Major Real-Time Challenges For Mechatronic Systems* (invited paper)†

Prof. John A. Stankovic
Dept. of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003

May 14, 1993

1 Introduction

While robots have been very successful in many environments, truly cognitive robots used in mechatronic systems have yet to appear. Such robots would sense, plan, act, and learn in the pursuit of dynamically varying objectives. The robots would have to be very flexible in dealing with hazardous and uncertain environments, sometimes alone, sometimes with other robots, and sometimes with direction from humans. Many of the tasks that such robots would be performing are subject to reliability and real-time constraints. Dealing with real-time constraints requires a major change to some of the paradigms and implementations previously used by AI researchers [3, 4]. For example, AI systems must be made to run much faster (a necessary but not sufficient condition [11]), allow preemption to reduce latency for responding to new stimuli, attain predictable memory management via incremental garbage collection or by explicit management of memory, include deadlines and other timing constraints in search techniques, and further develop anytime algorithms, time driven inferencing, and time driven planning and scheduling. Rules and constraints may also have to be imposed on the design, models, and languages to facilitate predictability. Understanding what predictability means for such systems is also important.

*This work has been supported in part by NSF under grants IRI 9208920 and CDA 8922572, by ONR under grant N00014-92-J-1048, and the Scuola Superiore S. Anna, Pisa, Italy, while the author was on sabbatical.

†To appear in *Proceedings of International Workshop on Mechatronical Computer Systems for Perception and Action*, June 1993.

Typical applications which might make use of this technology include teams of robots working together to dig up a gas pipe, or working in other hazardous environments such as nuclear power plants; or autonomous navigation in various environments such as on the surface of the earth, subsea, space, or on the surface of planets such as MARS. This brief paper identifies many challenges that must be met for such applications, but focuses on the real-time issues. We also briefly discuss several promising ideas that may serve as a basis for these systems (from the real-time perspective), and present suggested research directions. While most of the real-time challenges posed in this paper are discussed in terms of the grand challenge of truly cognitive robots, they also apply to much simpler real-time mechatronic systems.

2 Real-Time Challenges

The real-time challenges for mechatronic systems are divided into two classes: those that are more specific to mechatronic systems (Section 2.1), and those that are more general within the field of real-time computing [12] (Section 2.2).

2.1 Specific to Mechatronic Systems

Major real-time challenges for Mechatronic systems include:

- **learning timing behavior**: can a system learn what deadlines are and adapt to meet these deadlines in a predictable manner; what mechanisms are required to permit self analysis with respect to time; can neural networks handle sophisticated timing requirements [9],
- **re-learning**: due to environmental changes, new objectives, or faults can the system un-learn *in time* and re-learn new timely and predictable behavior; it seems incredibly difficult to characterize the timeliness of the learning process itself,
- **tradeoff analysis**: exact answers often cannot be achieved in time, if ever, therefore techniques to trade off quality of solution for processing time are required,
- **worst case execution time**: can AI programs be effectively bounded without limiting flexibility and learning,
- **controlling variance**: processing times for individual tasks, for composition of tasks including blocking, for outputs produced and the time at which they are produced, etc. may have high variance making it difficult to design a cost effective solution; we need better techniques to address possible high variance, especially with respect to meeting deadlines,

- **actively controlling deadlines**: in many situations it may be possible to actively seek to modify deadlines, e.g., by slowing down the robot or by reducing the frequency of when a sensor is polled; this may be one technique for controlling some aspects of variance; also, what deadlines are hard and which deadlines are soft and can they be controlled or manipulated,

- **system integration**: how can large numbers of tasks be composed to handle cognitive situations, yet their net timing performance be predictable,

- **distributed control**: moving more intelligence to the reactive front ends of the system increases ability to respond quickly (in time), but may increase communication costs and the need for distributed cooperation possibly slowing global decisions; distributed control requires the integration of scheduling of cpu and communication at two levels: between the distributed entities of reactive front end control and from this front end with higher level cognitive entities; see also the adaptivity and stability issues raised below,

- **time bounded communication**: real-time datagrams and real-time virtual circuits with quality of service levels are required [1]; vast amounts of information may be involved, e.g., in vision subsystems; interaction of the communication protocols with the kernel for end-to-end timely performance is required,

- **off-line vs. on-line**: what is the impact of off-line *a priori* knowledge together with subsequent design and implementation decisions made on the on-line operation of the system, both for functionality and timing,

- **faults**: the integration of handling faults and real-time constraints in a dynamic on-line manner is in its infancy; mechatronic systems should already be dealing with noisy, uncertain, missing and incorrect information [3, 5], but how this relates to handling faults and recovery in time as part of producing dependable systems has not been explored well, and

- **adaption and stability**: dynamically adapting *in time* to uncertain environmental inputs is very difficult especially when trying to guarantee stable performance.

2.2 More General Real-Time Challenges

The science and technology of real-time computing has progressed significantly over the last 10 years. However, many open questions remain. Many of these must be solved in order to provide a well understood and analyzable real-time foundation (software and hardware) upon which to develop the cognitive layers. For example, if a cognitive level planner is reasoning about time and decides to

execute certain tasks, if the underlying system is not predictable and controllable then the careful time planning may easily be violated by the actual run-time system. More general, but key real-time problems include:

- can we formally analyze the timing properties of the system,
- how should we specify timing constraints which are flexible, dynamically change and have more sophisticated semantics than addressed to date by the real-time community,
- what languages and language features are appropriate,
- what RTAI and system level paradigms [14] are or are not appropriate and how do they affect each other,
- what interfaces should exist between the OS and the reactive front-end and between the OS and the cognitive layer [13],
- how should various subsystems be integrated such as vision and robot control,
- can we solve the dilemma of wanting a predictable, analyzable, safely behaving system, but one which operates in highly uncertain environments and learns,
- how to exploit high performance parallel computing for real-time problems,
- what should be the comprehensive and integrated scheduling approach for networks of multiprocessors,
- what part will real-time databases play in the solution (note, very limited work has been done on real-time databases), and
- what improvements to design tools and its associated analysis are required to engineer such systems.

3 Promising Technology

To support mechatronic systems, real-time systems technology must provide a higher degree of on-line adaptability and flexibility than is typically found in most of today's real-time systems. Here we briefly discuss two promising research directions: reflective real-time systems, and multi-level and multi-dimensional scheduling algorithms. Embedded in both of these promising directions is adaptive fault tolerance.

3.1 Reflective Real-Time Systems

Reflection is defined as the process of reasoning about and acting upon the system itself. Reflection is not new. It has appeared in AI systems, e.g., within the context of rule based and logic based languages [7, 10]. More recently it is being used in object-oriented languages and object-oriented databases in order to increase their flexibility [6, 8]. Combining object oriented programming with reflection seems to be very important since object oriented programming supports abstraction and good design and adding reflection supplies flexibility. Reflection also has been touted as valuable to distributed systems for supporting transparency and flexibility [18]. While reflection is not new, using reflection is real-time systems is new. In fact, the only work we are aware of is our own work [15, 16].

We believe that the notion of a reflective architecture can serve as a central principle for building complex, flexible, and fault tolerant real-time systems, and, if used correctly, can support nice timing analysis properties because the notion of time and certain implementation aspects are visible across the layers. This visibility seems to be required to do realistic timing analysis, but is at odds with typical design that hide all implementation issues. Further, by identifying reflective information, exposing it to application code, and retaining it at run time, a system is capable of reacting in a flexible manner to changing dynamics in the environment including faults, to better evolve over time, and even for better monitoring, debugging, and understanding of the system. It is also a common understanding that we want integrated system-wide solutions so that design, implementation, testing, monitoring, fault tolerance, and validation are all addressed. We argue again that what is required is a reflective system architecture that **exposes** the **correct** meta-level information so that the application programmer and various tools can utilize it at design and implementation time, and the actual system can use it at run time. Exactly what this information should be and how it should be supported will be the subject of research for many years. An example of reflective information for real-time computing at the kernel level can be found in [15, 16]. Also, in [2], a structure for adaptive fault tolerance is proposed where adaptive control and management of redundancy under time constraints is supported by identifying and exposing reflective information.

3.2 Multi-Level and Multi-Dimensional Scheduling

One key challenge we face in building complex real-time systems is that they must be both flexible and predictable. To do this we must first move away from completely static solutions. For example, in static scheduling we assume that we know all the tasks, all their deadlines, worst case computation times, importance levels, precedence constraints, communication requirements, fault tolerance requirements, resource requirements, and their *future* arrival times.

This is unrealistic. Second, we must control how dynamic the system can be. For example, for most real-time systems, it is reasonable to require that we know everything in the above list *except* the future arrival times. For a RTAI system less information may be known, but even for these systems we cannot regress to the completely dynamic case and hope to analyze the timing properties of the system. A completely dynamic situation is acceptable for general purpose computing, but not for real-time computing. All of this information that we must acquire is part of the reflective architecture we propose. Next, we must decompose the system according to functionality, granularity of timing constraints, fault tolerance requirements, and task criticalness. This will entail different scheduling solutions in different subsystems and for different granularity and criticalness requirements. These multi-level scheduling solutions must sometimes be integrated across subsystem interfaces, although a goal is to minimize such interactions. Also, it may also be necessary to schedule in a multi-dimensional manner, meaning simultaneously accounting for cpu, data resources, I/O resources, precedence constraints, etc. Otherwise, unpredictable blocking may occur, thereby causing missed deadlines.

In such a system, the types of performance guarantees would vary depending on cost, subsystem, criticalness, timing granularities, fault hypotheses, and expected workloads including overloads. For example, it should be possible to guarantee 100% of critical tasks subject to fault hypotheses made when allocating time for these tasks, and have a quantifiable graceful degradation in overload without losing any critical tasks. If unexpected overload were to occur on the critical tasks (beyond what was conservatively planned for), then the system should be flexible enough to, in a best effort manner, borrow processing time from other tasks. In a more static system design the system would just fail at this point.

4 Research Directions

While it is obvious that research advances on any of the multitude of individual topics presented here may improve the likelihood of success, I propose several more *integrated* research directions, including:

- integrate research in intelligent sensors and actuators, robotic control, distributed control, real-time scheduling, and real-time kernels or microkernels,
- integrate research in cognitive level real-time processing with a real-time predictable platform consisting of a real-time kernel and hardware; part of the work here would be to develop good interfaces between the cognitive and system layers, and
- combine the first two suggestions into an integrated system.

To produce good research in these three areas requires a fairly large and interdisciplinary team. It is probably best to be somewhat application driven, at least at first, and then try to generalize. For smaller teams or even as a good strategy for incremental development, one could assume much simpler cognition levels, dynamics and goals than the ultimate goal of a truly cognitive robot. With the proper base of a predictable and analyzable framework, one could then try to incrementally improve it without losing these nice properties.

5 Summary

This paper has presented a wealth of open problems and challenges, many very difficult to solve. It is unfortunate that so many problems still remain, but that is reality. Too often the difficulty of satisfying timing constraints in a scientific manner is underestimated or ignored, e.g., by treating real-time as fast computing. For a discussion of this common misconception as well as others see [11]. One challenge, then, might be to not repeat the mistakes of the past where time was treated as an afterthought, rather we must elevate it to a central principle in the system design and implementation. In fact, possibly the first question that must be answered when building such systems is how will we represent and deal with *time* in the deployed system. Many research directions can be envisioned for this emerging field, but those that stress integrated solutions across hardware and software, across reactive levels to the highest levels of cognition, and across functional and time requirements, will be the most valuable.

6 Acknowledgments

I would like to thank Rod Grupen and Giorgio Buttazzo for their comments on an earlier version of this paper.

References

[1] K. Arvind, K. Ramamritham, and J. Stankovic, A Local Area Network Architecture for Communication in Distributed Real-Time Systems, invited paper, *Real-Time Systems Journal*, Vol. 3, No. 2, pp. 113-147, May 1991.

[2] A. Bondavalli, J. Stankovic, and L. Strigini, Adaptable Fault Tolerance for Real-Time Systems, *Proc. Predictable Distributed Computing Systems*, September 1993.

[3] K. Decker, V. Lesser, and R. Whitehair, Extending a Blackboard Architecture for Approximate Processing, *Real-Time Systems Journal*, 2, pp. 47-79, 1990.

[4] B. Hayes-Roth, Architectural Foundations for Real-Time Performance in Intelligent Agents, *Real-Time Systems Journal*, 2, pp. 99-125, 1990.

[5] E. Horvitz and G. Rutledge, Time Dependent Utility and Action Under Uncertainty, *Proc. of the Sixth Conference on Uncertainty in AI*, Los Angeles, July 1991.

[6] M. Ibrahim, editor, *Proc. OOPLSA 91 Workshop on Reflection and Metalevel Architectures in Object Oriented Programming*, October 1991.

[7] P. Maes, Concepts and Experiments in Computational Reflection, *OOPSLA 87*, Sigplan Notices, Vol. 22, No. 12, pp. 147-155, 1987.

[8] S. Matsuoka, T. Watanabe, Y. Ichisugi, and A. Yonezawa, Object Oriented Concurrent Reflective Architectures, 1992.

[9] W. Miller, R. Sutton, and Paul Werbos, editors, *Neural Networks for Control*, MIT Press, Cambridge, Mass., 1990.

[10] B. Smith, Reflection and Semantics in a Procedural Language, MIT TR 272, 1 982.

[11] J. Stankovic, Misconceptions About Real–Time Computing: A Serious Problem For Next Generation Systems, *IEEE Computer*, Vol. 21, No. 10, pp. 10-19, October 1988.

[12] J. Stankovic and K. Ramamritham, *Hard Real–Time Systems*, Tutorial Text, IEEE Computer Society Press, Wash. DC, 618 pgs., 1988.

[13] J. Stankovic, K. Ramamritham and D. Niehaus, On Using The Spring Kernel To Support Real-Time AI Applications, *Proc. EuroMicro Workshop on Real-Time Systems*, June 1989.

[14] J. Stankovic and K. Ramamritham, The Spring Kernel: A New Paradigm for Real-Time Systems, *IEEE Software*, Vol. 8, No. 3, pp. 62-72, May 1991.

[15] J. Stankovic, On the Reflective Nature of the Spring Kernel, *invited paper*, *Proc. Process Control Systems '91*, February 1991.

[16] J. Stankovic, Reflective Real-Time Systems, in preparation May 1993.

[17] J. Stankovic, Distributed Real-Time Computing: The Next Generation, invited paper, special issue of *Journal of the Society of Instrument and Control Engineers of Japan*, Vol. 31, No. 7, pp. 726-736, 1992.

[18] R. Stroud, Transparency and Reflection in Distributed Systems, position paper for *Fifth ACM SIGOPS European Workshop*, April 1992.