Proceedings of the 1999 IEEE
International Conference on Robotics & Automation
Detroit, Michigan • May 1999

Open Architecture Controller Software for Integration of Machine Tool Monitoring

Shige Wang, C.V Ravishankar, Kang G. Shin
Real-Time Computing Labroatory
Department of Electrical Engineering & Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
email: {wangsg,ravi,kgshin}@eecs.umich.edu

Abstract

In contemporary machine control systems, the monitoring functions are developed and tested separately, requiring additional time and effort for their integration into a machine control system. Also, the software for most contemporary controllers is fixed and very application-dependent, so the system may not run correctly after such integration since the algorithms are time-sensitive. In this paper, we show how to modularize machine tool control systems with object-oriented concepts. We define a set of software components and system services for reuse, present some system guidelines based on simulations and test analyses to help users implement controllers that satisfy real-time constraints.

We present the integration of broken tool detection functionality into an existing three axis motion controller, and demonstrate that the integration requires minimal effort and skill, and that the hard real-time constraints for broken tool signal processing can be satisfied with our software architecture.

Key Words: Machine tool monitoring, real-time control, software architecture, open architecture controller

1 Introduction

Machine tool monitoring improves economy and safety when used on-line to monitor run-time status. To facilitate integration of machine tool monitoring, the machine control systems software must be well modularized [12], have public interfaces to system internal information [4, 6], and have the ability to specify control flow [6, 10]. Integration of machine tool monitoring is difficult in current machine control systems since the software modules are designed for specific applications and implemented so that no internal component, variable or algorithm is accessible [4, 8]. While such isolation has advantages, the machine tool monitoring modules cannot work with other modules to control a machine correctly without exchanging some internal information. Fixed software also reduces the flexibility and portability of a system, and makes it difficult to change the control flow after integration. Currently, all the control flow information is embedded in the application code, and such information can be extremely hard to understand [6].

Machine tool monitoring applications also have stringent hard real-time constraints. Since most control algorithms are sensitive to time variations [2, 7], it is important to sample, process and pass data around the system in a timely fashion. Such constraints may include bounds on signal passing delays and on the variances of execution intervals of real-time execution modules [5, 11, 12]. No previous research in this domain addresses this problem at the software architecture level.

It is widely agreed that open architecture controllers can help solve the above problems and enable quick and easy integration of new functionality [6, 8, 12]. But the open architecture controller products currently in use have very limited capabalities, mainly because they only provide ad-hoc openness [9]. The software is usually application-specific, and has to be modified after integrating machine tool monitoring [4, 6, 8].

Our software architecture includes components to describe machines, infrastructures to define execution environments, and mechanisms to specify the control flow. These features distinguish our research from others' in this domain.

The rest of this paper is organized as follows: Section 2 describes our software architecture and its organization. Section 3 presents how machine tool monitoring can be integrated into a system with our software architecture. Section 4 provides a detailed example implemented on The University of Michigan Open Architecture Controller (UMOAC) testbed. Finally, we give our conclusion in Section 5.

2 Software Architecture

The software components in our architecture can be classified into two categories: *generic components* for machine definitions and *standard system services* for platform descriptions.

Figure 1 shows relationships among these software sets, machines, platforms and applications.

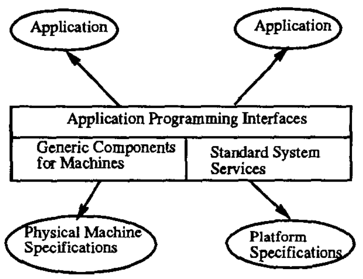
Application
Application
Application Programming Interfaces
Generic Components for Machines
Standard System Services
Physical Machine Specifications
Platform Specifications

Figure 1: Relationship between software, application, machine and platform

2.1 Generic Components

A machine control system can be subdivided into many smaller components such as axes, spindles and sensors. Each component can be represented as an object with the appropriate functionality. A generic component can execute operations defined as its member functions to drive a machine component. Machine operation is a matter of the order these functions will be called. We can thus separate the definition of the behavior of a machine from its functionality, and change its operation without changing its control software.

Our generic components are organized hierarchically. Blocks on the left in Figure 2 describe the hierarchical organization of these generic components.

2.2 Standard System Services

Standard system services define a virtual execution environment where a generic component can run as desired, and meet its real-time requirements. Our standard system services include definitions of execution units, communication mechanisms, and real-time services.

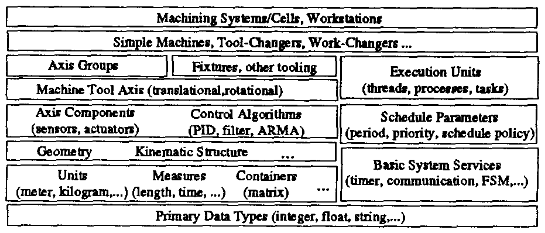
Machining Systems/Cells, Workstations
Simple Machines, Tool-Changers, Work-Changers ...
Axis Groups
Fixtures, other tooling
Execution Units (threads, processes, tasks)
Machine Tool Axis (translational,rotational)
Axis Components (sensors, actuators)
Control Algorithms (PID, filter, ARMA)
Schedule Parameters (period, priority, schedule policy)
Geometry
Kinematic Structure
...
Basic System Services (timer, communication, FSM,...)
Units (meter, kilogram,...)
Measures (length, time, ...)
Containers (matrix)
...
Primary Data Types (integer, float, string,...)

Figure 2: Hierarchical organization of software classes

Execution units. An execution unit is a container in which generic components can reside and execute as separate processes or threads on a processor. Execution units are classified into periodic tasks, aperiodic tasks and pseudo-periodic tasks.

Communication mechanisms. Communication mechanisms define how a component exchanges information with other components. We define three kinds of basic communication services: shared memory, message passing and signals.

Real-time related services. Real-time services include timing and scheduling services, which provide a high-resolution timer and mechanisms for defining the scheduling policy (e.g., preemptive or run-to-complete, FIFO or Round-Robin) and priority management. The platform specifications determine which services are available and how they are implemented.

The blocks on the right of Figure 2 present the organization of standard system services in our architecture.

2.3 Control Flow Specifications

Control flow defines how to sequence the operations or commands that a machine control system will execute at runtime. In many cases, integration of machine tool monitoring will not change the setup of the existing machine, but only the control flow. The definition of appropriate generic components makes it possible to specify control flow separately. A Finite State Machine (FSM) based mechanism is used as an engine to execute a user-specified control flow. A user can define and modify the system control flow using this mechanism without changing source code. A user can also split the FSM for a complex system into several small FSMs for subsystems, define and test the control flow specifications for each subsystem and put them together. Details are available in [10]

3 Adding Monitoring Functions

The open architecture software described above provides modularity and openness for machine tool controllers, and enables machine tool monitoring software to be integrated into an existing control system at different levels. This section describes how the machine tool monitoring is integrated.

Defining Functionality. To integrate machine tool monitoring, both new and current functionality for machine tool controllers must be defined with generic components. The machine tool monitoring modules can either be built from scratch using the generic components and services, or purchased as a software package.

For existing control software, we assume that the interfaces to exchange information with the machine tool monitoring module are present. These interfaces enable the machine tool monitoring module to read the necessary internal states and data from the system and send its commands.

Defining Execution Model on a Platform. To execute the machine controller on the selected platform, the modules in the system are grouped into one or more execution units, and mapped to an implementation on the platform.

Timing services are also defined with the execution units. Users can select a timer for each execution unit or for a set of closely-coupled units to obtain better performance. Scheduling parameters for execution units must also be assigned if the platform supports multiple choices.

Communication channels are created to support information exchange both within a single execution unit and across different execution units. The selection of a communication mechanism depends on both system time constraints and the execution environment.

Changing Control Flow Changing the control flow involves defining new events for machine tool monitoring, modifying the state table of FSM, and invoking corresponding the member functions of generic components.

The new events serve to notify the control system of abnormal tool states that a tool monitoring module may detect, such as tool wear or breakage. A user defines a new event by giving it a global unique identifier (for example, a name or a number). A control module to react to the event may have its own definition of the same event, called a *local event.*

The FSM state table specifies how to map an event to member functions of the generic components. Integration of machine tool monitoring may require a machine to operate in a different way, which implies changing state table.

Verifying Real-Time Constraints. Real-time constraints must be verified after the machine control modules are defined and allocated to a selected platform.

Two parameters of each execution unit must be checked: *execution interval consistency* and *conformance to deadlines.* Consistent execution intervals ensure that information on systems and environmental states is collected at the right time. Deadline checking ensures that an execution unit can finish the computation within the given time slot.

Time delays for data transfer also must be verified. Time delays for message passing are determined mainly by three factors: number of execution units in the message-passing path, the scheduling parameters for these execution units (periods, priorities, etc.), and the communication mechanisms used.

Real-time performance must be verified both for the newly-integrated machine tool monitoring modules and for old software modules. After integration, the behavior of existing modules may have changed due to the increased workload and communication in the system.

4 A Case Study

To demonstrate that our open architecture software meets the requirements of machine tool monitoring integration, we integrated a broken tool detection module into a motion controller on the UMOAC testbed. We study and analyze the effort for integration as well as system performance after integration.

4.1 UMOAC Testbed

UMOAC testbed is built on a three-axis milling machine. The controller software runs in a distributed environment on three computers connected point-to-point with 10-Base T Ethernet, as shown in Figure 3. Two Intel/386-based VME machines run the real-time operating system QNX, and the third Pentium machine runs Windows NT. The existing motion controller software consists of force acquisition, force supervisory control and motion control modules. Motion control consists of four execution units: the AxisGroup Control unit to coordinate the motion of the three axes, and one Axis unit each for controlling motion of X, Y and Z axis. The execution units of each module and their allocation are shown in Figure 4.

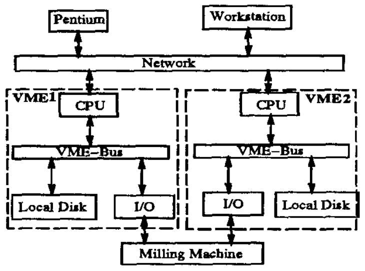

Figure 3: UMOAC testbed

4.2 Requirements and Design

The broken tool detection algorithm uses force values to determine the status of a tool. A tool breakage signal is sent to the motion control module to trigger a timely stoppage when an abnormal value is detected. The broken tool signal is a real-time signal subject to the constraints that the signal passing delay be within two execution cycles of motion controller.

A new module with two execution units is added to process the force values and generate a broken tool signal. The control flow of the motion controller is changed to react the broken tool signal.

To evaluate performance with different integration granularities and accessibility, we integrate the broken tool detection module at two different levels. In the case of motion control level integration, the signal is sent to motion controller. In the other case, the signal is sent to each axis control module.

4.3 Implementation

Figure 4 shows the modules, execution units and the communications in our testbed.

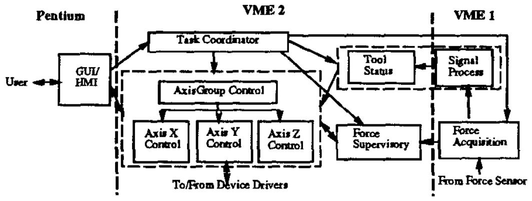

Figure 4: Execution units in the experimental system

Table 1 describes characteristics of execution units. Since non-real-time modules are allocated on a separate computer and do not effect the performance of real-time modules, we exclude them from the analysis.

Execution Unit (Tasks)	Type	Period (ms)	Priority	Location
Force Acquisition	Periodic	1	20	VME 1
Force Supervisory	Periodic	40	16	VME 2
Motion Control	Periodic	10	20	VME 2
AxisX Control	Periodic	10	21	VME 2
AxisY Control	Periodic	10	21	VME 2
AxisZ Control	Periodic	10	21	VME 2
Task Coordinator	Aperiodic	N/A	17	VME 2
New execution units for broken tool detection				
Signal Process	Periodic	10	18	VME 1
Tool Status	Periodic	10	18	VME 2

Table 1: Execution units

The execution units that interact with the external world (including *force acquisition* and *Axis X, Y, Z control*) and running control algorithms to generate commands (include *force supervisory control*, *AxisGroup Control*, *signal process*, and *tool status*) execute periodically. Execution units are scheduled using the rate-monotonic algorithm [5]. Units with the same priority are scheduled and executed first-in-first-out.

A new event, E_{broken}, is introduced into the system to represent a tool breakage status. This event is generated by the tool status to the motion control module.

A new state, *bstop*, and a transition from the moving state to *bstop* are added into the machine control FSM. Figure 5 shows the state diagram changes before and after integrating broken tool detection module.

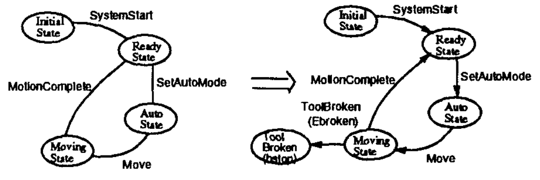

Figure 5: Simple state diagrams before and after integration

Only the FSM specifications for the machine control FSM in the motion control module need changes. Other FSM specifications, including the higher level and lower level of the machine control FSM, need no changes.

4.4 Experimental Results

We no discuss the effort needed to integrate the broken tool detection module, and examine real-time performance of individual execution units, signal processing, and message passing.

Building a system with our proposed software architecture reduces the skill and effort required. According to our experience from graduate students and undergraduate students, integrating a simple machine monitoring function, such as broken tool detection as described above, using our open architecture software takes significantly less time than using unstructured software. For instance, it took tens of hours less to integrate a broken tool detection module into a motion control system using our proposed open software architecture than to integrate the broken tool detection in a traditional way.

We analyze real-time performance in terms of two parameters: stability of execution intervals and percentage of missed deadlines, both before and after integration. We also examine the time delay for passing and processing an emergency message, such as a broken tool signal. We use a special hardware device, the VME StopWatch, to collect the elapsed time of each execution.

4.4.1 Interval Consistency and Missed Deadlines

Tables 2 and 3 show timing results of each execution unit in the system before and after integration.

Execution Unit (Task)	Average Execution Interval (ms)	Worst-Case Execution Interval (ms)	Standar Deviation	Deadline Missing (%)
Force Acquisition	0.998	2.182	0.085	0.11
Force Supervisory	39.976	42.365	0.132	0.35
AxisGroup Control	9.976	14.668	0.185	0.27
AxisX Control	9.965	12.384	0.085	0.24
AxisY Control	9.943	14.132	0.068	0.18
AxisZ Control	9.971	12.379	0.073	0.21

Table 2: Before integration

Execution Unit	Average Execution Interval (ms)	Worst-Case Execution Interval (ms)	Standard Deviation	Deadline Missing (%)
Force Acquisition	0.998	2.834	0.086	0.15
Force Supervisory	39.968	48.766	0.332	0.32
AxisGroup Control	9.972	16.731	0.241	0.29
AxisX Control	9.965	16.925	0.133	0.15
AxisY Control	9.958	15.332	0.125	0.21
AxisZ Control	9.971	15.784	0.142	0.21
Signal Process	19.963	25.429	0.103	0.35
Tool Status	19.974	27.471	0.177	0.18

Table 3: Performance of units after integration

We observe no significant difference between average execution intervals for execution units before and after integration. Similarly, there is no significant difference observed for missed deadlines. But both the worst-case execution times and standard deviations of intervals are slightly increased after the broken tool detection modules are integrated.

Based on these observations and real-time schedulability analysis [5], we can ensure that whenever new execution units integrated into our system are schedulable, their real-time constraints can be satisfied. The operating system overheads can be assumed to be constant. However, the execution intervals become inconsistent as the number of execution units increases. Thus, if a module assigned to a processor requires more consistent execution intervals, the number of execution units on that processor should be reduced.

4.4.2 Signal processing and transportation

Tables 4 and 5 shows the signal passing times in our experimental system.

	Signal passing path		Communication mechanism	Elapsed time (ms)
	From	To		
Case 1	Force Acquisition	Signal Process	SM	0.357
	Signal Process	Tool Status	MQ	2.578
	Tool Status	Motion Control	SM	6.176
	Motion Control	Aixs Control	SM	5.217
	Total			14.328
Case 2	Force Acquisition	Signal Process	MQ	0.675
	Signal Process	Tool Status	MQ	3.152
	Tool Status	Motion Control	MQ	8.157
	Motion Control	Axis Control	MQ	7.392
	Total			18.896

Table 4: Signal passing time (Motion controller as a whole)

	Signal passing path		Communication mechanism	Elapsed time(ms)
	From	To		
Case 1	Force Acquisition	Signal Process	SM	0.386
	Signal Process	Tool Status	MQ	2.762
	Tool Status	Axis Control	SM	7.462
	Total			10.71
Case 2	Force Acquisition	Signal Process	MQ	0.584
	Signal Process	Tool Status	MQ	2.835
	Tool Status	Axis Control	MQ	9.073
	Total			12.492

Table 5: Signal passing time (Axis controller is accessible)

Comparing cases 1 and 2 in Tables 4 and 5, the time cost of communication using shared-memory is shown to be less than that using message queues. This is because the message queue mechanism invokes more service calls of the operating system, introducing more unpredictability and delay. Therefore, shared-memory should be preferred when a short delay is required and the receiver needs to access the message immediately. The most significant disadvantage of using shared-memory mechanism is that it can only be used within the same node. If the communication is across computers, other communication mechanism has to be used.

Another observation from Tables 4 and 5 is that a finer granularity of accessibility increases message passing times. The total elapsed times are much shorter in Table 5 than in Table 4 for both cases. This shows that fewer execution units on the communication path will shorten the signal passing time. Thus, better accessibility will help satisfy real-time constraints. However, a better accessibility may involve more execution units in the system, which will also increase the overheads of the whole system and make executions of each unit inconsistent.

5 Conclusion

Integrating process monitoring functionality into an existing controller requires more effort and skill in current manufacturing practice. In this paper, we propose PC-based open architecture software that can make this integration easier. The software consists of building blocks with standard interfaces that are used to define the machine, the execution platform, and the control flow. We also describe how to integrate process monitoring at different system levels with different granularities. Our software is shown to be flexible enough to support integration with various kinds of controllers. Our case study of integrating broken tool detection into a motion controller on the UMOAC testbed demonstrates that our software can be used to integrate process monitoring functionality, satisfying openness and real-time requirements.

Acknowledgments

The work reported in this papaer was supported in part by the National Science Foundation (Grant # EEC-9529125) Engineering Research Center for Reconfigurable Machining Systems. We would also like to thank Sushil Birla of General Motors, and Mike Washburn at the University of Michigan for their assistance.

References

[1] S. Birla, "Software modeling for reconfigurable machine tool controllers" *Ph. D. Thesis*, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, June 1997

[2] R. Du, M.A. Elbestawi, and S.M. Wu, "Automated monitoring of manufacturing processes, Part 1: Monitoring Methods," *Journal of Engineering for Industry, Transactions of the ASME,* vol.117, No.2, pp. 121-132, May 1995.

[3] ESPRIT Consortium AMICS, *CIMOSA, Open System Architecture for CIM,* 2nd revised and extended edition, Springer-Verlag, 1989.

[4] Y. Koren, F. Jovane, and G. Pritschow, *Open Architecture Control Systems: Summary of Global Activity,* ITIA series, vol. 2, 1998

[5] C. M. Krishna, and K. G. Shin, *Real-Time Systems,* The McGraw-Hill Companies, Inc, 1997

[6] OMAC working group, *OMAC API Documentation,* version 0.18, 1998.

[7] S.M. Pandit, S.M. Wu, *Time Series And System Analysis With Applications,* John Wiley and Sons, 1983.

[8] F.M. Proctor, and J.S. Albus, "Open architecture controllers," *IEEE Spectrum,* vol. 34 no. 6, pp. 60-64, Jun 1997.

[9] G. Rice, R. Moreno, and M. King, "Process data acquisition: Real-time and historical interfaces," *Proceedings of SPIE Open Architecture Control Systems and Standards,* Boston, Ma, pp. 196-206, Nov. 1996.

[10] C. Shiu, et al, "Specifying reconfigurable control flow for open architecture controllers," *Proceedings of 1998 Japan-USA Symposium on Flexible Automation, Vol.2,* Otsu, Japan, pp. 659-666, July 1998.

[11] J. Xu, and D. Parnas, "On satisfying timing constraints in hard real-Time systems,"*IEEE Transactions on Software Engineering,* Vol 19, No.1, pp. 70-86, Juan. 1993.

[12] L. Zhou, M. Washburn, K. G. Shin, and E. A. Rundensteiner, "Performance evaluation of modular real-time controllers," *Proceedings of the ASME Dynamic Systems and Control Division,* DSC-vol. 58, Atlanta, GA, pp. 299-306, Nov. 1996.