Fourth International Software Architecture Workshop, in Conjunction with the 22nd International Conference on Software Engineering, Limerick, Ireland, June 4-5, 2000.

An Architecture and Tool for Large-scale System Control

with a manufacturing system application

Hui-Min Huang, James Albus, William Shackleford, Harry Scott, Tom Kramer,
Elena Messina, and Fred Proctor

National Institute of Standards and Technology
Gaithersburg, Maryland 20899

Abstract

This paper describes a reference architecture that is applicable to multiple classes of large-scale, complex real-time control systems. An associated tool, Generic Shell, is also described. Generic Shell employs a set of code templates that facilitate system development and integration. A case study is presented.

1 Introduction

A desired large-scale manufacturing environment may contain many pieces of equipment that are physically distributed and functionally integrated. The operation of such systems may be a combination of automatic control and manual operations. It is imperative to use an approach that allows easy development, testing, operation, and maintenance of these systems. Researchers have been pointing out that software architecture is a viable approach [1, 2, 3, 4].

There have been several major manufacturing architectural development efforts. Open, Modular Architecture Control (OMAC) [5] is industry-driven, takes a bottom-up approach, and focuses on low-level machine control. Computer Integrated Manufacturing Open System Architecture (CIMOSA) [6, 7] describes multiple views of a system. However, CIMOSA seems to focus on enterprise modeling and not on low-level, time critical control subsystems. Antsaklis and Passino described a hierarchical control architecture, which contains a concept of successive delegation of duties among the defined levels [8]. However, the defined three-level organization has not been demonstrated to be suitable for a variety of problems. Senehi and Kramer [9] proposed a framework, including a terminology, that set up a foundation for constructing control system architectures.

We propose that an architecture should possess the following characteristics so that the associated benefits can be provided for the system developers:

- An architecture is to be generic. This makes it applicable to many types of problems.
- An architecture is to provide a reference model for system organization. This provides a basis for the system to be developed and facilitates the system's understandability.
- An architecture is to provide common component and interface structures and a common execution mechanism throughout the entire architecture. These make the architecture scalable and reconfigurable to both large and small problems, ease the maintenance tasks, and make the components more reusable.

- An architecture is to include methods and/or tools that guide developers through the design and implementation processes.

A reference model architecture, called ISAM (Intelligent System Architecture for Manufacturing) satisfies the first three characteristics. A tool called Generic Shell, addresses the fourth characteristic. Generic Shell models the generic functions and interfaces of ISAM with a set of C++ code templates. Developers build a skeleton control system by connecting the templates. The developers then link the application-specific behavioral or processing algorithms or software components in the functional templates and populate the interface templates.

In this paper, we describe the generic functional unit of ISAM first, followed by the system structure. We then describe the multiple layers of abstraction that transition the architectural model to real systems. We describe the architecture in a way that facilitates implementation. Modularization, object-orientation, common process and interface structures, and system openness are among the benefits of our architecture and tool.

We demonstrate that, in ISAM, the generic architecture aspects and the Generic Shell tool aspect form an integrated conceptual framework. An implementation is illustrated.

2 Common Functional Unit

ISAM is based on the Real-time Control System (RCS) reference model architecture [10], also developed at the National Institute of Standards and Technology. ISAM is a hierarchical control architecture with a generic control unit populating at all the control levels. Specific responsibilities and associated processing algorithms are assigned for each of the common control units.

ISAM is a goal or task driven control model. A control unit generates behaviors or actions based on the task commands that the control unit receives. The effectiveness of the actions or behaviors depends on the support from the control unit's sensory processing and knowledge base update functions. We call this control unit a control node. Figure 1 depicts the control node functions. A detailed description for all the functions follows.

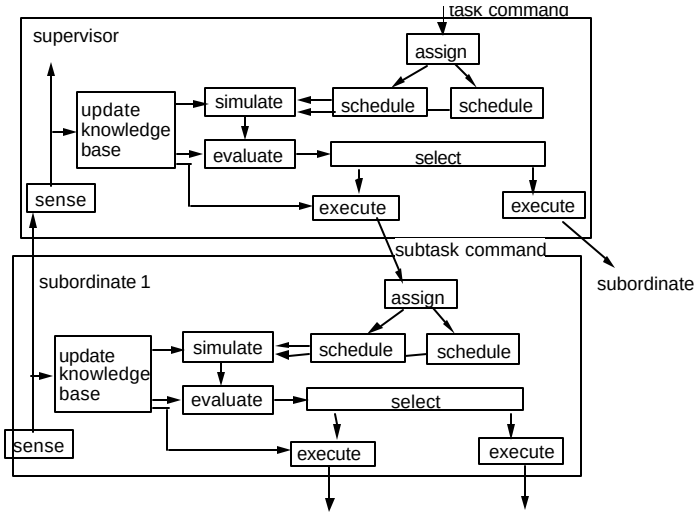

Figure 1: ISAM Control Node Functions

2.1 Planning and execution in the node

The behavior generation activity is characterized by a planning and execution process. The planning process generates schedules or plans. As shown in Figure 1, the planning process begins when the job assignor ("assign" in the figure) receives a task command. "Assign" retrieves required task knowledge and interprets it. This results in a set of scheduling jobs for a set of schedulers. ISAM specifies a scheduler for each of the subordinate control nodes that a control node supervises.

The schedulers ("schedule" in the figure) perform multiple aspects of planning, including spatial, temporal, and coordination. The specific concerns include coordinate frame transformation, path generation, subtask generation and sequencing, subtask timing assignments, and

tools/ material/ timing/ peer-status requirements.

The schedulers produce plan or schedule candidates and send them through other node functions, simulation and value judgement, as shown in the figure. These two functions verify the controllability and execution costs for the candidates, respectively. Plan selector ("select" in the figure) selects the best candidate for execution. The plans are constantly re-evaluated and replanned. The frequency of replanning depends on degree of uncertainty in the environment.

Executors are responsible for executing the generated plans. When errors occur, the executors may perform reactive, emergency functions to transition the node to safe states. The schedulers may then either replan, within the node's assigned authority, to attempt to achieve the original goals or may report errors to the node's superior node requesting further direction. The execution results in subtask commands being sent to the node's subordinate control nodes.

2.2 Knowledge-based behavior generation

An ISAM node performs information processing and maintains a knowledge base. The system knowledge in each node must be rich and updated in real time to support the planning and execution of the tasks. The knowledge may include task identifiers, plans, algorithms, resource descriptions, data logs, shop floor layout, maps, etc. The knowledge can be stored either externally in databases or in-line in code, depending on retrieval efficiency to support real-time behavior generation.

The knowledge base is updated with sensory information in real-time.

2.3 Functional decomposition and aggregation

As described, a control node is decomposed into the node functions. During implementation, to satisfy the concern of computation and communication load, these node functions may be combined or further decomposed to form a different but corresponding computing process configuration. Our case study, described later in this paper, demonstrates this effect.

3 Control Nodes to Form Canonical, Control Hierarchies

Hierarchical systems have demonstrated efficiency in performing system control [11]. ISAM prescribes a canonical hierarchical control structure. High levels handle tasks with larger spatial and temporal scope but with less detail. Low levels handle detailed and specific tasks that focus on immediate temporal and spatial span. The number of levels is scalable depending on the scope of the systems.

As we described in section 2, the control node function inputs task commands and outputs subtask commands. ISAM prescribes that this functional model is populated throughout all the system's control levels. In other words, each control level may contain a set of intelligent control nodes, with the same generic functional structure. The nodes at the various control levels are connected in a task-oriented command hierarchy. The nodes may share knowledge horizontally and coordinate task execution.

For example, in a manufacturing environment, there could be a manufacturing cell that is realized by a cell control node. This node may supervise a set of subordinate control nodes: { machining workstation, inspection workstation, assembly workstation }.

4 Multiple resolutions

Task commands from nodes to their subordinates define a command and control hierarchy. Task commands to the highest level node define the system's goals.

ISAM prescribes a unique task decomposition process that transitions the system goal into subgoals and subtasks at each of the control levels until the control signals at the lowest control level are generated to drive the electrical/mechanical devices. In other words, the task decomposition process generates a hierarchy of subgoals with a gradual decrease of scope and increase of details. Developers design task and command structures corresponding to the goals and subgoals.

The manufacturing system cell controller may perform a make_product task. This task is decomposed into a subtask set { machine_part, inspect_part, assemble_parts } for the subordinating workstation level. The inspect_part task may be further decomposed into a subtask set of { load_part, locate_part, inspect_features }. This process continues through the generation of motor signals to control the inspection probe.

ISAM also contains a perception process that performs an integration of low-level sensed features to form high-level perceived features. Image pixels are integrated to form linear features. The latter, themselves, are integrated to form surface features. The process continues to form object features and features of even higher levels of abstraction. Discrete events are integrated to proper levels of abstraction. For example, the completion of the following three tasks, machine_part, inspect_part, and assemble_parts are integrated to mean that the make_product task has been completed at the next higher level.

5 Generic Shell: system realization

We have described the functional and logical structure of ISAM. Now we describe an implementation tool, Generic Shell, to facilitate application system development.

Generic Shell is consistent with many object-oriented concepts. In Generic Shell, we apply many layers of inheritance, abstraction, and generalization-refinement relationships to transition the functional architecture to physical, executing systems. The ISAM model is generic and applies to most manufacturing processes. We define a set of more refined classes to be applied to a subproblem domain, the discrete part manufacturing processes. A further refined set of classes applies to the manufacturing inspection processes. Finally, a control workstation for a particular inspection machine is implemented.

Generic Shell, thus, contains multiple layers of code templates, currently written in the C++ language. When implementing specific systems, we embed specific software components or algorithms in the templates.

We specify the following types of templates:

Processing templates: As a default, we cluster the node functions shown in Figure 1 into a processing configuration that includes the following processes: planning (PL), sensory processing and world modeling (SPWM), and a set of execution processes (EX) corresponding to the node's subordinate nodes. Note that the terms "world modeling" and "knowledge base update" are regarded as equivalent in this paper. See reference [12] for a detailed description of these templates.

Interface templates: We have been applying a common message structure to facilitate the interfaces that the ISAM reference model prescribes [13]. The essential interface between PL and EX deals with the schedule. The message structure that PL sends to EX contains a schedule. The message directs EX to execute the schedule. The message may also contain pointers to the knowledge requirements for the execution.

The essential interface between EX and the subordinate PL is individual command messages corresponding to the schedule steps that the EX is executing.

The essential interface between the PL and SPWM and EX and SPWM is a query and

response mechanism, to obtain specific information to support planning.

Knowledge templates: ISAM is a task-oriented architecture. As such, the knowledge that is required in each node is also based on the task that the node is to perform.

Common forms of task knowledge include state tables and process plans. We have been developing a task frame model that can serve as an integrated information structure supporting behavior generation. The information may include task goal and its constraints, planning and execution states, plan description or references to planning algorithms or plan databases, transition conditions, errors, knowledge requirements, resource requirements, priority, and performance metrics [14].

Testing and Performance Measurement: Generic Shell includes the facility to allow system testing and performance measurement. Generic Shell builds in data logging facilities for users to examine the execution history. All the interface data buffers are accessible to facilitate performance analyses. Generic Shell applies a graphic tool [13] to view system execution status and to plot state variables, in real-time. Generic Shell stresses using simulation and animation to visualize system performance. The control node shells are also used to implement simulation nodes to interact with controller nodes.

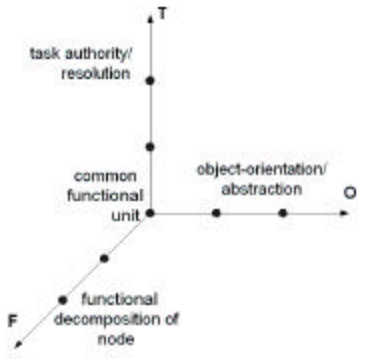

Figure 2: Multiple and integrated perspectives

During system development, Generic Shell is used with a scenario-based task decomposition methodology to develop ISAM-based systems [12, 15].

6 Integrated Framework

The previous sections describe the ISAM framework. The results can be shown in an integrated view in Figure 2. Section 2 corresponds to axis F. Sections 3 and 4 correspond to axis T. Section 5 corresponds to axis O. An application control system hierarchy would be:

- parallel to axis T to possess multiple control levels,
- perpendicular to axis O with certain distances away from the origin because of the inheritance and abstraction process, and
- extending along the F axis due to the fact that the Generic Shell models detailed control node functions.

Previous documents provided some earlier concepts for this framework [15, 16]. The following case study is our latest illustration.

7 Inspection Workstation Implementation

The NIST inspection testbed is used to study many architectural issues, some of these are beyond the scope of this paper [17]. We will focus on several control nodes that are implemented in Generic Shell.

Operators use a manufacturing inspection system to measure a manufactured part to find out if the part is made to the specification. As shown in Figure 3, we have implemented a workstation-level controller to coordinate the execution of an inspection operation.

The Generic Shell based control nodes are implemented in a planner (PL) and an executor (EX) shell. The world modeling and sensory processing aspect is included in

the PL processors for computational efficiency.

The workstation controller commands two subordinates, a Coordinate Measuring Machine (CMM) controller that interprets the inspection programs and a fixturing controller that coordinates the placement of the part.

The CMM controller PL interprets the inspection programs written in a standard language and converts them into the command language native to the control system. The executor sends the traverse, measure, computer vision processing, and tool-change command messages to its multiple subordinates. These subordinates are not shown in Figure 3. See reference [18] for details. The CMM controller receives inspection data for the SPWM function to compute for the inspection results.

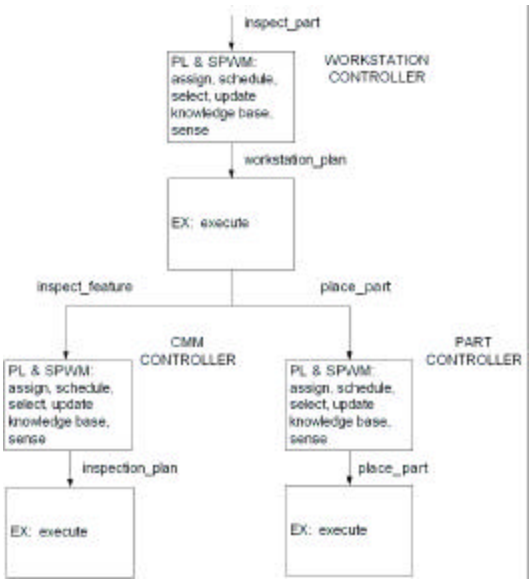

Figure 3: Inspection Control Hierarchy (partial)

We have conducted a series of tests on this Generic Shell based implementation. The control system has successfully inspected a part and generated inspection reports. See reference [12] for details.

8 Summary and Future Work

We have presented a brief overview of a reference model architecture, ISAM, proposed as a common architecture for large-scale, complex real-time control systems. We also described a Generic Shell approach to facilitate the system development effort based on the ISAM architecture. The Generic Shell approach provides a unified system execution and interfacing pattern. Generic Shell intends to be a source-code level open system approach. It allows different component or subsystem solutions to integrate to meet system goals. It facilitates a modularized structure for complex systems. It intends to provide a rigorous formalism and yet does not induce the cost of using commercial compute-aided system design tools. Researchers have begun applying the Generic Shell to other projects to demonstrate its applicability.

For the next phase, we plan to explore applying industrial standards, such as the UML representation, to model the Generic Shell templates.

References:

1 Garlan, D. and Perry, D., "Introduction to the Special Issue on Software Architecture," Guest Editorial, IEEE Transaction on Software Engineering, April 1995.

2 Ribeiro-Justo, G.R. and Cunha, P.R.F., "An Architectural Application Framework for Evolving Distributed Systems," Journal of Systems Architecture, Volume 45, No. 15, September 1999, Elsevier Science B.V.

3 SEMATECH Technology Transfer 93061697J-ENG, Computer Integrated Manufacturing (CIM) Framework Specification Version 2.0, SEMATECH, Inc., January 1998.

4 http://imtr.ornl.gov/Default.htm

5 http://www.arcweb.com/omac/

6 Kosanke, K., et. al., “CIMOSA: enterprise engineering and integration,” Computers In Industry, Volume 40, Elsevier Science, 1999.

7 http://cimosa.cnt.pl/Docs/Primer/primer5.htm

8 Antsaklis, P.J. and Passino, K.M., eds., *An Introduction to Intelligent and Autonomous Control*, Kluwer Academic Publishers, Norwell, MA, 1993

9 Senehi, M. K.; Kramer, Thomas R.; A Framework for Control Architectures; International Journal of Computer Integrated Manufacturing; Vol. 11, No. 4; 1998; pp. 347-363.

10 J. S. Albus and A. Meystel, “A Reference Model Architecture for Design and Implementation of Intelligent Control in Large and Complex Systems,” the International Journal of Intelligent Control and Systems, 1996.

11 Y. Maximov, and A. Meystel, “Optimum design of multiresolutional hierarchical control systems,” Proceedings of IEEE International symposium on intelligent control, pp 514-520, Glasgow, UK, 1992.

12 Huang, Hui-Min, et al., “Toward a Common Architecture for Large-scale Real-time Control Systems,” Accepted in the 16th IFIP World Computer Congress, Conference on Software: Theory and Practice (ICS2000), August 2000.

13 http://www.isd.cme.nist.gov/projects/rcs_lib/

14 Huang, H. and Senehi, K., “Task Frame Conceptual Design,” NIST Internal Document, 1996.

15 Huang, H., “An Architecture and a Methodology for Intelligent Control, ”*IEEE Expert*, Volume 11, Number 2, April 1996, pp. 46-55, IEEE Computer Society, Washington, D. C.

16 Huang, H., et. al., “An Open Architecture Based Framework for Automation and Intelligent System Control,” Invited Paper for the IEEE Industrial Automation and Control Conference’95, Taipei, Taiwan, May 1995.

17 Messina, E., Horst, J., Kramer, T., Huang, H., Tsai, T., and Amatucci, E., "A Knowledge-Based Inspection Workstation," Proceedings of the 1999 IEEE International Conference on Information, Intelligence, and Systems, Bethesda, MD, November, 1999.

18 Horst, J., “Architecture, Design Methodology, and Component-Based Tools for a Real-Time Inspection System”, The 3rd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2000), Newport Beach, CA, March 15-17, 2000.