Autonomic Systems for Mobile Robots

Nik A. Melchior William D. Smart
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO 63130
United States
{nam1,wds}@cse.wustl.edu

Abstract

Mobile robots are an excellent testbed for autonomic computing research. The ultimate goal of robotics research is to develop a platform that can function autonomously in the face of hardware and software failures. This goal is becoming more important as robots are increasingly being deployed outside of controlled environments. In this paper, we discuss our work toward implementing an autonomic system for a mobile robot. This work is motivated by our experiences with existing mobile robot control software during real-world deployments.

1. Introduction

Mobile robots are an excellent testbed for autonomic computing research. The ultimate goal of robotics research is to develop a platform that can function autonomously in the face of hardware and software failures. This goal is becoming more important as robots are increasingly being deployed outside of controlled environments.

Many of the common failure modes can be successfully addressed using autonomic computing techniques. In particular, robots suffer from a higher rate of hardware failure than other computer systems. This is mostly due to the fact that robots spend their lives moving about the world, often colliding with things. After a while, connectors are shaken out, memory modules work loose, and disk drive heads crash. A typical mobile robot contains several computers on a local area network, a range of sensor hardware, and a locomotion system. This results in a lot of hardware that can fail. Dealing gracefully with such failures will be a key challenge for any successful autonomic computing system. It seems clear that a recovery-oriented approach [4] to robot operations is essential if we are to field robots for extended periods in the real world.

Deep space probes and planetary rovers are perhaps the most obvious beneficiaries of effective self-managing systems. Since they are designed to operate far from Earth, human intervention is often not an option. Systems such as Remote Agent, carried about the Deep Space One probe [2, 5], the Mars Pathfinder mission, and the two rovers currently on Mars have conclusively demonstrated the utility of such systems.

We are interested in exploring the use of autonomic computing techniques in the domain of ground-based mobile robots. Much of the work in this area has traditionally centered on efficiency, portability, and ease-of-use issues [3]. However, based on our deployment experiences, this is not enough, and we must also address issues of robustness and fault-tolerance.

2. Autonomic Systems for Mobile Robots

Our experiences with deploying mobile robots in the real environments have highlighted a number of problems that seem to recur over and over again. (1) All hardware will fail eventually. (2) Making control decisions based on bad sensor data is very dangerous. (3) Software written on-site is often very buggy. In this section, we discuss how we are addressing these issues as we develop an autonomic system for our mobile robot.

2.1. Sensors and Service Interfaces

Robots decide what to do based on the information from their sensors. We need to recognize when a sensor fails (or loses its calibration), and begins to return incorrect information. Such failures can be in the hardware of the sensor, or in the sensor driver code or firmware. If the sensor hardware fails, then there is no choice but to abandon the sensor, and try to get the same information elsewhere. If the fault is in software or firmware a restart might solve the problem.

Proceedings of the International Conference on Autonomic Computing (ICAC'04)
0-7695-2114-2/04 $20.00 © 2004 IEEE

Robot control programs are typically written in terms of specific sensors and actuators. For example, an obstacle-detection program will typically read values from the laser range-finder, and use these values to calculate the position of objects. However, if the laser range-finder fails, the obstacle-detector also fails.

A potentially better way to structure such software is using a service-based framework. Instead of asking for a specific sensor, a program asks for information conforming to a specific interface. In this case, the obstacle-detector could ask for "distance" in various directions. This request is assigned to a particular service, which then supplies the information. Typically, distance data would be supplied by the laser range-finder. However, if this device fails, another service could take over the role of supplying distance information. This provides a more graceful performance degradation as sensors fail.

Although some programming frameworks currently offer abstractions such as "distance" [1], it is still an open question as to how effective such an interface is in practice. Although many sensors are capable of returning distance measurements, they all have different characteristics and failure modes. These are typically accounted by the control program. If we no longer have knowledge of the actual sensor we are using, how can we account for it's foibles? Is it always safe to substitute one sensor for another, without letting the control program know?

Sometimes we *do* care about the low-level details. In this case, we're swapping fine control over the sensors and actuators for less autonomic failure recovery. The key challenge will be to devise a control framework that allows both modes of operation to co-exist with each other.

By making each available service small, and self-contained, we can localize the effects of many software and hardware failures. Simple processes can be restarted more effectively, and are generally easier to test and debug. This does, however, result in a complex assignment and optimization problems to provide a given set of services.

2.2. Power Awareness

Although saving computation is important in any environment, it is especially important on a mobile robot. Robots are powered by batteries, and more computation drains the batteries faster. We need to be aware of the amount of computation that the robot is performing.

In addition to standard techniques, such as processor speed control, specific services can be chosen at runtime based on a consideration of the available power. For example, when power is low, faster, more approximate sensor processing algorithms can be used to conserve power. This introduces some interesting optimization problems, if more than one service can supply the needed sensor information.

2.3. Getting Help

We would like the robot to be completely autonomous, never needing human intervention. This is, however, not practical since there are failures that we simply cannot deal with without help. The autonomic system should be capable of asking for human intervention, and also of summarizing the state of the system in a meaningful way. We can't assume that the human supervisor will be paying attention all of the time, so any request for help will have to come with some context. This context could take many forms, from a simple history of device failures to the output of an automated reasoning system.

3. Current Work

We are currently implementing and testing the ideas discussed above. Our initial experiments are encouraging, and we expect to be able to present quantitative results at the conference. Based on our experiences, the key aspects of an autonomic system for a mobile robot are:

Strong Modularity: By splitting control into small subtasks, we can localize the effects of many problems.

Early Fault Detection: Even if we can't solve a problem, we can stop it before it gets worse. Poorly calibrated sensors can be disabled, for example, preventing collisions with obstacles in the environment.

Include Humans: Humans, if available, can often diagnose and perhaps correct problems more effectively than an automatic system. They should be integrated into the error recovery strategy.

References

[1] G. K. Kraetzschmar, H. Utz, S. Sablatnög, S. Enderle, and G. Palm. Miro – Middleware for Cooperative Robotics. In *Robocup 2001: Robot Soccer World Cup V*, pages 411–416, 2002.

[2] N. Muscettola, P. Nayak, B. Pell, and B. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1–2):5–47, August 1998.

[3] A. Orebäck and H. I. Christensen. Evaluation of architectures for mobile robotics. *Autonomous Robots*, 14:33–49, 2003.

[4] D. Patterson and *et al.* Recovery oriented computing (ROC): Motivation, definition, techniques, and case studies. Technical report.

[5] B. C. Williams and P. P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 971–978. AAAI Press / The MIT Press, 1996.