Integratating Robotic Technologies with JavaBots*

Tucker Balch and **Ashwin Ram**
College of Computing
Georgia Institute of Technology

What's Holding Back Robotics Research?

Mobile robotics research advances through developments in theory, and implementation in hardware and software. While theory is important, this article is primarily concerned with hardware and software technologies. It is our view that significant strides in robot performance can be made just by combining existing hardware and software tools. Thus the focus of this paper is answering the question: how can we more easily integrate robotic technologies?

The typical research robot is a hodge-podge of sensor and actuator components from several vendors integrated in a common computing resource. Control system software accesses hardware through software APIs provided by the hardware developers, and "soft technologies" through their associated APIs. The focal point of integration is therefore the control system software.

The problem with most control system implementations, from the standpoint of reusability and integration, is their committment to a robot platform and/or robot architecture. It is rather difficult for instance, to move a proven control system from one robot to another, because the APIs for sensing, steering and motor commands are often quite different in syntax and semantics on different robot hardware. Portability problems may extend to the software components of robotic systems as well: extracting the path planner from a robotic system, for example, is just as challenging as porting a control system.

Therefore, to advance intelligent mobile robotics in the *"Next Leap"*, we must emphasize integration of technologies from the multiple sub-areas of our field. We should build systems by integrating, rather than re-inventing technology. Software frameworks for combining behavior, learning, planning, sensing and actuation should be *inclusive* rather than constrained to a particular robot architecture or hardware platform.

We argue that the most effective approach is through standardized interfaces (APIs) to robotic hardware and soft technologies (e.g. path planning toolkits). JavaBots is an example framework that provides this kind of integration in simulation and on robot hardware. A high-level common interface to sensors and actuators allows control systems to run on multiple simulated and real hardware platforms. Conversely, JavaBots supports the evaluation of competing control systems on the same hardware. In the rest of this article we briefly describe JavaBots, and provide examples of robot systems built using it.

JavaBots

JavaBots is a new system for developing and executing control systems on mobile robots and in simulation. Individual and multi-robot simulations are supported (including multiple robot types in the same simulation). The JavaBot system's modular design enables researchers to easily integrate sensors, machine learning and hardware with robot architectures. It is being utilized in several ongoing research efforts, including RoboCup robot soccer, foraging robot teams and a "robotic pet" development effort.

JavaBots was initially developed as a simulation tool for investigating robot soccer (Figure 1) (Balch 1997c; 1997b). The ease with which behaviors can be designed and tested, and the flexibility of the simulation environment lead us to consider the system for use in a foraging task on mobile robots (Figure 2). In this project, two Nomadic Technologies' Nomad 150 robots were entered as a multi-robot team in AAAI's 1997 Mobile Robot Competition. In addition to their holonomic drive and ultrasonic sensors, the stock Nomad 150s were equipped with simple grasping manipulators and vision. The project was completed in less than three months, with the robots going on to place first in the "Find Life on Mars" event. The compressed time in which the development took place speaks to the integrative advantages of the Java-based system.

We have recently begun a new project to develop a robotic pet. The long-term objective of this research is to build an intelligent, adaptive, user-friendly physical agent. In order to accomplish these objectives,

*Published in *Working Notes of the AAAI 1998 Spring Symposium.*

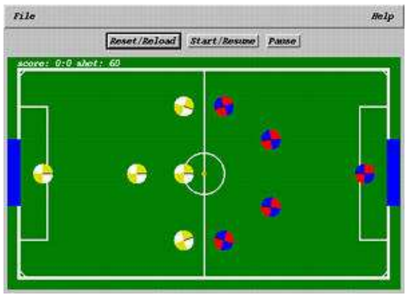

Figure 1: A simulated RoboCup small-size league game running in the JavaBot simulator.

Figure 2: These Nomadic Technologies' Nomad 150 robots, Lewis and Clark, won the multiagent "Find Life on Mars" event at the AAAI-97 Mobile Robot Competition using the JavaBot system.

we need to build an agent that has a personality, interacts with its user on an affective level, and learns about its environment, in addition to autonomously and spontaneously pursuing its own goals. An ISR Pebbles (Figure 4) has been selected as the hardware platform for this work. It is equipped with a tank-like differential-drive suspension, sonar, IR and bump hazard sensors and vision. JavaBots enables us to integrate demonstrated navigational behaviors (developed for the AAAI Competition) with new research in synthetic agent "emotion." More details on this ongoing project will be presented at the Symposium.

System Design

JavaBots is not a robot architecture. It is, rather, a set of Java classes and APIs that bridge robotic software components together. One of those components, of course, is the robot architecture used in the development of a control system. JavaBots includes the robot architecture Clay (derived from Arkin's AuRA (Arkin & Balch 1997)) even though the researcher is free to use his own (Balch 1997a).

The robot control system interfaces with real and simulated robot hardware through a common API (Figure 3). The API provides access to robot sensors and actuators through a basic set of accessor methods. Any control system using the interface is automatically

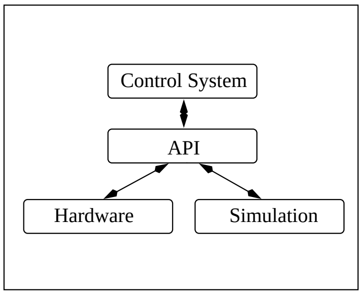

Figure 3: The same control system can run on hardware and in simulation since a single API describes the interface to both.

supported in simulation and on mobile robots.

Several types of robots are supported, including stock Nomad 150s, Nomad 150s outfitted with grippers and vision, RoboCup soccer robots, and ISR's Pebbles. All the robots share a common subset of interface methods. Additional methods are added when a specific platform offers a new or unique sensor or actuator (e.g. vision or a manipulator). A few examples of the common methods include:

- `getHeading()` returns the heading of the robot.
- `getPosition()` returns the location of the robot.
- `setSpeed(s)` move the robot on it's current heading at s meters/sec.
- `setSteer(h)` steer the robot in the h direction.

A new platform can be supported by simply implementing the API for that robot. This means that, not only can the same control system run in simulation and on hardware, but it can run on different types of robot as well.

Why *Java*Bots?

One of our goals in developing JavaBots is to provide a stable, portable platform for the robotics research community. We hope JavaBots will serve as a common platform for the exchange and evaluation of new architectures, learning techniques and other robot technologies. The choice of Java as the language for this system is a crucial design decision supporting our goals:

1. **Portability:** JavaBots runs under Windows 95, NT, Solaris, SunOS, Linux and IRIX as well as several embedded platforms without an OS. All other robotic development environments we know of are firmly tied to specific operating systems. This is an important issue when one considers the wide range of platforms in use by robotics researchers.
2. **Productivity:** It's our experience that programmers produce working code much more quickly in Java than in C or C++. In fact, the present JavaBots distribution was developed in less than nine

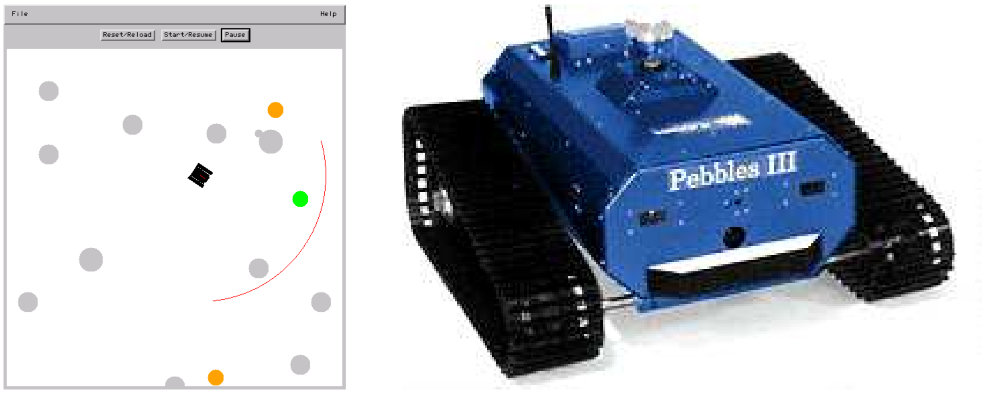

Figure 4: JavaBots also supports simulation of (left) and hardware runs on the ISR Pebbles robot (right).

months. Additionally, the object-oriented features of Java provide for code re-use.

3. **Modularity:** Packages supporting reinforcement learning, motor schema-based navigation, vision, hazard sensing and manipulation have all been integrated and reused in each of our projects.

The significant benefits afforded in items 1 and 2 stem directly from the choice of Java as a programming language.

What about speed? Since Java is interpreted, many conclude it isn't fast enough for robotics applications. In contrast, we have experienced no problems in the performance of our behavior-based control system. In simulation on a Pentium 200, ten agents can be simulated and animated with full double-buffering at 40 Hz. On mobile robots the system, including vision sensing, sonar sensing and motor control, runs at 10 Hz on Nomad 150 robots. The primary bottleneck is serial I/O to vision and robot sensor hardware; each sensor cycle takes nearly 100 milliseconds. It is also important to mention that many new Java compilers (including those available for Windows 95 and IRIX) generate native machine code. Executables run directly on the hardware.

What about garbage collection? In Java, the programmer doesn't have to worry about memory allocation or disposal, instead, the language handles this automatically through periodic garbage collection. This is a key factor in its ease of use, but a potential obstacle to predictable real-time performance. Garbage collection can happen at any time and may take several hundred milliseconds. Our solution to this potential problem is to explicitly call for garbage collection on every behavioral execution cycle. In simulation, overall performance degrades by about 10%, but cycle times never fluctuate. On mobile robots there is no measurable change in performance.

The *Next Leap* for JavaBots

JavaBots will continue to evolve in the future. One of our highest priorities is supporting integration with contributed software and hardware packages. We look forward to the the acceptance of standardized APIs for sensing, learning, planning and other robotic technologies, as these will provide the best avenue for technology interchange. We invite researchers to download and evaluate JavaBots from the world-wide-web (`http://www.cc.gatech.edu/~tucker/JavaBots`).

Acknowledgments

JavaBots was initially developed in Georgia Tech's Mobile Robot Laboratory. We are indebted to the Laboratory's Director, Ronald Arkin, for providing access to robots and computing equipment used in this work. Funding for the Robot Pet project was provided by Yamaha Motor Corporation.

References

Arkin, R., and Balch, T. 1997. Aura: principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence* 9(2). to appear.

Balch, T. 1997a. Clay: Integrating motor schemas and reinforcement learning. College of Computing Technical Report GIT-CC-97-11, Georgia Institute of Technology, Atlanta, Georgia.

Balch, T. 1997b. Social entropy: a new metric for learning multi-robot teams. In *Proc. 10th International FLAIRS Conference (FLAIRS-97)*.

Balch, T. 1997c. Learning roles: Behavioral diversity in robot teams. In *AAAI-97 Workshop on Multiagent Learning*. Providence, R.I.: AAAI.