write rule.* The symbol appearing on the *left-hand side (LHS)* of the "::=" can be written by one of any of the alternatives on the *right-hand side (RHS).* Alternatives are separated by a vertical bar (|). The syntactic categories and all LHS of rules are syntactic constituents. To produce a syntactically legal message, each symbol is re-written until all symbols appear only on the RHS of any rule. These are called *terminal symbols.* If a terminal symbol appears in quotes, it is a keyword and is re-written with the corresponding symbol from the lexicon. Otherwise, the terminal symbol is a syntactic category and must be re-written using any word which is a member of that category.

Each message type is given a distinct structure which includes each of the content roles specified for that type. The structure of the message types are as different as possible. This allows the receiver to identify the basic message type by its structure. We add keywords to **warns**, **urgent-warns**, **urgent-requests** and **commands** so they can be distinguished from informs and requests even

[3]Here, again, if portions of the AUVs knowledge base are not actually shared, we can use the lexicon as the bridge from the model of the knowledge representation used in the language to the actual knowledge representation of an individual AUV.

when no optional information is present.

As discussed above, COLA's syntactic constituents reflect abstractions from the knowledge representation through the content roles of messages. The content roles are described in Section 3. Since content roles function as constituents of the grammar, the legal structures of a content role are the same regardless of the message in which it appears. Other syntactic categories and constituents reflect the structure of the knowledge representation:

The following are atomic values in the knowledge representation:

frame: any frame name
slot: the name of a slot in the preceding frame
val: a symbol that can fill some slot, but which is not a frame

The following reflect the structure of concepts:

sv: gives the attribute/value list associated with a specific frame
v: captures all ways of giving a value to a slot

The following reflect the conceptual category to which the named entity belongs:

agent: any AUV or other agent that is known to all communicating AUVs
op: the frame name of some operation that the AUV can perform
sit: the frame name for any concept that is not an operator
num: a number

The following reflect sets of values that will be used in the same way. By grouping values in this way, we can conserve bandwidth by eliminating slotnames from the communication. These values are used as specified by the grammar and by additional information contained in the frame that they will fill.

p-list: a list of values for all parameters in the same order given in the "param-order" slot of the preceding operator frame

man-vals: a list of values for all parameters in the same order given in the "comm-mandatory" slot of the preceding operator. These values must be given in the message.

We describe **desname**, **type** and **class** in Section 3.6 when we discuss naming descriptions of entities.

Currently in COLA. syntactic processing is performed by an augmented transition network (ATN) [25]. An ATN can be thought of as an enhanced finite state machine. The words of the message serve as the input for choosing the next state. If the message can be used to successfully traverse the ATN with no remaining words, the message has a legal structure. If at any point the input does not correspond to a legal move from the current state, the sentence does not have a legal structure and the parse fails. The ATN is augmented by tests and actions which can be applied to a move from state to the next. During parsing, these tests and actions capture the structure of the constituents. During message generation, the tests and actions are used to select words that capture the intended meaning of the message while following legal syntax for the message type.

Any method for parsing context free grammars (CFGs) would be capable of parsing COLA. However, ATNs have several advantages. They are well-studied and can be implemented for COLA without modification. The tests and actions provide important flexibility. Tests allow the parser to check semantic information during syntactic parsing. If the semantic tests rely on information that must be available to the syntactic parse, they often can be performed more efficiently during syntactic parsing. In addition, if the test fails, the error is found sooner in the message processing. For example, ATNs allow us to check that the slots in a message are legal slot in the preceding frame. The test and actions also allow us to use the same basic ATN with task-specific tests and actions for both understanding and generation. The most important feature of ATNs for COLA is that ATNs are easy to modify and extend. This is important because we expect some details of COLA's grammar to change as it is tested in the water on increasingly complex missions. In addition, because ATNs were designed to handle natural language, so our formalism can handle significant extensions that may be needed in the future.

In addition to supporting abstraction in conceptual communication, we have been careful to construct COLA's grammar to address other important characteristics of communication [20]. We feel that several of these features are worthy of note here because they will benefit others who will extend COLA or who are writing new conceptual languages for their multi-vehicle systems.

The grammar is kept simple. By keeping the grammar simple we save effort in both developing the grammar and in processing messages using the grammar. To keep the grammar simple, we include only symbols that are necessary to convey the conceptual content of the message.

The parser does not need to backtrack. To parse a message, the syntactic processor must choose the rules which will account for the structure of the message. Backtracking occurs when the processor makes the wrong choice and must try alternatives. To be able to backtrack the processor must keep track of all of its choices. If we can guarantee that the processor will never have to backtrack, it can avoid the work of keeping this information as well as the work of trying many alternatives. We guarantee that the parser does not have to backtrack by constructing the grammar so that the parser can always choose the right alternative simply by looking at the word which is to be processed next. Such grammars are called LaLR(1) grammars [2].

Information is made explicit if it will be difficult to infer. Sometimes, although information could be inferred by the receiver, this would require considerable computational effort. In these cases, we must consider the trade-off between the size of the message and the computational resources required to process it. We faced this trade-off when deciding how to handle the names of slots in the initial grammar for COLA. One option was to simply list the values and rely on the receiver's knowledge of the slots to determine which should be filled. For example, in English we can say "The dog is brown." and rely on the hearer's knowledge of dogs to allow him or her to understand that the *color* of the dog is brown. This conserves bandwidth because the name of the slot is not explicitly mentioned, but it increases computation because the receiver must match the given value to the slot which can accept it. The computation can be more complex in cases where a single value could fill several slots. For example, a numeric value could fill many slots associated with frames needed by an AUV. In these cases, both additional knowledge and computation would be needed to unambiguously associate the value with a slot.

We expected few values to usually be given per frame. Although the computational effort to match a value to a slot could be significant, the bandwidth consumed to state the slot would not be significant. For this reason, we decided to explicitly include the slots for each value in most cases.

Not all slots must be explicitly stated, however. There are cases when we expect the values of certain slots to be given with a frame. This is especially true for operators where some parameters must be given. For example, if one AUV requested that another "vert", make a verticle move, we would expect it to specify the depth to move to. For these cases, we specify in the frame the slots which should have values given and the order in which these values should occur. The slot names for all mandatory values are given in a specific order in the "comm-mandatory" slot. As the receiver parses the operator, it simply checks this list to associate the values with the appropriate slot. Similarly, if a sender wishes to send all of the parameters for an operator, it can simply give them as a list following the order given in the "param-order" slot of the operator. This allows us to conserve bandwidth when a significant number of slots would be given, but does not significantly increase the amount of computation because value are easily associated with slots through "param-order."

The first word of each message can be used to determine if the AUV should continue processing the message. Since the receiver is reasoning about the primary mission task as well as understanding messages from other AUVs, there are cases where its work on the primary task has a high priority or rapidly-approaching deadline. There are also cases in which the message contains an error, and both bandwidth and processing effort would need to be expended to resend the messages. In both situations, if the AUV can determine from the first word whether or not it should continue processing the message, it can avoid a great deal of needless processing. Consequently, urgent messages are marked with a keyword at the front of the message. When an AUV sees this keyword, it continues processing the message. When an intended receiver is specified, the name of the receiver follows the keyword. Similarly, the intended receiver is the first word in **commands** and **requests**. If the agent which receives the message is not the specified agent, it can stop processing. We allow the receiver to make this decision based on processing constraints because sometimes "eavesdropping" can provide useful information.

3.6 Semantics

Semantics refers to the meanings of words in a language and how those meanings are combined to create meaningful messages. The abstraction in basic semantic interpretation is derived from the abstraction hierarchy of the concepts from which the words get their meanings. However, when vehicles refer to individual entities, they can no longer rely on shared knowledge. Instead, COLA must provide a mechanism for creating references to objects that allows for as much abstraction as possible. We discuss the need for abstraction in references to objects and COLA's mechanism for handling reference after a brief discussion of the basics of semantic interpretation in COLA.

3.6.1 Semantic Interpretation in COLA

We have deliberately left the semantics of COLA as uncomplicated and unambiguous as possible. This is accomplished by taking the meanings of most words directly from the shared portion of the vehicles' knowledge representations, as described in Section 3.4.

This simplification makes semantic processing efficient. In most cases, the semantic interpreter constructs frames in the knowledge representation language directly from the results of parsing. The semantic interpreter simply creates a frame that corresponds to the type of message. This frame has slots for the roles of the message type. The frames which fill these roles are created by creating a frame of the type named in the message and filling the slots with the values given. These frames, slots and values are found by a simple lexicon "look-up" which replaces the words of the message with the symbols from the knowledge representation.

In addition to constructing the meaning of the message, semantic processing can check that the message is *semantically well-formed.* For a message to be semantically well-formed, the meanings of the words must fit together. "Colorless green ideas" is a well-known example of an English phrase that is not semantically well-formed. Here, words conflict based on broad classifications in their meaning — no abstract object such as an idea can have a color.

We can use the same principle when interpreting messages in COLA, but define the constraints more narrowly. For example, we can constrain the depth of a "GOTO" command to be less than the crush depth of the vehicle. If a **request** was received which instructed the vehicle to go below its crush depth, an error would occur. We implement this in COLA using constraints on the values of slots in Orca's knowledge representation. If a value cannot be added during semantic interpretation because it violates a constraint, an error is signaled.

Checking these constraints is particularly important because the sender does not necessarily know the constraints on a receiver's actions. Consequently, unlike syntactic errors, these errors occur even if the sender follows the algorithm for producing messages and the message is transmitted without error.[4]

3.6.2 Referring to Entities During Collaboration

Any language must have a way for agents to refer to objects in the world. The expression used to identify the object is called the *reference*, and the object associated with the reference is called the *referent*. In COLA the mechanism for handling reference must work effectively and efficiently within the constraints of the AUV domain. The **ref** syntactic constituent and its associated semantics provide this mechanism.

Since the sender does not have complete information about the receiver, it must be able to specify the pertinent characteristics of the reference and allow the receiver to choose an object with those characteristics that can most efficiently be used to respond to the message. For example, suppose the sender wants a sample of radioactive rock collected. Assuming the receiver can identify radioactivity, the sender can simply ask the receiver to "collect a radioactive rock." This relieves the sender of the burden of identifying such a rock, and leaves the receiver free to choose a rock which it can easily collect at some time during the mission.

Bandwidth must also be conserved in COLA, whenever possible. Since once an object is referred to, it is likely to be referred to again, a short name is created for each object under discussion. This way, the name, instead of the complete description, can be used for future references. These names are analogous to pronouns in English. However, unlike pronouns, they can be created during a mission and used to unambiguously refer to a particular description of an entity. This can be the case for AUVs because their processing capabilities allow them to easily keep track of the created name and associated descriptions.

Because the receiver is free to choose one of several objects, it is important for the AUVs to understand what knowledge about the object is shared (information in the original description) and which is known only to the receiver (other details about the actual object). Without this distinction, the vehicles will not be able to continue to communicate effectively about the object. For example, if the receiver were to say "help me pick up the rock," the sender would not know which cable to pick up. Instead, the receiver must recognize that the sender has only the partial information about the rock that was communicated. So, the sender would need to add identifying information, for example, "help me pick up the rock at location (X, Y, Z)."

These problems are not currently addressed in natural language processing research in a way that can be adapted for COLA. To handle them, we propose a mechanism for handling reference that allows the AUVs to distinguish between the representation of the referent and the information

[4]Strictly speaking, this would no longer be part of semantic processing because many constraints come from specific features of a vehicle instead of constraints on the language. However, since we handle these constraints in the same way that we handle semantic restrictions, we discuss them here.

that has been communication about that referent [19]. The initial reference to an entity has the syntax *desname type class sv* where:

desname is a name created by the sender
type is the type of reference and shows how the reference should be processed
class is the name of the frame giving the class of the object (e.g., "rock" or "fish")
sv a slot-value list that gives the description of the object

For example, to specify any rock with the following characteristics, the sender can communicate the following description:

COLA Message	*English Gloss*
BFU BFR BDG AAH BBE	BFU any rock (substance radioactive substance)

Later references to this description can simply use the **desname** or may add information to the shared description by following the **desname** with an **sv** list. Both sender and receiver create a list of *description items (DIs)* which include the desname and description. When an AUV must find a real-world referent, this referent is pointed to by the DI. By separating the description from the referent in this way, AUVs can identify the features of an entity that are known to both agents without forcing agreement that cannot be guaranteed in the AUV domain.

3.7 Message Generator

COLA's messages are generated using a standard method associated with ATNs [4]. The process begins when Orca sends a goal to the communication module. The message generator first finds the proper message type for that goal. It then replaces the syntactic constituents of that message (and their constituents, etc) with words from COLA's vocabulary. If the grammar expects a keyword, that word is generated. If it expects another syntactic constituent, that constituent is generated by finding the portion of Orca's goal which corresponds to that constituent and generating the appropriate words. If the concept from the goal is part of the shared knowledge, the corresponding word is generated. References to entities are slightly more complicated because the concepts which represent them are not part of the shared knowledge. The generator first finds the nearest class of the object which is part of the shared knowledge. It then compares the reference to that class, keeping track of all slot/value pairs which do not match. These are generated as part of the **des** of the **DI**.

4 Abstraction in COLA

In this section, we present examples of AUVs communicating in COLA. Our examples illustrate how abstraction can compensate for the sender's incomplete knowledge. We begin with abstraction away from the most concrete levels of the machine. This abstraction is provided by even the simplest command languages. We then discuss several specific mechanisms for abstraction that are available only in conceptual languages.

The examples in this section are from our simulator, SMART. Each simulated AUV has a full implementation of COLA, as described above. Each AUV also has a simple version of Orca. This is enough to demonstrate the power of abstraction in COLA. Clearly, since the processing of an abstract language depends on the receiver's ability to reason about its environment, as Orca is further developed, vehicles will be able to handle more abstract messages even more efficiently.

4.1 Low-level instructions

Each vehicle has a set of low-level instructions that can be executed to perform the vehicle's most basic operations. For example, instructions to store data in a register of a computer might be part of this set. In computer science terminology, these are called *machine-level instructions.* There are also similar instructions for other vehicle hardware. For example, there may be an instruction which turns on a thruster. These are called *vehicle controller instructions.* Communicating such low-level commands certainly robs the receiver of the flexibility that abstract communication should facilitate. However, there are times when a human or another AUV needs to control an AUV at this low level. For example, a human may wish to give the AUV machine-level or vehicle controller instructions for testing. Here, COLA can be the interface for exercising the vehicle once it is in the water. Also, another AUV may need to issue low-level instructions if an AUV's higher-level reasoning capabilities fail. This transmits the sender's reasoning instead of relying on the receiver's. Sometimes the mission and the environment are such that the sender knows enough about the receiver to issue these detailed commands. The language needs to include commands at all level of details to give the sender this ability.

Even low-level instructions can be abstracted by COLA, to some extent. Specifically,

- An instruction set differs from vehicle to vehicle because of the exact form of the instructions. However, different instruction sets on different vehicles often perform the same function. For example, two different vehicles may have different low-level encodings for an instruction to turn on thrusters to move forward. These encodings need not be known to the sender, if the instruction to turn on the thruster is included in the language.
- Some errors may be found during semantic interpretation. For example, a command to turn on a thruster may specify Thruster-7 as a parameter. If there is no Thruster-7, this should be handled as an error instead of attempting to execute the command.
- Not all parameters must be specified. This gives the receiver the kind of flexibility described below, even for machine-level instructions.
- The abstraction is useful for humans. If a communication net exists between a mother ship and a group of AUVs, we could allow humans to communicate with this system using COLA. In this case, COLA provides the same advantage as assembly languages, giving the bit-level instructions a symbolic form.

Our examples in this section rely on EAVE's vehicle command set. These commands are macros which specify vehicle controller instructions to be executed. They are the lowest-level EAVE instructions implemented in SMART and, as such, illustrate that low-level instructions implemented on different architectures can be used as part of COLA's vocabulary.

SMART's representation of EAVE's "vert" command is shown in Figure 6.[5] required when the command is issued as part of the vehicle command set. Default values are supplied for all non-mandatory parameters when COLA is implemented for missions at sea. There are also "comm-mandatory" and "param-order" slots which are used when COLA parses the parameter lists of messages.

In our first example, AUV-1 has the goal of AUV-2 executing a "vert" to go to depth 150. This goal is represented in a frame as a **request** for AUV-2 to execute the "vert" action. This frame is sent to the communication module which generates the message:[6]

[5]In the representation shown in our figures, slots in the frame are in capital letters. Slot values appear following the colon after the slot name. Comments appear to the left of semi-colons. In some cases, slots that are used by Orca for bookkeeping but do not affect language processing, have been omitted.

[6]COLA's words can change between missions as the lexicon is rebuilt. For readability we keep a consistent vocabulary for COLA across all of our examples. Different COLA words indicate different representations of the command.

```
^VERT is a frame with the following description:
  NAME:                   VERT
  ISA:                    (^LOW-LEVEL)
  SLOTS:
    ;; the actor which will execute the command
    o ACTOR: -none-

    ;; parameters required for all commands in EAVE's vehicle command set
    ;; any of these paramters could be included in the ^vert command
    o B: -none-
    o BMAX: 6
    o IGNORE-XY: NIL
    o MIN-TO-COMM: 0
    o UMAX: 30
    o VERT-MODE: -none-
    o X: -none-
    o Y: -none-
    o Z: -none-

    ;; parameter information for COLA - depth (z) must be given
    o COMM-MANDATORY: Z
    o PARAM-ORDER: (Z X Y B MIN-TO-COMM VERT-MODE UMAX BMAX)

    ;; the SMART command that can be executed to simulate the vert
    o COMMAND: (MOVE-TO-DEPTH @Z)

    ;; the depth should not exceed the crush depth of the vehicle (500)
    o CONSTRAINTS: ((Z (RANGE 0 500)))
```

Figure 6: Orca's representation of the "vert" command

COLA Message	*English Gloss*
BDJ 150	vert 150

As we can see, none of the non-mandatory parameters need to be given. In addition, the slot name for the mandatory parameter has been omitted from the message. AUV-2's semantic interpreter creates a frame representing AUV-1's goal for AUV-2 to execute a "vert" to depth 150. The event handler of AUV-2's Orca decides to accept this goal and adds it to the agenda. The goal is removed from the agenda, and AUV-2 executes the command to achieve the "vert", moving to depth 150.

Next, AUV-1 has the goal for AUV-2 to execute a "vert" command to move to depth 550. This depth is below the crush depth given in the constraint on the z slot for the command:

COLA Message	*English Gloss*
BDJ 550	vert 550

In this case, message handling fails during semantic interpretation because the constraint on slot z is violated. Because of this failure, AUV-2 does not attempt to execute the message. Once such an error is found, AUV-2 must use further processing to decide to ignore the message, ask the sender to re-send, or negotiate with the sender to find an appropriate value.

4.2 Abstract Command Types

The conceptual hierarchy is one of COLA's most important resources for abstraction. At the lowest level of the hierarchy are the machine level instructions which actually execute the actions. Higher levels of the hierarchy leave out details of how the action will be performed. For example, at the lowest level, vehicle instructions can turn thrusters on and off. Higher level commands would move the vehicle in a particular direction. At a higher level of abstraction, a "goto" command would specify that the vehicle should move to a particular location, but not specify how the vehicle should move. There are also higher-level instructions that are not strictly an abstraction of a single command, but, rather, are created by combining several low-level commands.

When abstract commands are communicated, the receiver determines the details of how the command will be executed. This is beneficial in real-world environments like the ocean for two reasons:

1. The receiver can use its local knowledge to choose the method for executing the abstract command which can be executed most efficiently.
2. The lower-level commands used to execute the instruction are seen as a coherent whole, not as single instructions that can be executed in isolation.

For the examples in this section, we use an abstract "goto" command which is executed by turning on thrusters to move in the x, y, and z directions to get to the proper location. In SMART, these instructions are named "move-along" and take heading and distance as parameters. In each of our examples, AUV-1 has the goal of AUV-2 moving to (300, 200, 50). This goal location is marked in the figures by a circle labeled "goal." At the start of each scenario, AUV-2 is at (150 100 50). It's movements are shown by the white dots.[7] AUV-1 is able to determine AUV-2's location at certain points in the mission. We assume that AUV-1 knows AUV-2's location at the beginning of each scenario.

In Figure 7, AUV-1 has communicated the following messages:

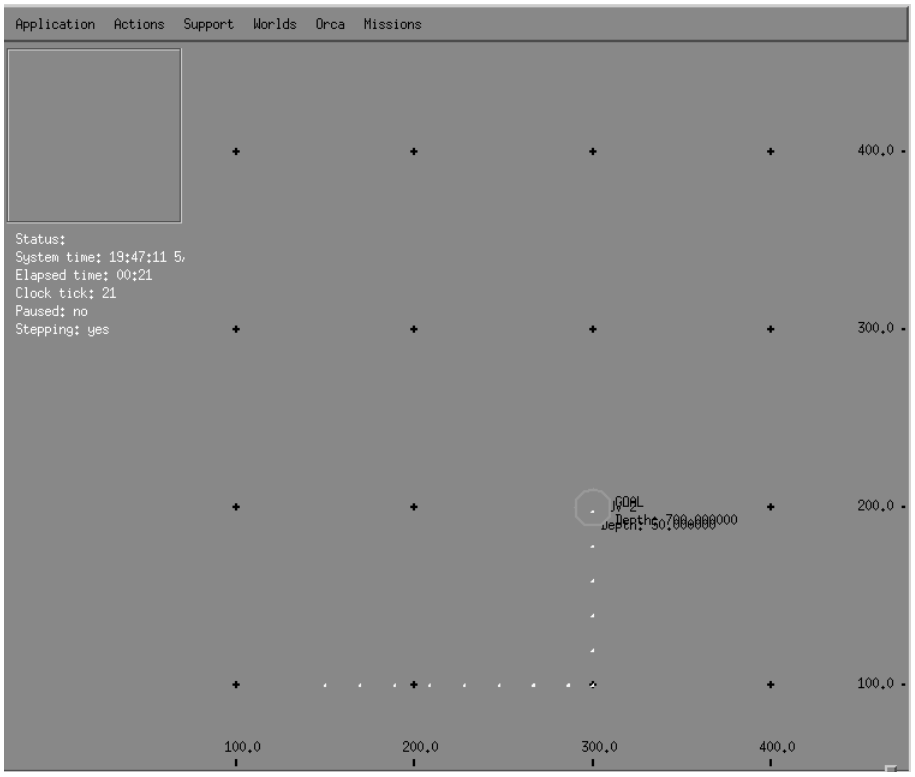
COLA Message	*English Gloss*
AEF 0 150	move-along 0 150
AEF 90 100	move-along 90 100

When AUV-1 has accurate information about AUV-2's location, these messages allow AUV-2 to reach the goal.

In the scenario shown in Figure 8, AUV-1 issues the same low-level commands. In this scenario, however, AUV-2 has a goal of its own which causes it to move to (250 350 50). AUV-1 communicates while AUV-1 is moving to this location. After reaching its goal, it selects actions associated with the messages from the agenda. Because AUV-2 has moved between the time that AUV-1 begins to construct the messages and the time the messages are acted upon, these low-level commands are no longer valid. When AUV-2 receives the messages and executes the actions, it moves to (400, 450, 50) and does not reach the location intended by AUV-1.

In the final scenario in this section, AUV-1 issues an abstract "goto" command. This allows AUV-2 to select the low-level commands to carry out the "goto," so current local knowledge can be brought to bear on these decisions. The scenario is shown in Figure 9. As AUV-2 is traveling to its goal location, AUV-1 sends the following message:

[7] COLA's words can change between missions as the lexicon is rebuilt. For readability we keep a consistent vocabulary for COLA across all of our examples. Different COLA words indicate different representations of the command. The figures are screen dumps from SMART. They have been altered to make labels readable and to add markers to the path to make it more clear.



```
Status:
System time: 19:47:11 5/
Elapsed time: 00:21
Clock tick: 21
Paused: no
Stepping: yes
```

Figure 7: Low-level commands with AUV-2 stationer.

COLA Message	*English Gloss*
AJZ AKD 50 AKE 300 AKF 200	goto-schema z 50 x 300 y 200

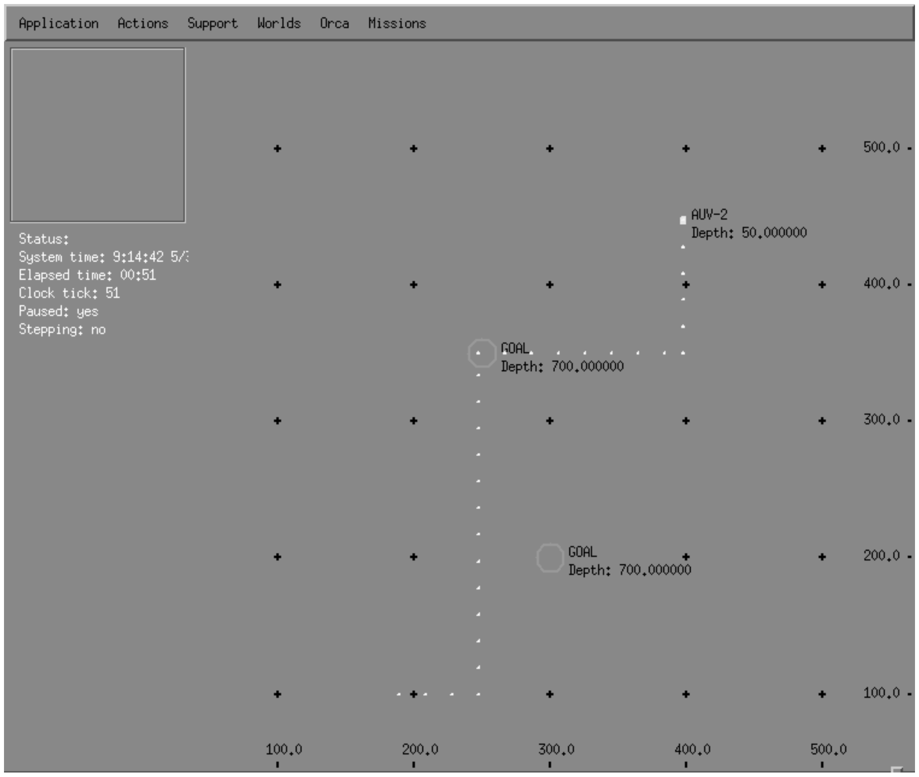
After reaching its goal, AUV-2 selects the "goto" schema associated with the message from the agenda. It expands the schema in the context of its local knowledge, executing the commands "move-along 0 50" and "move-along 270 150" which move it to the location intended by AUV-1.

4.3 Abstracting Parameters

COLA permits the sender to omit specific information about parameter values and, thereby, to further abstract commands. This is a distinct advantage over communicating instructions using traditional command languages that are similar to computer programming languages. In traditional command languages, each command must be entered following a specific syntax. Since this syntactic parsing has severely limited access to semantic information, there is usually very little flexibility in the form of the command. Consequently, parameters must follow a specific order and optional parameters are rarely allowed. In addition, parameter values generally must be fully-specified. In exceptional cases when parameters are not fully-specified, specific forms must be followed for a particular parameter in a particular command. These constraints allow commands in these language to be parsed efficiently and obviate the need for all but the most superficial semantic processing. This is enough to enable a receiver to simply carry out an action.

In contrast, COLA embraces deeper *pragmatic*[8] analysis so that a receiver can fully understand information and commands in the context of its knowledge and its mission. The same robust

[8]Pragmatics concerns factors other than syntax and semantics which give meaning to an utterance.



Status:
System time: 9:14:42 5/
Elapsed time: 00:51
Clock tick: 51
Paused: yes
Stepping: no

Figure 8: Low-level commands with AUV-2 movin.

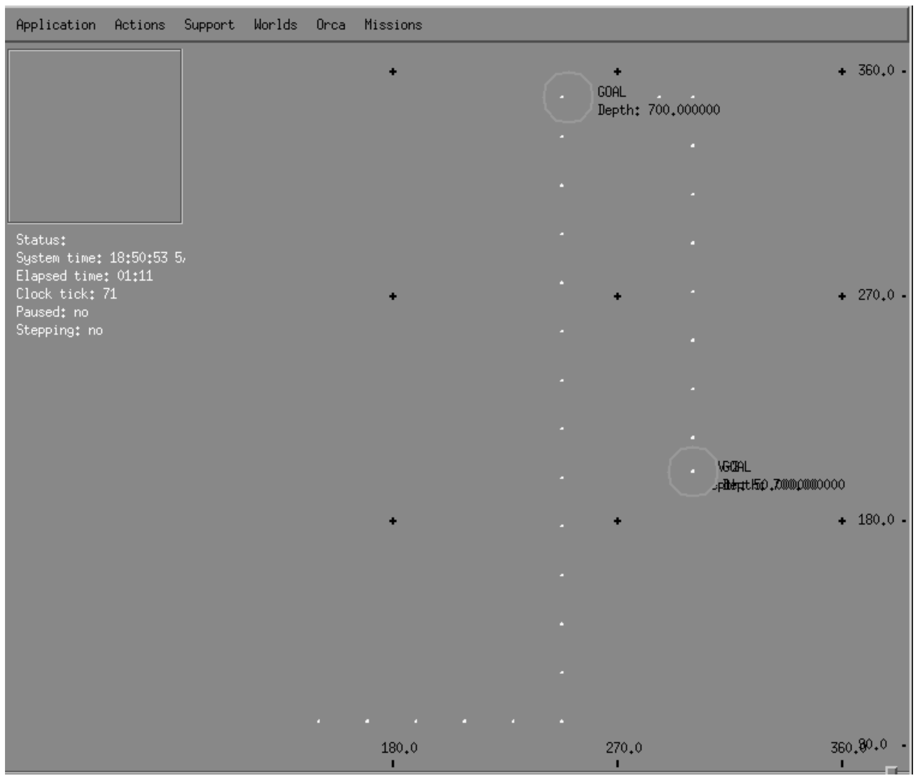
processing applied to the language in general is applied to the interpretation of parameters. This allows parameters to be interpreted in a general way, not according to special linguistic rules governing each command.

The current version of COLA allows parameters to be abstracted in three important ways:

Parameters can be referred to with any interpretable expression. This goes beyond the regular expressions and bound variables that can be used as parameters for commands in programming languages. Using COLA the vehicle is able to interpret a description of the value for the parameter. For example, suppose an AUV wanted to test another's starboard thruster, but it did not know the internal number for the thruster required by the vehicle controller instruction. By using COLA, the AUV could construct a description for the starboard thruster and use that description as the parameter to the communicated command. The receiver interprets the description and matches it to the frame representing the starboard thruster in its knowledge base. Then, when constructing the frame for the interpretation of the entire command, it fills the parameter slot with the number from the thruster frame.

The value of a parameter may be given in an abstract form. This abstract form may indicate a type of object or range of values. For example, "move obj rock to 100 450 500"[9] specifies only the class of the object that should be moved. This allows the receiver to choose any object of type rock to be moved.

[9] "Rock" would be specified in COLA as a complete **ref**.



```
Status:
System time: 18:50:53 5/
Elapsed time: 01:11
Clock tick: 71
Paused: no
Stepping: no
```

Figure 9: Abstract command with AUV-2 moving.

The value of the parameter may not be communicated. In this case, the receiver uses its knowledge about the command to reason about a value to be used or to supply a default value. For this reason, few parameters are deemed "mandatory" in COLA. For example, "goto" allows X, Y and Z parameters, but does not require any of these. For any parameter that is not supplied, the parameter could be interpreted as its value in the current location of the vehicle. So, a vehicle at location (100, 200, 50) and receiving the command "goto x 250 y 200" would travel to (250, 200, 50).

4.4 The Abstraction of Communicating Goals

The abstraction of message types representing the sender's goals in COLA gives the receiver even greater flexibility. In languages without abstraction, requests from a sender have the status of a mandatory action that must be executed immediately. In COLA, the communicated goal of the sender is integrated with the goals of the receiver.

To illustrate the advantage of this abstraction, we use an example similar to the one in Section 2.4. In these examples, AUV-1 has the goal for AUV-2 to rendezvous with AUV-1 at (500, 500, 100). This goal is communicated with the message:

COLA Message	*English Gloss*
AKC AKD 100 AKE 500 AKF 500	goto z 100 x 500 y 500

AUV-2 has the goal to move to (200, 300, 50), shown in the figures by the circle labeled "GOAL."

Figure 10 shows the scenario in which messages are considered as commands that must be executed immediately. After receiving the message, AUV-2 goes directly to (500, 500, 100), but

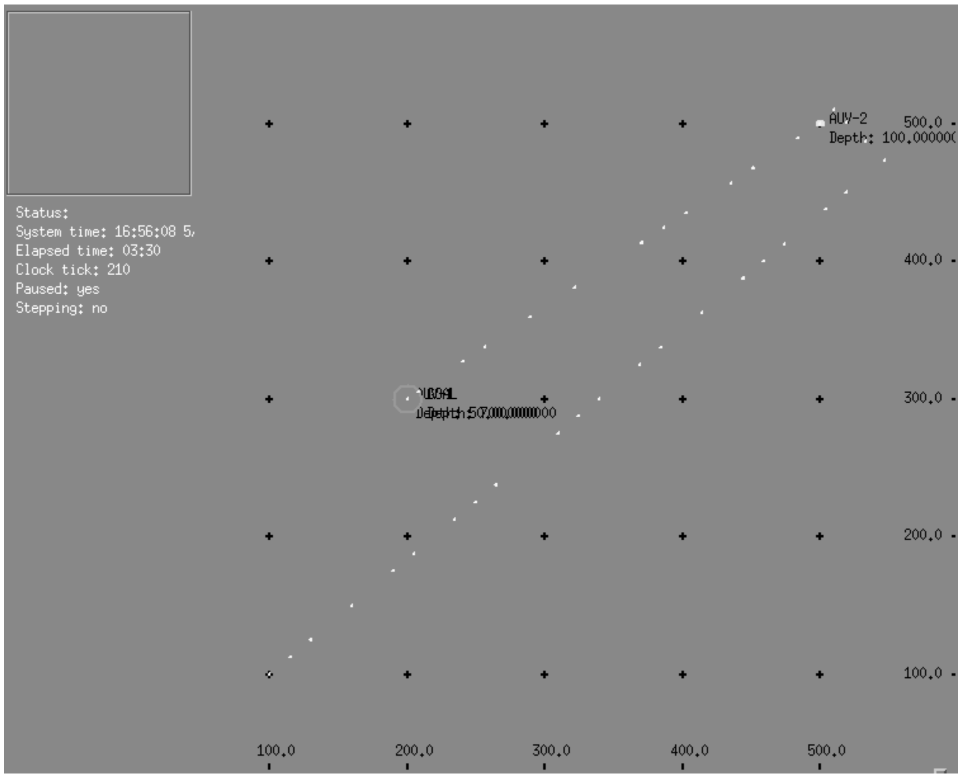

Status:
System time: 16:56:08 5/
Elapsed time: 03:30
Clock tick: 210
Paused: yes
Stepping: no

Figure 10: Communication as mandate

must return to (200, 300, 50) it achieve its own goal.

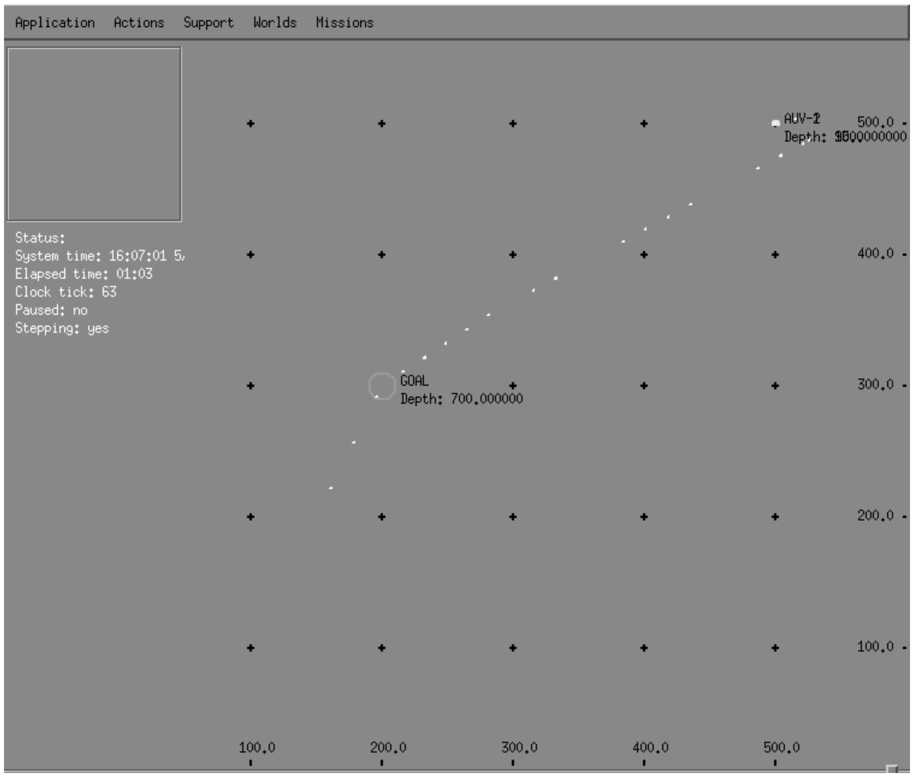
In Figure 11, the message is interpreted as the sender's goal. Orca handles the goal like any other, by placing it on the agenda. Using a simple closest point strategy for choosing goals from the agenda, the receiver first achieves its own existing goal to go to (200, 300, 50). The sender's goal remains on the agenda and is selected next, causing the AUV to move to (500, 500, 100).

5 Other Methods for Multi-vehicle Coordination

The purpose of conceptual communication is to allow AUVs to coordinate their actions so that they can perform collaborative missions efficiently. The abstraction afforded by this type of communication gives the receivers a great deal of flexibility but comes at the price of increased processing and intelligence. There are other methods for coordinating vehicles, each requiring different resources and providing different levels of flexibility. In this section we look at two such methods, *coordination by convention* and *reactive communication*, which have been implemented for AUVs.

5.1 Coordination by Convention

For some missions, the task can be accomplished by assigning certain invariant behaviors to the vehicles. Since their behavior need not change, communication is superfluous. Instead, the vehicles' actions may be coordinated by the engineers that design their programs and cannot be changed during the mission.



```
Status:
System time: 16:07:01 5/
Elapsed time: 01:03
Clock tick: 63
Paused: no
Stepping: yes
```

Figure 11: Communication as intention of sender.

To illustrate, consider a multi-vehicle system which clears mines from a beach [11]. Each vehicle is programmed to follow a specific path from deployment to the shore. Several waves of these vehicles, each with paths covering the area to be cleared, are released. Vehicles in early waves are destroyed close to the place of deployment as they detonate a mine. Later, waves pass these already-detonated mines, and detonate additional mines further from the place of deployment. Given enough waves of vehicles, most of the mines are found and neutralized or destroyed.

In this system, communication can be avoided because each vehicle's behavior is independent of the others. Although the system designers see the vehicles as working as a team, each vehicle performs its assigned actions regardless of the fate of those before it. Consequently, no communicated information or instruction needs to alter the behavior of any vehicle.

We might argue that this mission could be made more efficient if vehicles were able to communicate. For example, suppose a vehicle communicated that it had successfully reached the shore. This would show that its path had been cleared. The vehicles that were to follow it could remain at the deployment point or be used to increase coverage on other paths. The mission could then be conducted using fewer vehicles. This is a false efficiency, however, because it ignores the cost of the vehicles. Any hardware that would allow the vehicles to send and receive signals would significantly increase the cost of the vehicles. It is important to keep their cost as low as possible, because many vehicles will be lost during the mission. The need for low-cost vehicles and the ability to specify the task so vehicles work independently makes this system appropriate for coordination by convention.

This multi-vehicle system illustrates some of the benefits and limitations of coordination by convention. Coordination by convention requires the system designer to foresee all interactions between vehicles. This means that the mission must be simple enough that all such interactions can be defined. In addition, features of the environment that affect the mission must be predictable. Although the designer does not need to predict the exact environment at each moment of the mission, the range of possibilities must be predicted. This is the only way that all possible interactions can be predicted and specified in protocols of behavior that are programmed for the vehicles before the mission begins.

For these systems to be successful, all vehicles must follow the rules. These rules for behavior, then, provide all abstraction in the system. Although, in principle, these rules could be arbitrarily abstract, in practice, they most often specify fairly low-level, inflexible behaviors. This is, in part, because of the complexity of designing highly abstract interactions. It is also because the behavior of the vehicles must be predictable so that interactions and their results can be specified.

The most important advantage of coordination by convention is clear: vehicles which do not communicate are less expensive than those which do. Communicating vehicles must be provided with the hardware and software to produce and interpret signals. During the mission, energy and computational resources are consumed to support communication. When vehicles do not communicate, this overhead can be eliminated. The cost of communication versus the lack of efficiency in execution when general conventions are applied to specific situations is the most important trade-off to consider for coordination by convention.

5.2 Reactive Communication

By some definitions, communication occurs whenever a signal emitted by one agent causes another to modify its behavior. The receiver does not reason about the signal, but simply executes a predetermined response. No further interpretation takes place. We call this *reactive communication* because, like other reactive paradigms in artificial intelligence [e.g., 1; 13], communication is not mitigated by any representation or reasoning.

An automated lobster which has been implemented to follow odor plumes [10] could easily process this form of communication. This lobster modifies its behavior to follow an odor plume. The plume itself provides the signal, and a set of signal response rules cause the lobster to react to that signal. The lobster does not reason about the signal or the response; it simply follows its rules. Once the signal-response pattern has been implemented, any odor plumes with the same salient features will trigger the same response. So, for one vehicle to make another perform some behavior, it simply needs to release the proper odor plume.

To implement reactive communication, system designers must create a language of signals and responses. The signal *triggers* a response, causing the response to be performed. In addition, a mechanism must be provided by the system designer to select which of several available signals will be allowed to trigger its response. Strictly speaking, this must rank signal-response pairs solely by their type. The ranking cannot rely on information about goals of the mission or predictions about the environment.

This language abstracts away from the machine level because the set of signal-response pairs is implemented on each vehicle according to its set of sensors and manipulators. Some additional abstraction is available by parameterizing the response. However, the receiver is given no further flexibility. Reactive paradigms prohibit the use of conceptual knowledge of any kind. This precludes exploiting predictions about future behavior. Consequently, the vehicle cannot reason about the communication within the context of the mission. Once a signal triggers a response, that response must be executed; if more than one signal is recognized, information about the mission cannot be used to select the signal to which it will respond.

Because such a language is interpreted only as a system of signals and responses, there is very little computational overhead for processing this sort of language. Once the signal is recognized,[10] the response is simply executed. Not only is processing time relatively small in reactive communication, but no additional computational machinery is necessary to reason about the communication. Vehicles using reactive communication do not require a representation of the world or any mechanisms for reasoning about this knowledge.

This is also true for reactive architectures for problem solving. If the vehicles have reactive architectures for problem solving, they should use reactive communication. This has the advantage of communication relying only on the reasoning mechanisms needed for problem solving. More importantly, communication works within the reactive paradigm which is often an important research issue for the designer.

Because reactive paradigms do not allow reasoning to include knowledge beyond that which can be directly sensed at a particular time, reactive planning and communication are best used for missions where predictions about future behavior do not benefit current behavior. In some cases, the predictions are not helpful because they are not reliable. In a new environment, or where the world is rapidly changing, features of the environment that are salient to the problem are likely to be unpredictable. There are also cases where the efficiency of the mission is not significantly increased by these predictions. Here, too, the overhead of reasoning about the predictions is not warranted by results. Reactive communication is also useful if the vehicle can continually readjust its behavior in response to the signal. Because there is no overhead of reasoning or problem solving, communication can be handled rapidly. This makes it particularly useful for handling urgent messages.

However, reactive communication cannot handle several important tasks in multi-vehicle systems. Vehicles cannot agree to future actions. This is because they cannot reason about future actions within this paradigm. They are able to synchronize activity only by sending a signal and getting an immediate response. Additionally, vehicles cannot agree to meet at a rendezvous point after performing an assigned task. It is also difficult for another vehicle or person to control a vehicle using this sort of communication.

6 Discussion

We have demonstrated the need for abstraction in communication between AUVs and have presented a language which provides it. Abstraction allows vehicle to overcome several limitations to underwater communications:

- Bandwidth is conserved.
- Computational effort is reduced.
- Flexibility is provided.

This flexibility is important because it allows the receiver to respond to communication within the context of its current situation and future actions. This, in turn, improves the efficiency of executing a response to that communication.

COLA provides this flexibility, as we have shown through several examples. However, to take advantage of the flexibility, the communicators must have a conceptual representation and be capable of reasoning about their goals and actions. This increases the cost of the mission as a whole by increasing the cost of communication. For each multi-vehicle system and each class

[10]The recognition of these signals can, itself, be an expensive process. All vehicles must have the proper sensor suite so that a signal can be recognized. In addition, the vehicle must be focused on these signals. As Hayes-Roth points out for reactive planning [14], identifying the signals has a cost of its own which should not be overlooked.

of missions for which they might be used, system designers must carefully consider the trade-off between the cost of communication and the increased efficiency in executing a response to that communication.

In the future, we plan to look more closely at this trade-off. First, we plan to evaluate COLA on extended missions to determine when COLA is most effective. Second, we plan to integrate COLA and other communication methods in a multi-vehicle system. We are also continuing to develop COLA. We anticipate extending the language as we test it on more and more missions. We also look forward to continuing to redefine the grammar and the lexicon so that messages can be sent using fewer bits. We are also looking at questions more closely. Although we now follow Cohen and Parrault's formulation of questions as requests for informs [9], we plan to explore adding a question message type so that questions can be processed more efficiently. The major extension to COLA, however, will be extending message generation. We are particularly interested in mechanisms which allow the problem solver and COLA to work together to find the best level of abstraction for each concept in the message.

References

[1] P. E. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, Los Altos, California, 1987. Morgan Kaufmann.

[2] Alfred V. Aho and Jeffrey D. Ullman. *Principles of Compiler Design.* Addison-Wesley, Reading, Mass., 1977.

[3] J. F. Allen and C. R. Perrault. Analyzing intention in utterances. *Artificial Intelligence*, 15(3):143–178, 1980.

[4] James Allen. *Natural Language Understanding.* The Benjamin/Cummings Publishing Company, Inc., Reading, Mass., 1987.

[5] J. L. Austin. *How to Do Things with Words.* Oxford University Press, New York, 1962.

[6] D. Richard Blidberg and Steven G. Chappell. Guidance and control architecture for the EAVE vehicle. *IEEE Journal of Oceanic Engineering*, OE–11(4):449–461, 1986.

[7] Alan H. Bond and Les Gasser, editors. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[8] Steven G. Chappell, James C. Jalbert, Paul Pietryka, and John Duchesney. Acoustic communication between two autonomous underwater vehicles. In *Proceedings of the 1994 Symposium on Autonomous Underwater Vehicle Technology (AUV'94)*, Cambridge, Massachusetts, 1994.

[9] Philip R. Cohen and C. Raymond Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3:177–212, 1979.

[10] T.R. Consi, J. Atema, C. A. Goudey, J. Cho, and C. Chryssostomidis. AUV guidance with chemical signals. In *Proceedings of the 1994 Symposium on Autonomous Underwater Vehicle Technology*, pages 309–316, 1994.

[11] C. N. Duarte and D. J. Carlino. Using a fleet of autonomous bottom crawling vehicles for surf zone operations. In *Proceedings of the 1994 Symposium on Autonomous Underwater Vehicle Technology*, pages 317–322, 1994.

[12] P. Naur (ed.). Revised report on the algorithmic language ALGOL 60. *Communications of the ACM*, 6(1):1–17, 1963.

[13] R. J. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 202–206, Los Altos, California, 1987. Morgan Kaufmann.

[14] Barbara Hayes-Roth. Opportunistic control of action in intelligent agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1575–1587, 1993.

[15] Scott J. McCue, Michael J. Carter, and D. Richard Blidberg. An investigation into protocols for underwater communications. In *Proceedings of the 7th International Symposium on Unmanned Untethered Underwater Submersible Technology (AUV '91)*, 1991.

[16] Marvin Minsky. A framework for representing knowledge. In P. Winston, editor, *The Psychology of Human Vision*. McGraw-Hill, New York, 1975.

[17] J. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, UK, 1969.

[18] Elise H. Turner. Exploiting problem solving to select information to include in dialogues between cooperating agents. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pages 882–886, Atlanta, Georgia, 1994.

[19] Elise H. Turner, Brent W. Benson, and Kathleen P. Herold. An approach to reference in communication between cooperating autonomous systems. In *Proceedings of the International Conference on Systems, Man and Cybernetics*, pages 1435–1440, Charlottesville, Virginia, 1991.

[20] Elise H. Turner, Steven G. Chappell, Scott A. Valcourt, and Martin J. Dempsey. COLA: A language to support communication between multiple cooperating vehicles. In *Proceedings of the 1994 Symposium on Autonomous Underwater Vehicle Technology*, pages 309–316, 1994.

[21] Elise H. Turner and Collette M. Matthias. Rating usefulness of information to communicate. Technical Report 94-22, University of New Hampshire, December 1994.

[22] Elise H. Turner and Cathy L. Steffen. A language for cooperating AUVs. In *Proceedings of the 7th International Symposium on Unmanned Untethered Underwater Submersible Technology (UUST '91)*, 1991.

[23] R. M. Turner, D. R. Blidberg, J. S. Fox, and E. H. Turner. Multiple autonomous vehicle imaging system (MAVIS). In *Proceedings of the 7th International Symposium on Unmanned Untethered Underwater Submersible Technology (AUV '91)*, 1991.

[24] R. M. Turner and R. A. G. Stevenson. ORCA: An adaptive, context-sensitive reasoner for controlling auvs. In *Proceedings of the 7th International Symposium on Unmanned Untethered Underwater Submersible Technology (AUV '91)*, 1991.

[25] W. A. Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606, 1970.