Ayllu: Distributed Port-Arbitrated Behavior-Based Control

Barry Brian Werger

Ullanta Performance Robotics
barry@usc.edu, *http://www-robotics.usc.edu/ barry/ullanta*

Abstract. Distributed control of a team of mobile robots presents a number of unique challenges, including highly unreliable communication, real world task and safety constraints, scalability, dynamic reconfigurability, heterogenous platforms, and a lack of standardized tools or techniques. Similar problems plagued development of single robots applications until the "behavior-based" revolution led to new techniques for robot control based on *port-arbitrated* behaviors (PAB). Though there are now many implementations of systems for behavior-based control of single robots, the potential for distributing such control across robots for multi-agent control has not until now been fully realized.

This paper presents Ayllu, a system for distributed multi-robot behavioral control. Ayllu allows standard PAB interaction (message passing, inhibition, and suppression) to take place over IP networks, and extends the PAB paradigm to provide for arbitrary scalability. We give a brief overview of the Broadcast of Local Eligibility, a general technique for constructing such scalable systems in Ayllu, and survey some of the research projects and applications that have been implemented using Ayllu.

Keywords: multi-robot, behavior-based, distributed control, control architectures

1 Introduction: Distributed Behavior-Based Systems

There is a strong trend in the robotics community towards distributed robotic systems - the use of groups of robots rather than single robots for greater robustness, scalability, and economy, as well as to achieve *strongly cooperative* tasks – those which simply cannot be performed by a solitary robot ([14,19,2]. Though it is not uncommon these days to see mobile robots equipped with wireless Ethernet links, there are few that have communications architectures that address the many problems specific to the robotic domain. Some of these are:

- Unreliable communication: Even the most sophisticated radio systems are subject to interference, both loss of bandwidth due to RF noise and competition for bandwidth.

- Real world task and safety constraints: Robotic tasks have time dependencies that are crucial to task performance and system survival. Much information has a severely limited useful lifetime (often measured in hundreths of seconds) which influences how reliability of communication must be addressed.
- Scalability: Much of the motivation for mult-robot research involves economies of scale; the vision of "swarm" robotics, and all group robotics to a lesser extent, is to be able to increase task performance linearly merely by dropping in more robots. A good distributed robotics architecture must allow robots to be efficiently added to the system.
- Dynamic reconfigurability: At least for the present, robots are prone to failure; communication problems mentioned above lead to some robots being unreachable by others for varying periods of time; and scalability factors mentioned above lead to change in the makeup of a robot team. Most applications would benefit from the ability to match available robots to tasks without user programming or configuration.
- Heterogeneity: Robots may be equipped differently, or may differ in reliability of subsystems. Again, most applications would benefit from the ability to automatically match available robots to tasks based on individual robot capabilities, which may vary over time.

The rest of this paper describes AYLLU, an architecture designed to address the issues of distributed robot control. It allows distribution in many senses: distribution of control of one robot over several machines, coordination of several robots by one machine (in effect using them as a single configurable sensor/actuator), and most interestingly, group interactions without centralized control, through either explicit communication or sharing of sensor readings. First we will discuss briefly the port-arbitrated behavior paradigm on which AYLLU is based; we will outline AYLLU's extensions of this paradigm; we will present an AYLLU-enabled technique for building arbitrarily-scalable systems and present an example; and then we will survey a number of successful AYLLU systems.

2 Port-Arbitrated Behavior-Based Control

The behavior-based approach to robot control introduced by Rodney Brooks [4] has been so influential in the field of robotics that the term "revolution" is commonly accepted as a description of its effect. Years after this revolution, the principles of horizontal decomposition, the world as its own model, and "emergent" intelligence (in the eye of the beholder) are Brooks' most salient contributions [1]. A somewhat lesser-known contribution is a set of well-defined abstractions and techniques for behavior interaction, implemented in a number of special-purpose languages which are more flexible successors to the well-known Subsumption Architecture. We refer to these abstractions and techniques as the Port-Arbitrated Behavior paradigm, or PAB.

The Behavior Language [3] has been used in numerous experiments and applications, and has demonstrated scalability of the behavior-based approach to higher levels of competence, particularly in Mataric's demonstration of distributed mapping and "path planning" [9] and social learning [10], and Maes' work in action selection [8]. The MARS/L system [5] maintains the inter-behavior communication methods of the Behavior Language but allows behaviors to be coded with all the facilities of Common LISP. This system has been used successfully in such domains as robot soccer [19], multi-robot learning [11], and human-robot interaction [12].

In these PAB systems, controllers are written in terms of behaviors, which are groups of concurrent processes. Each behavior has an interface shared by all of its processes, some parts of which are accessible from outside the behavior. The externally accessible elements are referred to as *ports*; they are registers that hold a single data item (which may be simple or complex, such as an integer or an arbitrary data structure). Internally accessible parts of the interface are *slots*, which are registers like the ports but accessible only to processes within the behavior, and *monostables*, which are boolean variables which remain true for a specified period of time after they are *triggered*.

Ports in different behaviors are linked together by *connections*, unidirectional data paths between a *source port* and a *destination port*. A port can have any number of incoming and outgoing connections. When data arrives at a port, either written directly from a process in the behavior or indirectly through a connection, it is generally propagated along all of that port's outgoing connections. Such data flow can, however, be modified by connections which are specified to be *suppressive, inhibitory,* or *overriding*. Given a connection $C_{s,d}$ from port s to port d with an associated period p, the following are the effects on d whenever a message m is propagated from s

- If $C_{s,d}$ is *Normal*, then m is written to d, and is propagated along all of d's outgoing connections. p is not specified for Normal connections
- If $C_{s,d}$ is *Suppressive*, then m is not written to d, and for period p, no incoming connections will be able to write to d (that is, for all ports x, any connection $C_{x,d}$ will be temporarily disabled)
- If $C_{s,d}$ is *Inhibitory*, then m is not written to d, and for period p, no messages will be propagated out from d (that is, for all ports x, any connection $C_{d,x}$ will be temporarily disabled)
- If $C_{s,d}$ is *Input Overriding*, then d is *suppressed* for period p, except that *only* messages arriving along $C_{s,d}$ (including m) are written to d and propagated as normal
- If $C_{s,d}$ is *Output Overriding*, then d is *inhibited* for period p, except that *only* messages arriving along $C_{s,d}$ (including m) are propagated outward along all $C_{d,x}$ as normal

It is through these mechanisms of suppression and inhibition that subsumption hierarchies, as well as other forms of arbitration, can be efficiently

and intuitively implemented. Since connections are external to the behaviors, behavior code is easily re-usable, and interaction between behaviors can be modified dynamically. The port abstraction enforces a data-driven approach to programming that "grounds" computation in sensor readings and effector actions. By placing coordination in the interaction between behaviors (connections) rather than in the behavior code, these systems allow complex controllers to be built "bottom-up" from extremely simple, easily testable behaviors. The PAB approach allows a clean, uniform interface between encapsulated system components (behaviors) at all levels that abstracts away many issues of timing and communication; the "black boxes" of behaviors may contain reactive mappings or deliberative planners. While our research focuses on non-deliberative approaches, we believe that PAB interaction between system components can help reduce the complexity of the components themselves, whatever their type.

Unfortunately, the Behavior Language and MARS/L have been limited to specific target and development platforms, and have not seen widespread use. Other systems built to implement behavior-based control have not captured their elegance and ease of inter-behavior message passing, inhibition, and suppression. We believe that this lack of a good, widely-available implementation of the basic substrate of such behavior-based systems is one of the major reasons that the approach is seen to be problematic for scaling (see, e.g., [1]). Furthermore, neither L nor the Behavior Language has facilities for behaviors distributed across robots. The ALLIANCE architecture [15] and BeRoSH [18], among others, provide PAB-style control within robots and communication between robots, but do not generalize communication so that PAB arbitration techniques can be used between behaviors on different robots. We believe that by extending the notion of Brooksian inter-behavior connections across networks of distributed computers (i.e., over IP), we will be able to open up a wide range of new, simple, robust algorithms for multi-robot coordination. AYLLU has been developed for this purpose.

3 Ayllu: Distributed PAB Control

AYLLU is an extension of the C language intended for development of distributed control systems for groups of mobile robots. It facilitates communication between distributed system components and scheduling of tasks. While AYLLU has many features specialized for behavior-based systems, it is useful for development of a wide range of architectures from reactive to deliberative, and provides the means for coordinating processor-intensive tasks, such as high-level planning and vision processing, with responsive low-level control. A small interface, referred to as AYLLULite , can be easily added to non-AYLLU programs, allowing AYLLU to serve as a simple and effective "glue" between heterogeneous system components. It is designed to be highly

portable among operating systems and languages, and allow maximal interoperability.

AYLLU's principal goal is facilitate implementation of robust multi-robot systems that must cope with noisy and range-limited communication, rapidly-changing real-world situations, variations in resource availability, tasks that require redistribution of system resources, and various hardware failures. Towards this goal, AYLLUextends subsumption-style message passing to the multi-robot domain, provides for a wide variety of behavior-arbitration techniques, and allows a great deal of run-time system flexibility including dynamic reconfiguration of behavior structure and redistribution of tasks across a group of robots as determined by either task constraints or changing availability of resources.

The question arises as to whether AYLLU is a language or a library. Strictly speaking, AYLLU may be referred to as a language, since it involves syntactic structures foreign to C itself, and some differences in semantics. AYLLU makes extensive use of C's macro facilities, and as a result the standard C preprocessor is able to translate all AYLLU code into standard C code. Since AYLLUtherefore consists of header files and object code that uses standard C compilation, we refer to it in general as a library, or development environment, rather than a language.

3.1 Ayllu Extensions to the PAB Paradigm

AYLLU extends the PAB paradigm in two primary ways. One is the ability to connect behaviors (and arbitrate between them) over IP networks; the other is the addition of features that provide for scalable systems.

Connections Over Networks The extension across IP is straightforward; when making a connection $C_{s,d}$, s and d can each be specified either as a $(behavior, port)$ pair or as a $(host, behavior, port)$ triple. Within the behaviors themselves, no distinction is made. AYLLU also supports IP broadcasting on a local subnet; if a connection is made to $d = (host, behavior, port)$ where $host$ is specified as a broadcast address, then any message propagated along the connection is sent to port $(behavior, port)$ in every host on the network except the sender. Such broadcast connections can be special (i.e., suppressive, inhibitory, or overriding).

Scalability Features Given the single-item register nature of ports, in previous PAB systems it is not easy (and often not possible) to arbitrarily add new behaviors to a system; unless new ports are added to behaviors, comparisons cannot be made between data coming from various sources. As new robots are added to a system, either behavior interfaces in all robots must be changed so that data sent by the new robots will be available, or an increasing percentage of all robots' data will be overwritten and lost. This does not allow

for flexible group size, or robustness to robot failures. AYLLU addresses this problem through the addition of four specialized port types: when a message $m_{incoming}$ is written to a port p holding a data item $m_{current}$,

- if p is a normal port, $m_{incoming}$ replaces $m_{current}$
- if p is a *MaxPort*, $m_{current}$ becomes $max(m_{incoming}, m_{current})$
- if p is a *MinPort*, $m_{current}$ becomes $min(m_{incoming}, m_{current})$
- if p is a *MinPort*, $m_{current}$ becomes $m_{incoming} + m_{current}$
- if p is a *PriorityPort*, $m_{current}$ becomes $m_{incoming}$ iff $priority(m_{incoming}) > priority(m_{current})$

As we will demonstrate in Section 4.1, even the MaxPort alone is sufficient for implementation of arbitrarily-scalable group coordination strategies.

Write-Inhibition AYLLU also adds one new type of connection:

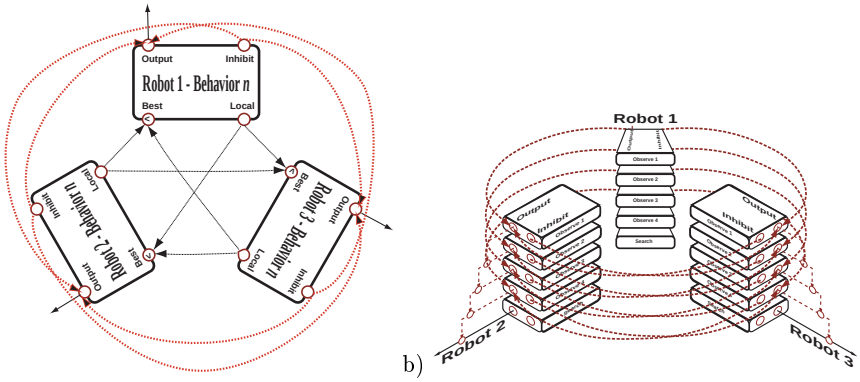
- If $C_{s,d}$ is *Write-Inhibitory*, then m is not written to d, and for period p, no messages written directly from *within the behavior* will be propagated out from d, though messages arriving *through connections* will be

This provides a "closure" of the control of information flow through ports, and allows construction of multi-dimensional arbitration structures such as the *cross-subsumption* we present in Section 4.1.

4 A Scalable Example: Multi-Target Observation

We now present an example of how effective multi-robot strategies can be implemented with a small number of connections and extremely simple component behaviors using AYLLU. This example is a system that performs the W-CMOMMT [20,15] cooperative multi-robot observation of multiple moving targets task. In W-CMOMMT, the goal is for a group of robots to keep a number of moving targets under observation (that is, in the field of view of the robot and within a meter or so). Each target has a priority, so that when there are more targets than robots, the highest-priority targets should be covered.

In order to construct a system that will maintain observation of the highest priority targets and make sure that robots are properly distributed to separate targets (rather than redundantly observing some targets while leaving others uncovered), while adapting to changing numbers of robots and targets and robot failures, we use a technique called the Broadcast of Local Eligibility. Our BLE approach to W-CMOMMT is thoroughly detailed in [20].



a) b)

Fig. 1. *a) Cross-Inhibition*: A cross-inhibited peer group. *b) Cross-Subsumption*: The structure of a cross-cubsumptive system created by the combination of local subsumption with cross-inhibition. Some lines are omitted for clarity; each "layer" is connected as in a).

4.1 Broadcast of Local Eligibility (BLE)

The BLE mechanism involves a comparison of locally determined eligibility with the best eligibility calculated by a peer behavior on another robot. A When a robot's local eligibility is best for some behavior B_n which performs task T_n, it inhibits the peer behaviors (that is, behaviors B_n) on all other robots, thereby "claiming" task T_n. Since this inhibition is an active process, failure of a robot which has claimed a task results in the task being immediately "freed" for potential takeover by another robot.

In order for a behavior to participate in BLE arbitration, AYLLU implicitly declares that all behaviors have normal ports named *Local* and *Inhibit* , and a MaxPort named *Best* .

Cross-Inhibition of Behaviors Cross-inhibition refers to the process of arbitration between *peer behaviors,* instances of the same behavior on different robots. Given that there is some behavior instance B_n (which performs task T_n) on each robot, cross-inhibition results in the selection of at most a single robot to perform T_n. The selected robot is the one that is most eligible (according to local criteria) for the task.

As illustrated in Figure 1a, the *Local* port of each robot's behavior B_n broadcasts a locally-computed eligibility estimate to the *Best* port of each other robot's behavior B_n. Each *Best* port maintains the maximum of the eligibility messages it has received in the current cycle. Whichever robot has a local eligibility better than or equal to the *Best* it receives writes to its Inhibit port, causing inhibition of behavior B_n in the other robots.

Cross-Subsumption Cross-inhibition arbitrates only between peer behaviors on different robots; some local mechanism must arbitrate between different behaviors on the same robot. Simple subsumption, when combined with cross-inhibition, is sufficient for flexible, scalable, and robust team cooperation in CMOMMT. We call the combination of cross-inhibition and local subsumption *cross-subsumption*.

In cross-subsumption, each robot has a local subsumption hierarchy. Each layer of this hierarchy may be cross-inhibited (as in Figure 1a), resulting in a system similar to the one diagrammed in Figure 1b. As a result, each robot is controlled by its behavior B_n which has the highest priority of any behavior which is generating output; a behavior that is not generating output fails to do so either because its current input is unsuitable for the task (e.g., some necessary object is not in the field of view), or because its output is cross-inhibited. Thus, each robot claims the highest-priority task that it is most suitable for.

4.2 Implementation

Figure 2 shows all of the AYLLU code necessary for the CMOMMT task; the descriptions below refer to behaviors mentioned therein.

Common Behaviors A single behavior (VEL) on each robot controls translational motion to maintain a safe velocity based on the distance to sonar-detected obstacles. The task-oriented behaviors specified below only control rotational motion of the robots. Two classes of behavior are implemented:

Observer behaviors: the target-observing behaviors (OBS1 .. OBS5) rotate the robot so that a specific target is centered in its field of view. This, combined with the common velocity control behavior, causes the robot to approach a specific target and maintain a distance of approximately 1 foot.

Search behavior: the search behavior is a random wander (WANDER).

BLE Coordination: The BLE controller is a subsumption hierarchy of instantiations of the *Observer* behavior, with the Target 1 observer (OBS1) having highest priority and the Target 4 observer (OBS4) having the lowest. Each is then joined (by broadcast connections) into a cross-inhibiting peer group which consists of *Observer*s of the same target on each robot (connected as seen in Figure 1a), producing a cross-subsumption hierarchy (as seen in Figure 1b. The local evaluation function for each *Observer* behavior is proportional to the height of its associated target in the visual field – an approximation of distance. The highest-priority behavior that is not cross-inhibited controls the robot - that is, each robot approaches and tracks the highest-priority target it sees that is not being observed by another robot.

```
#include <ayllu.h>
#include <ayllustd.h>         /* for MakeWanderBeh and MakeSafeVelBeh */

ayDefBehaviorClass(ObserveTarget) {
  ayINTERFACE {
    ayIntPort(TargetX, 0);
    ayIntPort(TargetY, 0);
    ayIntPort(Rotate, 0); }
  ayPROCESSES {
    ayInitProcess(TurnToTarget, ratepersecond(20));
    ayInitProcess(BLE_Select, ratepersecond(1)); }
}

ayDefProcess(TurnToTarget) {
  ayLocalPort TargetX, TargetY, Rotate, BLE_Local;
  ayWriteIntPort(Rotate, ayVISCENTERX - ayReadIntPort(TargetX));
  ayWriteIntPort(BLE_Local, ayMAXVISY-ReadPort(TargetY));
}                            /* BLE_Local is local eligibility estimate */

#define MakeObserver(num, next)                                      \
  ayInitBehavior(Observer, OBS##num);                                \
  ayConnect(VISION, Target##num##Angle, OBS##num, TargetX);          \
  ayConnect(VISION, Target##num##Bottom, OBS##num, TargetY);         \
  ayOverrideOut(OBS##num, Rotate, next, Rotate, seconds(0.2));       \
  ayBroadcastConnect(OBS##num, BLE_Local, Peers, OBS##1, BLE_Best); \
  ayBroadcastInhibitWrite(OBS##num, BLE_Inhibit,                     \
                          Peers, OBS##num, Rotate, seconds(0.5));

void main () {
  ayBROADCAST Peers = ayMakeBroadcast("10.255.255.255");
  ayInitPioControl("/dev/ttyS0");         /* for Pioneer mobile robot */
  ayInitIPComms();

  MakeSefeVelBeh(VEL);
  MakeWanderBeh(WANDER);
  MakeObserver(5, WANDER); MakeObserver(4, OBS5); MakeObserver(3, OBS4);
  MakeObserver(2, OBS3); MakeObserver(1, OBS2);

  ayRunBehaviors();
}
```

Fig. 2. AYLLU code for CMOMMT

4.3 Discussion

As detailed in [20], a slightly more complicated version of this BLE approach to CMOMMT implemented for three Pioneer robots and four targets has proven to be effective in properly assigning robots to targets. Although robots switch off observation of particular targets, coverage of the highest-priority targets was well-maintained, and the system took advantage of the occasional (due to patterns of motion) ability of robots to observe more than one target time such that the three robots tended to cover all four targets when possible. Further, the task division achieved had interesting beneficial effects that tended to reduce problems of sensor uncertainty and physical interference between the robots.

5 Other Ayllu Systems in Practice

In addition to the partial survey of research projects below, AYLLU has been used in real-world applications such as museum installations, film special effects, and roaming sales exhibits at trade shows; in thatrical works and performance art pieces; and in a number of robot competitions including AAAI's and RoboCup Robot Soccer competition.

AYLLU has been used to implement the cooperative behaviors for ground-based robots which cooperate with an autonomous robot helicopter [17] and formation control in outdoor environments. [6] uses AYLLU to build topological mapping systems that run on one robot that communicates with desktop displays, and for work on cooperative mapping with heterogeneous robots. [16] has used AYLLU as a substrate for both fuzzy-logic and multiple-objective behavior control. It is being used in experiments involving artificial emotions in social robotics in continuation of research presented in michaudvuICRA99, and in work on planar object manipulation [7]. [13] uses /Ayllu/ in experiments in which robots learn about cooperation through analysis of the benefits of interaction with humans.

References

1. R. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.
2. Ronald C. Arkin and Tucker Balch. Cooperative multiagent robotic systems. In David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, pages 277–296. AAAI Press/The MIT Press, Cambridge, MA, 1998.
3. R. A. Brooks. The behavior language; user's guide. Memo 1227, MIT AI Lab, April 1990.
4. R. A. Brooks. *Cambrian Intelligence*. MIT Press, 1999.
5. R. A. Brooks and C. Rosenberg. L - a common lisp for embedded systems. In *Proceedings of the Lisp Vendors and Users Conference*, 1995.
6. D. Dedeoglu, M. J. Mataric, and G. S. Sukhatme. Incremental, on-line topological map building with a mobile robot. In *Sensor Fusion and Decentralized Control in Autonomous Robotic Systems, Proceedings of SPIE*, pages 123–139, 1999.
7. Brian Gerkey and Maja Mataric. Murdoch: Publish/subscribe task allocation for heterogeneous agents. In *Proceedings of Autonomous Agents*, 2000.
8. Pattie Maes. Situated agents can have goals. *Robotics and Autonomous Systems*, 6(1–2):49–70, 1990.
9. Maja J Matarić. Navigating with a rat brain: A neurobiologically-inspired model for robot spatial representation. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats*. MIT Press, 1990.
10. Maja J. Matarić. Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2–3):323–336, 1997.

11. Francois Michaud and Maja J Mataric. Representation of behavioral history for learning in nonstationary conditions. *Robotics and Autonomous Systems*, 29(2):1–14, 1999.
12. Francois Michaud and Minh Tuan Vu. Managing robot autonomy and interactivity using motives and visual communication. In *Proc. Conf. Autonomous Agents*, pages 160–167, 1999.
13. Monica Nicolescu and Maja Mataric. Learning cooperation from human-robot interaction. submitted to DARS2000.
14. L. E. Parker. Behavior-based cooperative robotics applied to multi-target observation. In R. Bolles, H. Bunke, and H. Noltemeier, editors, *Intelligent robots: Sensing, modeling, and planning*. World Scientific, 1997.
15. L. E. Parker. Alliance: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14, 1998.
16. Paolo Pirjanian and Maja Mataric. Multi-robot target acquisition using multiple objective behavior coordination. In *IEEE International Conference on Robotics and Automation*, San Francisco, April 2000.
17. G. S. Sukhatme, J. F. Montgomery, and M. J. Mataric. Design and implementation of a mechanically heterogeneous robot group. In *Sensor Fusion and Decentralized Control in Autonomous Robotic Systems, Proceedings of SPIE*, pages 111–122, 1999.
18. Z.-D. Wang, E. Nakano, and T. Matsukawa. Designing behavior of a multiple robotic system for cooperative object manipulation. In *Proceedings of the International Symposium on Microsystems, Intelligent Materials, and Robots*, 1995.
19. Barry B. Werger. Cooperation without deliberation: A minimal behavior-based approach to multi-robot teams. *Artificial Intelligence*, 110:293–320, 1999.
20. Barry Brian Werger and Maja J Mataric. Broadcast of local eligibility for multi-target observation. submitted to DARS2000.