Autonomy in Spacecraft Software Architecture

Henry Hexmoor

University of North Dakota
Department of Computer Science
Grand Forks, ND 58202

Abstract

In this paper we discuss the concept of autonomy and its impact on building complex systems for space applications that require autonomy. We have developed a few metrics for quantification of autonomy in systems.

1. Introduction

As more complex systems are being developed, there is greater need for quantifying their level of autonomy (Brown, Santos, Banks, & Oxley, 1998; Gat, Pollack, & Cohen, 1998; Hanks, Pollack, & Cohen, 1993; Hexmoor, Kortenkamp, & Horswill, 1997). This is more evident in systems that control spacecraft.

Let's consider a device as a complex machine that appears capable of tasks commonly performed by intelligent organisms. An automobile equipped with cruise control, road-sensitive traction, and self-inflating tires is such a device. Such devices receive input from the environment and follow an algorithm provided by the device designer to produce an output. In general, devices cannot tell how well they perform. Furthermore, devices may only manipulate a fixed ontology of their surroundings -- they represent things in the world in a fixed way. Many robotic applications would qualify them as devices. To the extent a system can be aware of its performance and can improve its pre-programmed ability to interact with its surroundings, or can alter its ontology, it is autonomous. Autonomy is self-governance over its output. A system such as smart chess-playing programs can be intelligent without being autonomous. This notion of autonomy is desirable in systems such as autonomous space applications needed in long-duration space missions.

Elsewhere (Hexmoor, Lafary, & Trosen, 1999), we have argued that autonomy levels closely correspond to an agent's rank. We defined five ranks: fully autonomous, boss, cooperative, underling, instructible, and remote control. We argued that an autonomous agent's decision-making changes when it introspects about its rank with respect to other agents. In spacecraft software where a human user is in ultimate control of the agent, the agent is not required to introspect about its level of autonomy. The human user changes the agent's autonomy level and gives it new instructions.

Therefore, these systems also need to be interruptible. Although, these systems don't need to introspect about changing their autonomy when interaction with human users, they need to be aware of their resource usage. Finally, autonomous agents need to learn and self-detect faults.

2. Resource management

Resources are either shared or used up. Shared resources are called reusable and we will first focus on this type of resource. We are concerned with design of behaviors that account for using a shared resource. Specifically, we will examine design of two behaviors that share a single resource. Let’s consider a behavior F for navigating in the direction the robot is facing while following a moving target. A second behavior C uses the vision system to look for objects that are closer and smaller for obstacle avoidance. Let’s assume that the vision system can only process the target at a minimum distance of 5 feet, and a minimum height of 4 feet. The vision system cannot see objects that are shorter than 3 feet and closer than at 3 feet away from the robot. When the vision system is maintaining the target in view, the robot cannot attend to nearby objects. Behaviors F and C must share the vision system by time sharing it. Let the minimum frequency for maintaining the target in view be denoted by ft and the minimum frequency for avoiding nearby objects by fo. Assume the vision system has a frequency capacity of C. Assume the switching cost between two visual tasks is negligible. If C is less than ft+fo we need a more powerful vision system. Otherwise, we can measure the ability of the system for sharing the vision system. The best scenario is that fo and ft are met. If the actual frequencies are marked as fo' and ft', we can measure the shortcoming as |fo-fo'| + |ft-ft'|. We suggest (|fo-fo'| + |ft-ft'|)/C as a metric for measuring adequacy of sharing the vision resource. Let's call the general form of this metric as a *resource-sharing metric*.

This metric can be generalized to account for multiple behaviors requiring the use of a single resource. This metric can be useful for autonomy software by monitoring the level of resource management. Information can be maintained about the pattern of resource usage by behaviors to allow more equitable usage. For critical behaviors, a priority system among behaviors can be used to decide resource usage. We can easily extend the metric to be weighted with priorities.

Next, we will illustrate that deliberately accounting for consumable resources pays off. Consider a grid of 3 by 3 marked 1-9 in row major format. Assume 9 different objects that appear randomly. Each object will be placed in a single grid location designated for that object. Consider a robot arm that can place objects in their respective locations. We assume the cost of sensing is negligible. The arm can carry 1-3 objects at a time. Let's assume the cost of carrying one object is one unit and the cost of carrying 2 objects is 1+U and the cost of carrying 3 objects is 1+2U. Assume U < 1. The arm will carry multiple objects only if they are in neighboring grid locations. The arm can wait until 3 objects show up before taking action. This manner of using the arm is treating the arm as a resource that can be used more efficiently. It cannot wait until all objects show up and then pick neighbor objects. In the most reactive scenario, an object shows up and the arm places it in its location. Lets consider 9 units of work associated with the reactive approach. If with luck, neighbors of 2 always show up, the work is 5+4U. If with luck, neighbors of 3 always show up, the work is 3+ 6U. If U = _, then the work is 7 and 6 units respectively. This is a considerable saving over a purely reactive scenario. The savings are 22% and 33%. Let's assume we can measure the cost of plans with respect to resource usage. The cost of the plan with the most frugal resource usage is F. The cost of the plan with the most liberal resource usage is L. We define a *resource-usage metric* as (L-F)/F. Using this metric, we can determine if a consumable resource is being used optimally.

3. Fault-detection

Should the organism be cognizant of its own ineffectiveness or abnormalities in its environment? If things are very wrong, very little might be possible. The system might be able to execute a nesting maneuver until the situation improves or it can think of a solution. The latter would be a cognitive act and would require reasoning, and perhaps model-based reasoning is needed as in NMRA (Pell, Dorais, Plaunt, & Washington, 1998).

In simple situations where a parameter is being controlled, the rate of divergence in feedback loops can be used for fault-detection. The ratio of the time difference between existence of a fault and the time it is detected over time span of operation can be considered as a *fault-detection metric*.

4. Interruptibility

An autonomous system might share its behavior-generation with another agent. It is human nature to think that we need the ability to intervene in critical tasks. In the labs we take pleasure in designing creatures that are autonomous, but in arenas where it is very critical, the ability to access and selectively change system behavior is highly desirable. As in one of Azimov's laws of robotics, when it comes to interaction between a human user and agents, the human plays the role of Boss or "God" and the agent obeys. Instead of adjusting its level of autonomy it needs to be interruptible.

Interruptibility requires architectural features for sharing decision making about controlled, automatic, and reflexive actions. In its most general form, it might be desirable to modify knowledge, memory, or operation of decision making processes. However, we limit our interest in interruptibility firstly to an automatic system's ability for *suppression* and *superseding* what the system decides at certain levels of granularity. Suppression is cessation of an ongoing behavior. Superseding is replacement of the execution of one behavior over another. This type of interruption is asynchronous and is performed in a supervisory manner. The system has no expectations. Such a system will be obedient of its masters and we will call it auto-obide or autobide for short. An autobide can be measured as to how fast it responds to an authorized interruption. Let's assume an active behavior to be interrupted is a feedback loop generating impulse inputs with frequency of f; i.e., every f units of time, the feedback loop issues an impulse input. We can measure suppression rate in how many f units it takes before the feedback loop stops. We keep in mind that some feedback loops cannot end immediately and must end gradually. Superseding can be measured in terms of f units as well. Bringing up a new behavior might have to be gradual and it will affect the behavior replacement time. Let's denote the cost of the system's lethargy to respond to interrupts as L and the overhead cost of quickly responding to the overhead as O. L can be decomposed to cost per f. Quick response can be implemented by building a mechanism that is either specialized to expected input or that short cuts to extraneous considerations. At any rate, we think of this overhead as an architectural feature with nearly constant cost. With N as the nominal cost of system operation, (L-O)/N is the *interruption metric*.

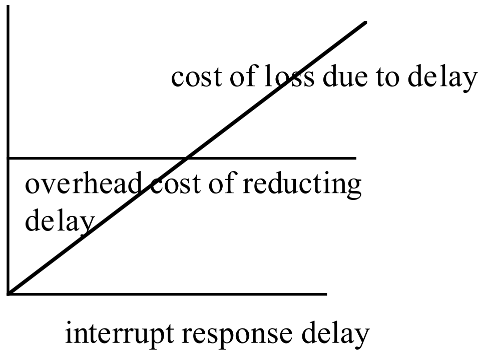

Figure 1

The general principle for this metric is shown in Figure 1. Reducing delay often incurs constant cost, and cost due to delays is linearly proportional to interrupt response delay.

For an example of the interruption metric, consider a conveyor belt delivering widgets at a constant speed and a robot arm pushes the widgets off the conveyor belt onto a bucket. The bucket location is mostly constant until it is full and a person replaces the bucket. The new location of the bucket may be different than the previous location. We expect the robot be able to react to new bucket location quickly. We can think of bucket replacement as an interruption. If the robot is not quick, widgets will be pushed onto the floor incurring a cleanup cost. However, the robot's more frequent monitoring of bucket location also adds to the cost. The interruption metric can be used to decide the utility of the more interruptible mode.

5. Learning

In this section, we consider learning as a cognitive function for modifying knowledge and performance. An important element of learning is being able to ascertain the current state of knowledge or performance. Metrics we have been discussing are useful in this determination. For a system that learns, there are additional metrics to judge the process of learning. The learning curve is a plot of system performance or knowledge attainment over time, and it is used to judge the merit of the learning algorithm.

5.1. Knowledge migration

Let's envision the ability to transfer knowledge from one layer of software to another. The layers might be high level control and low level control. Let the cost of a system operating with knowledge prior to the transfer be C1, and after the transfer to be C2, and the cost of the transfer to be O. (O+C2 -C1)/C1 would be a metric for transfer. For example, in a grid of 5 by 5, we can assume the system will plan steps for travel between the start and the goal positions in terms of grid step movements. Let p1 be the cost of coming up with a plan to travel between the start and the goal. Let P2 be the cost of using a reactive set of rules. P2 is the cost of matching and retrieving rules, and planning is not required, but the cost of caching the plan into rules is O, then (P1+O-P2)/P1 is the metric.

6. Conclusion

We presented a few metrics for quantification of autonomy in complex systems. These metrics can be used in measuring a degree of autonomy in complex systems such as systems for the spacecraft. Such metrics will aid in designing software that is robust and can be used in autonomous control of spacecraft.

References

Brown, S., Santos, E., Banks, S., & Oxley, M. (1998). Using explicit requirements for interface agent user model correction. International Conference on Autonomous Agents (Agents '98). Minneapolis, MN.

Gat, E. (1998). Three-layer architectures. In Kortankamp, D., Bonasso, P., & Murphy, R. (Eds.). Artificial intelligence and mobile robots. Boston: MIT Press.

Hanks, S., Pollack, M., & Cohen, P. (1993). Benchmarks, testbeds, controlled experimentation, and the design of agent architectures. AI Magazine 14(4): 17-42, MIT press.

Hexmoor, H., Kortenkamp, D., & Horswill, I. (Eds.). (1997). Software architectures for hardware agents. (Special issue). Journal of Experimental and Theoretical AI, 9, Taylor and Francis.

Hexmoor, H., Lafary, M., & Trosen, M. (1999). Adjusting autonomy by introspection. AAAI Spring Symposium, Stanford, CA.

Pell, B., Dorais, G.A., Plaunt, C., & Washington, R. (1998). The remote agent executive: Capabilities to support integrated robotic agents. Working notes of the AAAI Spring Symposium on Integrated Robotic Architectures. Stanford, CA.