Software Systems Facilitating Self-Adaptive Control Software

Theodore Q. Pham Kevin R. Dixon Jonathan R. Jackson Pradeep K. Khosla
{telamon,krd,jrj,pkk}@cs.cmu.edu

Institute for Complex Engineered Systems and
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Self-adaptive control software is a new paradigm to create robust, fault-tolerant mobile robots. This type of software analyzes its performance and dynamically modifies itself to operate better in adverse and rapidly changing conditions. We have created two systems that facilitate the creation of self-adaptive control software: PB3A and RAVE. PB3A, the Port-Based Adaptable Agent Architecture, is a mobile, agent-based framework that allows software to adapt itself at all levels. RAVE, the Real And Virtual Environment, is a mixed-reality simulation environment for mobile robots. Together, these two systems allow for the creation, testing, and analysis of self-adaptive control software by on- and off-line simulation. In this paper, we give brief overviews of PB3A and RAVE and present applications that demonstrate robotic systems using self-adaptive control software.

1 Introduction

In the past few years it has become imperative for robotic systems to become more robust. Mobile robots are being charged with handling dangerous materials and exploring distant locations on this planet, as well as others. As these systems are conferred with increased autonomy, their control software must be able to overcome changing conditions as well as hardware and software failures. We address the problem of designing robust, fault-tolerant mobile robots by endowing them with self-adaptive control software. Self-adaptive software analyzes its performance and takes action to bring the performance to a satisfactory level. Using several novel approaches, our control software can diagnose, overcome, and adapt to adverse and rapidly changing conditions. To this end, we have coupled a distributed, mobile multi-agent framework with a mixed-reality simulation environment.

In the monolithic programming model, increasingly capable systems require increasingly complex software. Multi-agent systems achieve capability through complex interactions, not complex software. This philosophy extends to the fault-tolerance domain. More robust systems using the monolithic model require more complex software, while systems using multi-agent paradigms can achieve robust behavior through the inherent modularity and reconfigurability of multi-agent systems. However, most implementations of multi-agent systems do not take advantage of this modularity and reconfigurability because they depend too heavily on the foresight of the author at design time. Reconfiguration is typically a time-consuming manual process that often involves changes to the components themselves. The creation of a general multi-agent software architecture that can learn from its own interactions with the world, evaluate its performance, and adapt itself to achieve its goals better would find natural use in the fault-tolerant robotics arena. With this in mind, we have created the Port-Based Adaptable Agent Architecture (PB3A).

In order to focus on algorithms and architectures for robust behavior, we have also created RAVE, a framework that provides a Real And Virtual Environment for running multiple mobile-robot systems. RAVE provides a unique set of capabilities to facilitate development of such systems. To have multiple-robot systems be developed and tested in simulation and then seamlessly transferred to real robots, RAVE allows any robot controller to be run on either a real or simulated robot. RAVE permits off-line simulation so that the control software can be evaluated before being deployed in a robot, or changes to control software can be simulated on-line, in real-time, and then deployed into the real robot. RAVE also allows virtual ob-

stacles to be added to the world model. This is necessary for running a system in simulation, but it also facilitates real robots to operate in virtual or partially virtual environments. These simulation capabilities allow real and simulated robots to operate simultaneously and interact with each other. RAVE provides a communications package that allows system components to communicate with each other across different computers on a network; RAVE can run entirely on a single computer or can be distributed over many computers.

In this paper, we describe the Port-Based Adaptable Agent Architecture in Section 2 and the Real and Virtual Environment in Section 3, give the three levels of self adaptation in Section 4, discuss issues surrounding RAVE and PB3A as the method for creating self-adaptive control software in Section 5, present applications using this methodology in Section 6, consider related work in Section 7, and lay out conclusions and future work in Section 8.

2 Port-Based Adaptable Agent Architecture

The Port-Based Adaptable Agent Architecture (PB3A) is a Java-based programming framework that aims to facilitate the development and deployment of self-adaptive, distributed, multi-agent applications. Unlike sequential programming models that require an application to be a single stream of instructions, PB3A utilizes a threaded programming model allowing simultaneous streams of instructions. To exploit the power of PB3A, a solution to a problem must be decomposed into a hierarchy of interconnected tasks. We consider a task to be some flow of execution that takes zero or more inputs, produces zero or more outputs, and may modify some internal state. We call these input and output points "ports", and refer to a fundamental unit of execution as a Port-Based Module (PBM).

In essence, a PBM clearly defines the boundaries, entry points, and exit points of the smallest unit of code in PB3A. Each PBM has zero or more input ports, zero or more output ports, and possibly some internal state (see Figure 1). All ports are typed in the typical object-oriented programming (OOP) paradigm. A link is created between two PBMs by connecting an input port to an output port while obeying the OOP rules of inheritance. A configuration can be legal if and only if every input port in the system is connected to at most one output port, however output ports may remain unconnected. An output port may map to multiple input ports. PBMs use a localized, or encapsulated, memory model. All state variables specific to an instantiation of a particular PBM, as well as all methods and static members, are contained within the PBM itself. This allows a PBM to be self-governing and independent of other PBMs.

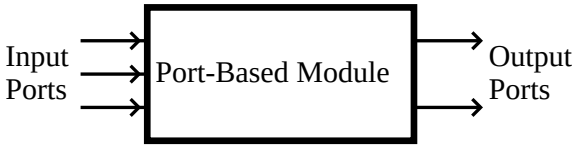

Figure 1: A generic diagram of a Port-Based Module.

PB3A provides the necessary features to facilitate the creation of self-adaptive, distributed, modular applications. All PBM code can be dynamically loaded on demand across a network. This ability allows for newly developed software modules to be quickly integrated into running applications with minimal down-time or disruption. PB3A also provides a uniform port-based communication interface to all PBMs regardless of location. While port mappings between PBMs located on the same computing node are established through shared-memory references, PB3A provides replication mechanisms to establish cross-network port mappings for PBMs residing on different nodes. This frees the PB3A programmer to concentrate on the information being shared, instead of on what mechanisms should be used to perform the information sharing. Additionally, changing the structure of an executing PB3A system is achieved by loading new PBMs and simply remapping ports. Finally, the self-contained nature of the PBM, coupled with its completely specified port-mapping dependencies, allows not only for easy distribution and coordination of code modules onto a network of computers, but also for those modules to be mobile. More succinctly, PBMs can migrate from one computing node to another during their execution.

To aid in building complex systems, PB3A provides three extensions to the PBM. First, recognizing that the definition of most tasks is recursive, PB3A allows for the creation of Macros. A Macro is a PBM that is itself composed of an interconnected collection of PBMs or other Macros. Macros allow a PB3A application to be designed and adapted at multiple levels of abstraction. Second, where the PBM represents the most basic unit of execution, the Port-Based Agent (PBA) represents the most basic unit of self reconfiguration and self adaptability. In other words, a PBA is the smallest unit of code that can analyze its performance and take steps to improve that performance. These steps may include internal parameter tuning, moving across processing nodes, spawning other PBAs, being replaced by a more suitable PBA, or internal reconfiguration of the PBA itself. Thus, the self-adapting PBA is the cornerstone of our approach to managing software complexity. Finally, though the PB3A is a high-level programming framework, low-level interactions must be considered. Since most incarnations of Java are inappropriate

to interface with hardware, device drivers must be written in a language that can handle pointers, access registers, and other low-level, machine-specific interactions, such as C or C++. A PBM that interfaces to non-PB3A-based software is called a Port-Based Driver (PBD).

By coupling the PBM's modularity and specializations with distributed code, communication, and mobility services, PB3A allows for the creation of self-adaptive systems where complexity arises from interactions among simple software modules. A more detailed discussion of PB3A is contained in [3].

3 Real and Virtual Environment

RAVE is a mixed-reality simulation environment that allows the user to create robust control software for mobile robots efficiently. RAVE's capabilities allow robots to simulate the performance of their controller before deployment into the real robot. Robot controller programs are linked to libraries that provide a standard interface to real or simulated robots. These libraries form a layer of abstraction that separates the high-level control software from the low-level interaction with device drivers. This provides the facility for real robots to sense virtual obstacles, the placement of virtual sensors on real robots, and the interaction between real and simulated robots. This layer of abstraction also allows the direct transferring of programs from simulated to real robots. Transferring of programs between real and simulated robots is accomplished by dynamically loading the appropriate drivers at execution. This allows the high-level control software to reconfigure and adapt itself without disturbing the integrity of the low-level code.

Furthermore, real robots can be endowed with virtual sensors which are not augmentations of a real sensor counterpart. For instance, if the user of RAVE had acquired a mobile robot chassis but had not yet determined the sensor suite, the user could place any number of virtual sensors in appropriate (or completely inappropriate) locations and evaluate the performance of the robot. This process can be repeated until the user is satisfied with the locations and types of sensors to be used. This can save much time and energy as compared to experimentally determining the sensor suite in hardware. Also, RAVE offers users the flexibility to write a virtual sensor driver which has no real-world analogue, such as a sensor that detects the velocities of other robots, or an indoor GPS sensor. When virtual obstacles are injected into the world model of a real robot, several issues arise. First is the issue of sensor fusion. Detecting virtual obstacles requires RAVE to have an appropriate model of the real sensor being augmented. Once the virtual component of the sensor computes its reading, this result is then fused with the result from the real component

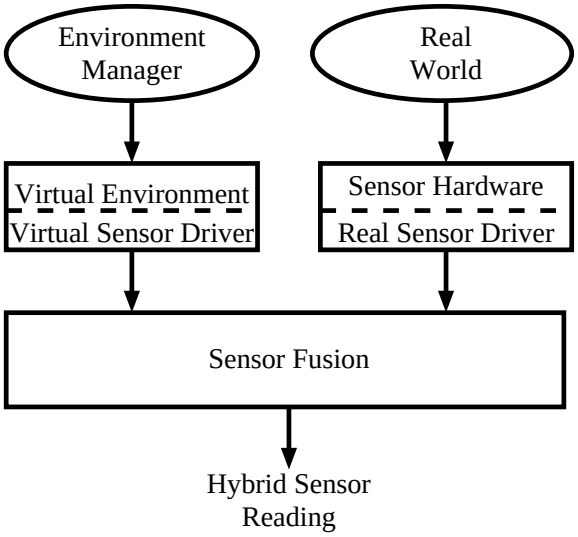

Figure 2: Diagram of a hybrid real and virtual sensor.

of the sensor and a single reading is returned (cf. Figure 2).

RAVE allows for interaction between real and simulated robots. This is useful to evaluate the performance of the controller software against a wider range of scenarios than would be possible with available systems, but gives more realistic results than simulation alone. This allows the user of RAVE to consider more avenues, with greater confidence, and create controller software that is better suited to adapt itself to changing conditions. A more detailed discussion of RAVE is given in [2].

4 Three Levels of Self Adaptation

As software systems grow in complexity, it becomes infeasible for humans to monitor, manage, and maintain every detail of their operation. From a human-computer interaction standpoint, it is desirable to build systems that can be tasked easily, perform intelligently (as evaluated from the perspective of the user), and complete the tasks with little or no human intervention. Recognizing this need, the ultimate goal of our system is to aid in developing systems that are self adaptive. These systems analyze their performance and can dynamically reconfigure themselves to fit better to the current operating conditions and goals in a distributed environment. From a software perspective, three natural forms of adaptation arise.

The first form of adaptation is parametric fine tuning. Most software is written in terms of algorithms that manipulate data. The behavior of these algorithms depends on their parameters. Much research has been done on estimating the error of an algorithm and using that metric to mod-

ify the parameters. For instance, this could be changing the synaptic weights in an artificial neural network through backpropagation or the coefficients of an adaptive digital filter.

The second form of adaptation is algorithmic change. There is seldom one way to solve a given problem; every different approach to solve a problem or calculate a quantity gives rise to a unique algorithm. Two algorithms designed to address the same problem may behave differently based on the precise circumstances under which they are used. A system that is aware of the current operating conditions and the limitations of the algorithms it employs could dynamically choose and switch algorithms when conditions change. For instance, as lighting conditions vary, swapping a stereo vision algorithm for an HSI-based vision algorithm may improve performance.

The third form of adaptation involves mobility. In a distributed environment, computing resource availability varies both temporally and by location. Certain nodes may have special-purpose hardware, more abundant memory and processing power, or lower data-access latency. Software that is aware of the resource conditions under which it operates could migrate to complete its tasks sooner or make progress in the case of failures.

Our system provides the primitives and methodologies by which all three forms of adaptation may be realized and initiated by the software itself when operating conditions warrant. These forms of adaptation are crucial to create robust control software because of the wide range of conditions under which mobile robots operate today. Control software in these robots must be able to analyze its performance and possibly invoke some form of adaptation to achieve its goals better.

5 Self-Adaptive Control Software

Creating self-adaptive control software for mobile robots requires the union of the distributed, mobile, multi-agent framework that is PB3A and the mixed-reality simulation environment that is RAVE. Where RAVE provides interfaces to both real and virtual robot entities, as well as both a real and virtual world, PB3A provides the structure and components for a self-adaptive robot controller.

When RAVE is used independently, each controller must be written as an individual C++ program that utilizes RAVE's interface to the actuators and sensors of a robot. When combined with PB3A, each robot runs a shell program that relays actuator commands and sensor readings through a TCP/IP socket to PBDs in a PB3A system. This allows the PBMs that comprise the control software to interface with the actuators and sensors in a port-based manner, thus preserving the modularity and self-adaptivity inherent in a PB3A system. This integration allows a PB3A-based application to simultaneously control one or more robots through a consistent actuator and sensor interface, regardless of whether the components are real or virtual. Due to the consistent information-sharing scheme of port-based systems, PB3A control software may execute directly on the robot's on-board computer (if available), on a proxy computer, or across several computers. Also, the control software may dynamically move its component PBMs to different machines or the PBDs may open a new socket and begin controlling a different robot. This facility, coupled with PB3A's dynamic loading aspects, greatly simplifies the task of upgrading robot control software, which can be upgraded on the fly with minimal downtime and disruption. Since PB3A was designed with self-adaptability in mind, the control software can completely change its structure of the executing robot program simply by remapping ports or instantiating new PBMs. Finally, PB3A's migration aspect allows simple context transfer between robots. All or part of a robot's executing control software may be transferred to another robot should the software decide such an adaptation is necessary.

Figure 3: Robots used for mapping: Patton and Rommel.

6 Applications

In this section, we present two scenarios. In the first scenario, the control software for the mobile robots is endowed with the ability to be mobile in order to avoid hardware failure. In the second scenario, we demonstrate self adaptation by task decomposition and on-line skill composition.

6.1 Fault-Tolerant Mapping

In this scenario, there are two mobile robots, shown in Figure 3, that are tasked to map an area using ultrasonic sonars. Furthermore, the robots must be able to overcome hardware failure to complete the task. The PB3A-based self-adaptive control software (shown in Figure 4) consists of a simple strategy to explore its environment, a Bayesian mapping algorithm, and a diagnostic hardware monitor. The diagnostic hardware monitor is responsible for detecting any hardware failures and proposing remedies for the failure. For this application, we used a naïve Bayes classifier (see [6]) that was trained while the robot is known to be operating properly. If the behavior of the robot changes dramatically while mapping, the classifier outputs a failure flag. This flag is passed on to another PBA that analyzes its available options. Presently, this agent uses a case-based reasoning system to determine the course of action to take. Once the option has been selected, the control software is adapted accordingly. The broad classes of possible adaptation are discussed in Section 4 and are provided by the PB3A system.

In a typical scenario, we have one robot that begins mapping a laboratory, while the other robot sits idle. If the first robot experiences a hardware failure (such as a graduate student cutting the power to its motors) then the hardware monitor issues a request to the system to move the controller software to the idle robot. The system then requests that all components of the controller serialize their state so that any information collected by the first robot can be sent to the second robot. The controller finally resumes execution on the healthy second robot and then continues mapping after the first robot fails. The transferring of the control software takes about five seconds once hardware failure is detected. Thus, control software written in this fashion can overcome crippling hardware failures and still complete its task.

6.2 Task Decomposition

In this scenario, we demonstrate self-adaptation by task decomposition and on-line skill composition. Task decomposition is a method used to aid the design of new software by decomposing a task based on its similarity to previously known tasks, or portions thereof, using a case-based reasoning engine [4]. Given a specification, the task is broken into a hierarchical set of subtasks, which are subsequently assigned to the available robots. These are then translated into known skills implemented with PBMacros that utilize the specific sensors and actuators available on that robot. Finite-state and parallel automata are used for temporally sequencing the skills. These automata are used to perform subtasks and, ultimately, the task itself, with the subtasks

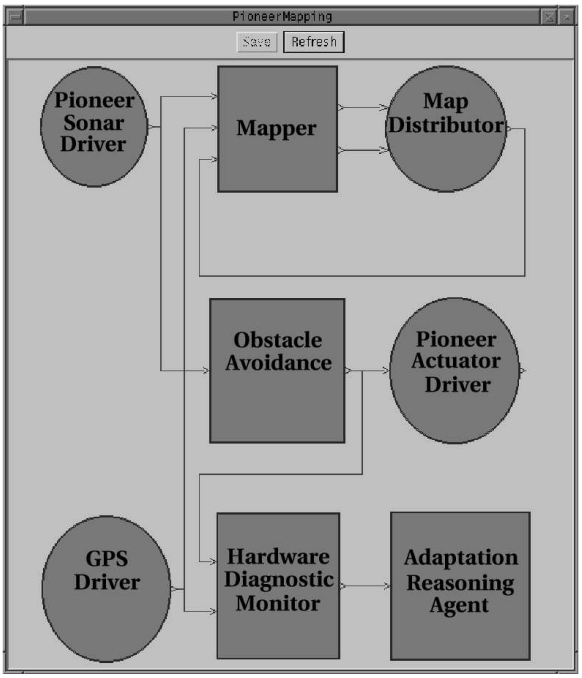

Figure 4: Layout of the PBMs used for mapping.

having been designated by the task-decomposition software. If a resource (e.g., the robot's motors) will be used simultaneously by multiple skills then the task decomposer attempts to use a skill-composition mechanism to produce a new skill that can perform all the functions of the conflicting skills.

The skill composition operates by using the pre-existing skills as guides for exploration in a reinforcement-learning paradigm (for an overview of reinforcement learning refer to [5]). Currently, the reward function for the reinforcement learning is provided by the user, but a reward function may be created from a specification stored in the decomposition database. RAVE allows the robots to simulate the new control software derived from the skill-composition learning. When a PBA deemed the performance satisfactory, the old control software is swapped out, replaced by the new control software that could successfully complete the task.

In addition to providing a mechanism for the initial design of the software, the task decomposition system can also be used to facilitate the dynamic reconfiguration of the PBAs by providing available alternatives. Within each PBMacro controlling a subtask is a pair of PBAs responsible for coordinating the reconfiguration of that particular subtask: a 'trigger' agent and a 'reconfiguration' agent. The trigger agent is responsible for deciding when the reconfiguration needs to occur; the specifics of this agent are left open-ended, but an example is the diagnostic monitor dis-

Figure 5: ActivMedia Pioneer II robots used in task decomposition: Asbestos, AgentOrange, and Saccharin.

cussed in the previous section. The reconfiguration agent is responsible for communicating with the task decomposition system to determine what changes need to be made and actually performing those changes to the software using PB3A's built-in capability for moving and swapping agents.

To demonstrate these capabilities, the robots are tasked to locate lost compatriot robots in an unknown environment. Each robot (shown in Figure 5) is unaware of its own location and how many other robots are operating in the environment. Each robot has a color camera and a ring of sixteen ultrasonic sonars. Each robot starts in a random location in the environment and wanders the area, avoiding whatever obstacles, whether dynamic or static, it may encounter along the way. Using a simple HSI matching algorithm, it searches for other robots with the camera. As the robots search the area, they form together into groups with the other robots they encounter and continue searching. Within the formation, the front robot continues to search for other robots while the other robots follow behind it.

7 Related Work

Mobile robots initially began as monolithic and fragile systems. The first major development was Shakey from SRI [7]. While successful in various experiments, Shakey operated in a sterile and unrealistic environment and its planning software was susceptible to severe failures when it encountered small disturbances in its environment. Brooks later created several robots that were designed to operating in cluttered, unfriendly environments using the "subsumption architecture" [1]. This architecture allows lower, more critical levels of operation to subsume control from higher levels. Though these special-purpose robots interacted with this relatively hostile environment brilliantly, their ability to reason about the world was intentionally neglected thus limiting the robots to reactive control schemes. Furthermore, since the controllers for many of these robots were designed in hardware, self-adaptation was not feasible.

The port-based concept is derived primarily from port-based objects, first proposed and implemented by Stewart and Khosla in Chimera [8]. Port-based objects were designed for real-time control applications in a multiprocessor environment with a single high-speed backplane. The informational scope within which the port-based objects exist is a flat, public data structure visible to all objects. This implementation is very efficient for monolithic systems, but it provides no concept of agency [9].

PB3A should be viewed as the natural evolution of the port-based concept. Where port-based objects were designed for multi-processor environments and for direct human-initiated reconfiguration, PB3A is being designed to utilize loosely coupled distributed computing infrastructures and self-initiated software adaptivity. The modern-day computing paradigm exemplified by distributed and self-adaptive systems absolutely requires the autonomy and self-awareness that are the hallmarks of agent technologies. Software composed from independent, self-aware agents that are able to alter their own structure, are best suited to complete tasks in the case of network latencies, node failures, and general operating condition variations that characterize real-world environments. PB3A's first advantage over Chimera is that PB3A uses dynamically loaded Java byte-code to avoid recompiling and relinking the entire system when new objects are added. Specifically, to support distributed computing, PB3A augments the notion of the port to include cross-network links, employs an encapsulated memory model to make each PBM self-contained, and utilizes mobile Java byte-code along with the previously mentioned dynamic loading to provide code on demand to individual nodes of the network.

8 Conclusions and Future Work

We have described two systems that facilitate the development of self-adaptive control software for mobile robots. The first system is a mobile, agent-based framework, the Port-Based Adaptable Agent Architecture (PB3A). The second is a mixed-reality simulation environment, the Real And Virtual Environment (RAVE). Together these systems allow the user to create self-adaptive control software at all three levels: parametric fine tuning, algorithmic swapping, and software mobility. When control software is written for mobile robots incorporating this self-adaptivity, more robust systems emerge. These robots have controllers that can adapt to overcome adverse and rapidly changing con-

ditions.

Future work will allow the user to have a more interactive role in the execution of a self-adaptive controller. Presently, once the controller begins execution, the software itself determines when or if the system should adapt (moving agents to different nodes, swapping algorithmic modules, etc.). We will allow the user to adapt the system during execution by specifying any aspect of the controller possible. Also, we plan to extend RAVE from two to three dimensions. Clearly mobile robots now operate in environments that cannot be characterized by planar geometry, which limits the usefulness of these simulations.

Acknowledgments

We would like to thank Enrique Ferreira and Rich Malak for their help and insight into this work.

This work was supported in part by DARPA/ETO under contracts F30602-96-2-0240 and DABT63-97-1-0003 and by the Institute for Complex Engineered Systems at Carnegie Mellon University. We also thank the Intel Corporation for providing the computing hardware.

References

[1] R. A. Brooks. A robust layered control system for a mobile robot. Technical Report Memo 864, Massachusetts Institute of Technology Artificial Intelligence Laboratory, 1985.

[2] K. R. Dixon, J. M. Dolan, W. Huang, C. J. Paredis, and P. K. Khosla. RAVE: A real and virtual environment for multiple mobile robot systems. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1999.

[3] K. R. Dixon, T. Q. Pham, and P. K. Khosla. Port-based adaptable agent architecture. In *International Workshop on Self-Adaptive Software*, 2000.

[4] J. R. Jackson. Using automatic task decomposition to design port-based adaptable agent software. Master's thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 2000.

[5] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[6] T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[7] N. J. Nilsson. Shakey the robot. Technical Report 323, Stanford Research Institute Artificial Intelligence Center, 1984.

[8] D. B. Stewart and P. K. Khosla. The chimera methodology: Designing dynamically reconfigurable and reusable real-time software using port-based objects. *International Journal of Software Engineering and Knowledge Engineering*, 6(2):249–277, 1996.

[9] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995.