Autonomy Control Software

Henry Hexmoor[1] and Salvatore Desiano[2]
[1]Department of Computer Science, University of North Dakota Grand Forks, ND 58202, USA. Email: hexmoor@cs.und.edu
[2]Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA. Email: sal@ri.cmu.edu

1. Introduction

For a number of years, researchers in AI and robotics have been sharing organization principles for software development called "architectures", (Arkin, 1998; Hexmoor, Kortenkamp, Horswill, 1997; Kortenkamp, Bonasso, Murphy, 1998). Most recently, there has been an interest in extending the software architectures originally designed for robotic applications to accommodate autonomous operation beyond robotics (Pell, Dorais, Plaunt, Washington, 1998). These architectures attempt to state principles for organizing software for intelligent systems that exhibit cognitive functionalities such as natural language understanding, planning, and learning as well as organism-like functionalities to cope in the world such as reflexive and reactive behaviors.

These architectures, and the systems on which they are implemented are used to control complex systems or environments and often require unattended or minimally supervised operation. In order to control such systems, the architectures often have a hybrid or multi-tiered structure. Partly, this is due to the many academic and engineering disciplines from which the techniques are borrowed and from which the programs originated. Additionally, multi-tiered architectures allow for a natural abstraction and separation of system capabilities by their sophistication, much like organic systems.

The complexity of the systems and environments make it difficult to provide guarantees on the system performance. It is precisely this complexity, however, that makes it essential to develop evaluation techniques at the time of design and as proofs of system capability for deployment. These techniques go beyond software engineering metrics and evaluate the system on a higher, behavior level. We, as system builders, seek evaluation of agent architectures with respect to objective system qualities, that are commonly agreed and expected criteria. By example, a few of these qualities are: Autonomy, Timeliness, Purposefulness, Robustness and Failure Tolerance, Coherence, Flexibility and Adaptability, and Reliability, Interruption, and Learning and Adaptation.

(Hanks, et al 1993) discuss the merits of test beds and benchmarks for revealing values of agent design concepts. These issues mostly revolved around reactiveness versus planned behavior. They argued about inconclusiveness of experiments but that generalizations and interpretations are left to researchers. Our approach offers generalized metrics and uses benchmarks for discussion and instantiation.

(Mali and Mukerjee 1998) define behaviors and tasks, and present concepts for comparing behaviors along dimensions of power, span, task space, usefulness, flexibility, and scalability. Their comparative metrics become guides for knowledge engineering behaviors at appropriate levels along their dimension of analysis. (Mali 1998) goes further and defines goals, environments, and markers. Mali sketches that more complex goals are achieved by adding markers or making behavior sets complex by adding coupling among behaviors. He argues that reactivity and planning do not have to be considered orthogonal but are related along how well the agent treats complexity of the world. With that he implies that there is a spectrum of ways to interact with the world. A knowledge engineer can change the behavior sets or markers to gain more or less reactivity. We agree with Mali's points. However, this does not invalidate multi-layered architectures since they provide concurrent means of interaction with the world. To use Mali's terms, his analysis provides

a framework for designing behavior systems that address reactivity along spatial dimension. Hybrid systems are addressing spatial as well as temporal dimensions of interacting with the world.

MissionLab (MacKenzie and Arkin 1998) is a platform for editing where behaviors and the mission for multi-agent system are specified apart from agent architecture. Flexibly, users select architectures to instantiate the mission. This is a highly desirable feature that relieves the burden of coding missions in agent systems from the start. Although they have not done this, we believe metrics can be incorporated into architecture instantiations for measuring the effect of architectural principles.

(Brown, et al 1998) preset metrics for interface agents. Their metrics in the areas of adaptivity, autonomy, collaboration, and robustness are ratios measuring system's actual level of appropriate output over maximum expected appropriate output. These measures consider the overall system but do not target effect of specific system design/architecture. In contrast to objective metrics defined here, ours are relative.

(Gat, 1998) presents a history of a three-tiered architectures. Gat argues that separation of control software into layers has mostly to do with the need to maintain internal state. Internal state refers to recording what is true in the world. The three layers are identified by characteristics. First, little or no state is maintained and world interactions are as responsive as they can be without imposing hard-real-time requirements. Furthermore, all failures are detected. Second, states are based on past and world interactions governed by this layers are somewhat slow. Third, states are based on future and world interactions are slow.

Our workshop focused on definition and evaluation of architectural qualities, (Hexmoor, 1999). The goal was to bring together cutting edge researchers in an informal setting to develop *specific* metrics and methods for the objective comparison of system architectures. To this end, we had three categories of papers. The first group discussed the general requirements for architectural metrics. The second group presented specific metrics that can be applied to whole architectures or to specific subsystems of those architectures. The third group discussed specific systems and the metrics that are important to those systems. Many authors brought empirical experience to the table, while others came from a more theoretical background. This workshop combined the theoretical ideals with practical requirements to arrive at useful results and inspirations for future work. In the following section, we put the workshop papers into an organized structure and discuss their primary arguments and contributions.

2. Workshop papers

The difficulty of defining metrics arises from the complex nature of the agents that we are programming. In a simple agent, such as one that monitors a system for erroneous behavior, there is an obvious metric: the percentage of erroneous behavior that the agent notices. More commonly, the systems are so complex that the correct metric is not always clear, or even if it is, it is not always feasible. This problem raises several questions: What, then, makes a good metric? What are examples of that metrics? How are these metrics used in practice?

2.1 Metric ideals

Lynne Parker took on the difficult task of defining several categories of metrics: robustness, fault tolerance, reliability, flexibility, adaptivity, and coherence. While she discusses them in the context of multi-agent systems, the definitions provide a framework and a mind set in which to adapt them to other types of systems. Each metric is defined explicitly in terms of measurable quantities, a quality that is essential to a usable metric. Parker discussed the notion of limited awareness of an agent with respect to other agents and how that limitation can affect metrics that measure time and energy. Furthermore, she followed through to show how each metric is achieved by the ALLIANCE system, and discussed how they can be applied to other systems.

Several papers, rather than arguing for specific types of metrics, suggested ways in which simpler metrics can be combined with more complex systems to provide equally strong guarantees.

Chris Landauer and Kirstie Bellman raise the interesting comparison of complex agents and biological systems. In both, they argue that the ability to use adaptive behavior is essential. They also argue that ensuring correct performance of such systems not only requires objective guarantees, but also requires a significant amount of infrastructure to monitor and modify the system's ongoing performance. Using a runtime supervisory infrastructure allows us to specify metrics that would otherwise impossible to guarantee. Rather than guarantee that a system will behave in a certain way, if at any point during operation, it begins to perform outside of acceptable operating parameters we can simply stop or alter the system's behavior. They proposed viewing many qualities as a resource. In fact, they suggest a general understanding of a *resource* as a combination of a *context* and a given *problem*.

Paolo Petta, et al., also invoke a biological comparison, but raise the issue of the model quality decay over long runs of an autonomous agent. If systems are to run for long periods of time, the model that we provide to the system may, over time, become grossly invalid. Therefore, if we build or system so that it can acquire a model by itself, the system is capable of error detection and cognizant failure. Like the suggestion of systems with more complex infrastructure, above, a system capable of cognizant failure can provide stronger guarantees by monitoring itself at runtime. Their approach considers agent *emotions* as a complex meta-management system.

Bringing these concerns together, Paul Scerri and Nancy Reed provide the selection of an agent architecture for several simulation environments as an example of how metrics are used in practice. In several tables, the qualities of different general agent architectures are compactly described. Combined with an explicit description of each environment, it becomes simple to select the appropriate architecture for each case. As metrics become more and more common, we can shift away from the traditional dogmatic method of choosing an architecture and shift toward precisely this kind of objective selection.

2.2 Specific Metrics

One of the benefits of working in a domain as varied as autonomous agents is the virtually everyone is working in a distinct environment with different concerns and difficulties. Though this requires us to work harder to interact, the resulting common ground is rich with general lessons that can be applied to many different areas of agent research. In this section we discuss specific metrics that authors designed for their own particular domains, but it becomes rapidly obvious that they are applicable in a wide variety of domains.

Eric Sanchis provides an excellent list of specific metrics that can be used to describe agents. He further divides these metrics into two categories: qualities and mechanisms. Qualities are more abstract qualitative measures like autonomy, proactiveness, and intelligence. On the other hand, mechanisms are abilities like perception, continuous execution, and mobility. Though these are only defined qualitatively, they provide us with ways to describe our agents. If these qualities and abilities could be described quantitatively, they would allow us to objectively compare systems and choose the appropriate agent for a particular problem.

Suzanne Barber and Cheryl Martin provide a fascinating discussion of the degree of autonomy as the level of input each agent makes in decision making. In contrast to the measure of autonomy, the level of autonomy is externally assigned. In a given level of autonomy, degree of autonomy varies. In a highly structured environment, for the supervised agents, the degree of autonomy of each agents is fairly small whereas for the supervisors, the degree of autonomy is fairly large. Barber and Martin give us a clear framework and a good feature (degree of decision-making) for measuring autonomy.

Robin Barker and Anthony Meehan discuss the importance of negotiation, and the autonomy of the negotiating agents, in the design process. Rather than using metrics to describe system performance, however, they suggest that the system can use metrics internally for controlling autonomous negotiation.

Denny Rock extends the idea of measuring an agent's autonomy to consider the agent's interaction with humans. Working in the context of helicopter collision avoidance, he describes the qualities required of a system that is capable of working in that setting. Introducing humans into the loop create a whole new range of issues, from the system's ability to work with the human to its' ability to gradually assume control or relinquish control. While the presence of humans makes these metrics are harder to define, most agent systems involve humans to some degree which makes these issues unavoidable.

More generally, many systems have multiple agents in which none of them are humans. Ralph Deters provides a very practical discussion of dependability in multi-agent systems. In order for such a system to behave correctly and efficiently, it must keep track of several characteristics of the individual agents: availability, reliability, security, and safety. To use this information in a useful manner, however, we must have certain guarantees on how those characteristics will change over time. These guarantees are phrased in terms of design goals or agent metrics: transparency, self-awareness, agent dependability, fault tolerance, obedience, and capability for local treatment. These very practical metrics, all of which can be considered at design time, better allow us to design systems that operate as reliably as we would like.

James Gunderson and Worthy Martin tackle uncertainty, an unavoidable aspect of virtually all agent environments. They divide issues of uncertainty into sensor uncertainty, uncertainty due to exogenous events, and operator uncertainty. Each of these is modeled as a probability between 0.0 and 1.0. Further, they explicitly state that the level of autonomy that an agent can support is determined by the agent's inability to meet goals without using external resources. From this set of definitions, they used simulation data to show that each has a direct, independent relation to the agent's ability to successfully execute a plan. From this work, we can see that metrics benefit extensively from extensive, well-defined testing.

Lynne Parker discusses several desirable criteria in multi-agent environments. It is argued that robustness and fault tolerance is increased by decentralization of action arbitration within an agent and among agents. Reliability is taken to be the system's ability to diagnose failures in task achievement and rapid task reallocation. This is accomplished by including mechanism that uses a parameter for tracking the agent's commitment for tasks. This issue seems related to agent timeliness but it is not discussed. A reasonably fast perceiving agent will notice and respond to tasks yet to be accomplished. She presents the companion notions of flexibility and adaptivity, which amounts to agent's awareness and responsiveness to changing performance levels in agents and environmental variations. Issues about automatically learning to note and respond to changes arise. Coherence is the last criteria discussed as the issue of agents working on tasks that is mutually beneficial and do not detract from their shared goals. One-way broadcast is offered as a first solution for agents for agents to gain information about one another's task. This seems to a good starting point although in some cases the cost of this communication and processing might make it difficult for agents to do their own tasks well.

Chris Landauer and Kirstie Bellman also define robustness, but add the measure of timeliness, or the qualitative question of whether the agent performs its actions soon enough. This has a very close relation to the work done in real-time systems, which takes care to define hard and soft deadlines, and in which timeliness is strictly defined as meeting a set of those deadlines.

To this end, David Musliner has formulated several guiding questions we can use to consider whether a system can work in real time. We must consider whether the agent's domain even allows the system to work in real time; is there always a solution? Or must we explicitly state restrictions under which guarantees of

safety can be made? Is the planner complete and guaranteed to have a bounded run time? Does the executive system operate fast enough to carry out those plans? Can the operating system and hardware support all of the software systems, or are they not fast or reliable enough? By considering these questions, and by coming up with explicit metrics that provide answers to these questions, we can more easily discuss whether systems are fit to run in real-time situations.

2.3 Implemented systems

To complete the picture, four of the papers describe actual systems and the metrics by which they were designed. These papers highlight a fact that is often overlooked in discussions about metrics: designers have always used them, though they have been called by other names such as "design requirements" and "design rules". Complex systems are described by design requirements and their success or failure is described by how well they adhere to these requirements. What is left for us to do is to apply the principle of design and performance metrics to the more abstract area of autonomous control software.

Ella Atkins, Ed Durfee, and Kang Shin present an architecture called CIRCA-II. They demonstrate that CIRCA-II meets the real-time requirements of unmanned aerial flight. They offer a mapping of domains such as automobile and Internet agents and cite system requirements such as coordination and natural language processing. We do not see the discussion of requirements and domains helps in evaluation.

David Kortenkamp, et al, report on their 3T architecture. They have implemented a sophisticated user-interface that allows a human operator to interrupt the system, to take over some tasks, and to redirect the system. This is the central notion of *traded autonomy*. Larry Pyeatt and Adele Howe present a two layered robot control architecture with design criteria of reliability, generalization, and adaptability. They argue that these qualities are interrelated. For instance, the low level of their architecture was designed to be adaptive and allowed for accumulation of learned actions and that provide the system with some level of reliability.

Lesser, et al, described their experimental environment Intelligent Home. They have carried our experiments on coordination and resource usage. They show that agents that need multiple resources and have to plan for their usage perform worse than reactive agents that need a single resource. This might be intuitive in itself but it is going to be used in developing quantitative metrics of coordination and resource management. Cyrus Nourani reports on multiagent concepts and fault-tolerance. Maurizio Piaggio, et al, discuss issue of scheduling concurrent robotic activities. The relevant issue of navigating robotic applications to this workshop is the quality of timeliness.

3. Conclusion

In this workshop, we discussed many of the issues surrounding the use of metrics in autonomous systems. One of the fundamental problems that we need to surmount is the inherent complexity of the agents we are describing. The submitted papers and abstracts have raised several possibilities for solving this problem: further research on system analysis, making higher-level metrics that can be more easily verified, or including run-time verification in our systems that allow us to make guarantees with less extensive design verification.

Another issue that we must overcome is the amazing diversity of these autonomous agents: how can we possibly generate metrics that applies to all of these systems? Again, the authors have provided several answers. First, we can discuss general types of metrics, such as robustness, timeliness, and reliability. While the specific definitions of these metrics differ from system to system, knowledge regarding the way in which they are measured and used can be transferred between disparate domains. Second, even if some metrics are only relevant in some domains and not others, we can discuss the manner in which we developed those metrics in one domain and apply those techniques for developing metrics to other domains.

It is most heartening that even the most theoretical of papers in this symposium are working with actual implementations. Furthermore, the span of papers from theory- to system-driven and across many different domains provided our discussions with disparate perspectives on the development and use of metrics for autonomous control software.

References

R. Arkin, 1998. **Behavior-Based Robotics**, MIT Press.

S. Brown, E. Santos, S. Banks, M. Oxley, 1998. Using Explicit Requirements for Interface Agent User Model Correction, International Conference on Autonomous Agents (Agents ’98).

E. Gat, 1998. Three-Layer Architectures, D. Kortankamp, P. Bonasso, R. Murphy (eds), 1998. **Artificial Intelligence and Mobile Robots**, MIT Press.

S. Hanks, M. Pollack, P. Cohen, 1993. Benchmarks, Testbeds, Controlled Experimentation, and the Design of Agent Architectures, AI magazine 14(4): 17-42, MIT press.

Hexmoor, H. editor, 1999. Autonomy Control Software, A workshop in conjunction with Autonomous Agents '99 conference, Seattle, WA. (http://www.cs.und.edu/~hexmoor/AA99.html)

H. Hexmoor, D. Kortenkamp, I. Horswill, 1997. Software architectures for hardware agents, **J. of Experimental and Theoretical AI**, 9, Taylor and Francis.

D.C. MacKenzie, R.C. Arkin, 1998. Evaluating the Usability of Robot Programming Toolsets, The International Journal of Robotics Research, 17(4):381-401.

D. Kortankamp, P. Bonasso, R. Murphy (eds), 1998. **Artificial Intelligence and Mobile Robots**, MIT Press.

B. Pell, G.A. Dorais, C. Plaunt and R. Washington, 1998. The Remote Agent Executive: Capabilities to Support Integrated Robotic Agents. In Working Notes of the AAAI Spring Symposium on Integrated Robotic Architectures, Stanford, CA.