Open System Architecture for Real-time Control Using a UML Based Approach

Hui-Min Huang, Elena Messina, Harry Scott, James Albus,
Frederick Proctor, and William Shackleford
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
{hui-min.huang, elena.messina, harry.scott, james.albus, frederick.proctor, william.shackleford}@nist.gov

Abstract

We describe a generic architecture that is applicable to the engineering of many real-time control problems. We further describe how UML is used to apply the architecture to the problems.

1 Introduction

Industry desires short lead time for production to enable companies to compete in the global economy [1, 2]. Applying software architectures yields the benefits of interoperability, portability, efficient system integration, etc. [3, 4, 5, 6, 7], which contribute to the industrial objective.

Albus [8] proposed a generic, open, reference model architecture called Real-time Control System, (RCS). RCS and its variants, including Intelligent System Architecture for Manufacturing (ISAM), have been applied to various large-scale engineering problems [9, 10, 11, 12, 13, 14].

ISAM/RCS attempts to model the fundamental behaviors of complex systems. The ISAM/RCS model is a hierarchical control structure that is composed of multiple, organized control nodes. High control levels cover system behaviors that are longer in temporal span and wider in spatial span. Low control levels handle tasks that are more detailed, with short time and spatial spans. For example, a shop floor control may handle a production order that takes months to make certain products with multiple workstations. On the other hand, the lowest level may involve the servo control of motors for the machines. We will describe the architecture, together with a method for applying the architecture to engineering problems.

Unified Modeling Language (UML) has rapidly emerged as an industrial standard for software engineering. There are many characteristics in ISAM/RCS that make UML a feasible modeling and representation language for the architecture. Since our effort started before UML's emergence as an industry standard, we will describe how our approach transitions from being C++ language based to being UML-based.

We use a manufacturing inspection scenario to illustrate the key concepts. Manufacturing shop operators use inspection systems to measure dimensions of manufactured parts. The systems compare the measurements of the parts with the parts' models to determine whether the parts are made to the specifications.

2 Reference Model

ISAM/RCS provides that nodes in a control system all have the same structure despite their different behaviors. A control node generates behavior through a planning and execution process. The behavior generation process is supported by physical or logical sensory processing functions and real-time knowledge processing functions. These functional aspects of the architecture are shown in Figure 1. This generic-plus-specific structure repeats throughout the entire system.

This generic structure gives rise to our software implementation method called generic shell.

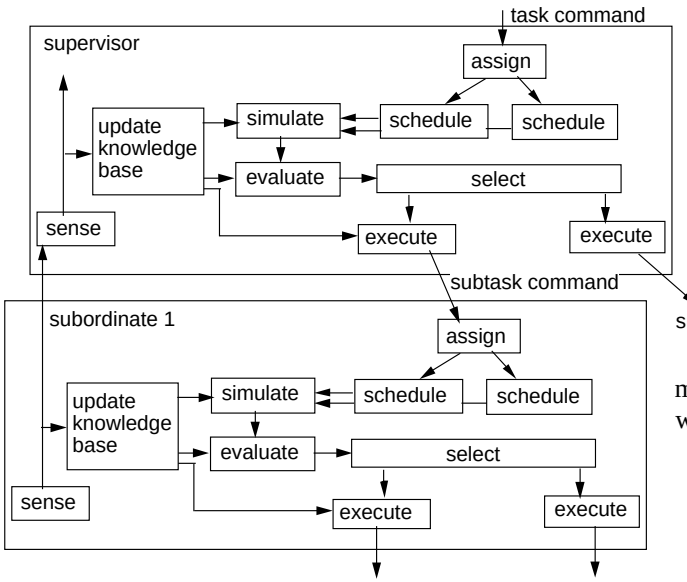

Figure 1: ISAM Control Node Functions

Generic shell provides generic computing models for ISAM-based hierarchical control systems, including interfacing and communication. Developers apply the generic building blocks to all the control nodes and interfaces to build a skeleton for their control systems. The developers then embed the application-specific behavioral or processing algorithms into the skeleton and fill in the inter-process interface templates. This is consistent with the generalization/inheritance and data abstraction concepts in object-oriented paradigms.

The number of layers and nodes that form the organization of a particular application is dictated by the application's task and behavior requirements. This highlights ISAM/RCS's task orientation.

In Figure 2, we show the integrated architectural conceptual framework for ISAM/RCS.

At the center of the figure is the generic control node model. The node is populated up along the T-axis (Task) to form a hierarchical system; each level having different task responsibility. The node is functionally decomposed along the F-axis (Function) depending on each node's application complexity. Various domain or application specific functionality may be added along the O-axis (Object) to form subclasses and, eventually, applications themselves.

Along the T-axis, the ISAM task decomposition process decomposes system goals into a hierarchy of subgoals with gradual decrease in scope and increase in details. These subgoals are assigned as responsibilities for the control nodes at the various control levels. The behaviors for the nodes at the bottom of the hierarchy would be the actuator signals to manipulate the hardware. In a manufacturing process, a high level task INSPECT_PART may be decomposed to a low level task set including { GOTO, PROBE_TO, SET_PARAMETER}. GOTO itself may then be decomposed into a set of waypoints for execution.

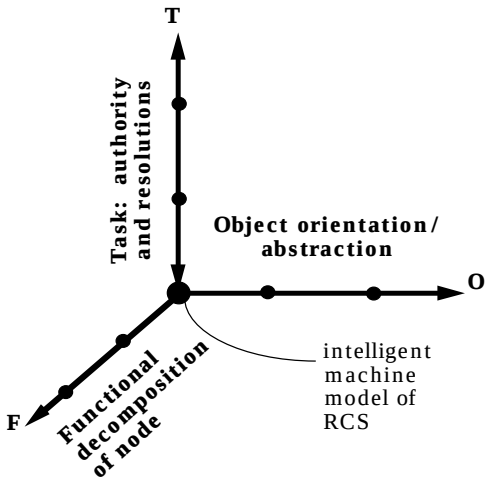

Figure 2: Conceptual Framework for ISAM/RCS Generic Shell

A perception process performs successive integrations that transform low-level features into high-level 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 abstractions.

Along the F-Axis, as indicated in Figure 1, all the control nodes perform the same types of functions. However, different applications may dictate nodes invoking different solution paradigms or algorithms, such as AI, fuzzy logic, or traditional, analytical control algorithms.

From a system realization perspective, the O-axis captures the concepts of from abstract to concrete, from generic to specific, from skeleton to expansion, and from reference model to applications.

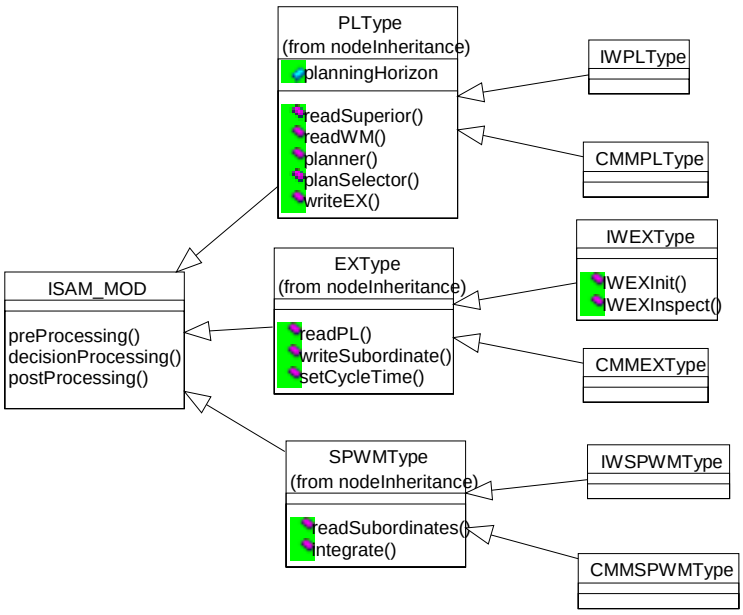

Figure 3: UML representation for generic shell process templates

3 Generic shell

Generic shell is an implementation method for ISAM/RCS. Generic shell can be realized in various forms, from C++ code templates, to commercial tools, to UML classes. Our descriptions contain work from both the C++ and UML aspects. The two versions of generic shell share the same software engineering concepts.

Figure 3 depicts a UML representation for generic shell process templates.

The first layer, the most generic, is the base class. The second layer comprises templates that model the ISAM functions. The third layer models nodes for the inspection system. These three layers follow the O-axis. The fact that the third layer classes would populate the control hierarchy is consistent with the T-axis.

We have combined the functional model as described in Figure 1 into three processing templates, planning (PL), executor (EX), and sensory processing and world modeling (SPWM). This is consistent with the F-axis.

3.1 The base class

The base class, ISAM_MOD, specifies a generic, cyclic pre-decision-post process model. The preprocess obtains and computes all the information to support the decision process. The post process outputs and stores the outcome.

We derive the following ISAM shells.

3.2 The PL shell

This shell typically covers the assign, schedule, and select boxes in Figure 1. Figure 3 illustrates how we plan to use the methods to implement those boxes. This diagram is developed with the Rational Rose® tool[1] from the Rational Software Corporation.

During preprocess, the input channels include commands from the superior, execution status from the subordinates, and queries and responses for data. In ISAM/RCS, the planning function plans ahead for ten steps, typically. The system's knowledge at any control level tends to be more uncertain and the environment tends to change significantly to make looking further ahead useful.

During execution, the generated plans are re-evaluated and replanning is performed every cycle. Generic shell employs a plan selector to manage the plan buffer. The plan selector is also responsible for coordinating with the executors that execute the plans.

Figure 4 illustrates how a PL shell may select different plans depending on an input command. Figure 5 illustrates a particular plan, which is to initialize the control node. This set of examples has been implemented and executed on an NT platform. These two diagrams are developed with the ARTiSAN Real-time Studio® tool from the ARTiSAN Software Tools, Inc.

[1] Certain commercial products or company names are identified in this paper to describe our study adequately. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the products or names identified are necessarily the best available for the purpose.

3.3 The EX shell

The EX shell interfaces with PL by receiving command schedules and other instructions. It sequences, schedules, executes schedule steps, and reports status to PL. When errors occur, EX either performs emergency routines or reports the error and suspends execution.

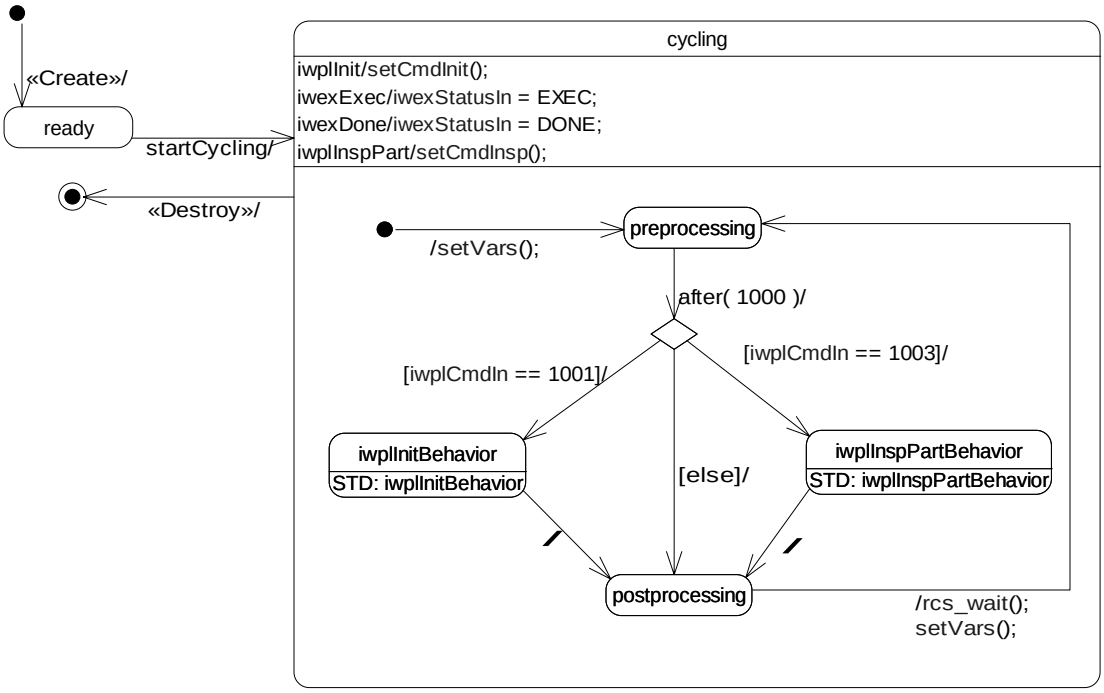

Figure 4: A PL Shell Example

3.4 The SPWM shell

The SPWM shell typically covers the sense, update knowledge base, simulate, and evaluate boxes in Figure 1.

3.5 Knowledge Structure and Interfacing

Generic shell contains a task-oriented method and, as such, the knowledge that is required in each system is based on the task that the system is to perform. The knowledge can be organized in terms of node tasks. The tasks require many attributes, including 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.

ISAM/RCS maintains a system perception process in real-time. The perception process supports task generation and execution. The perception process acquires and assimilates system knowledge through the sensory processing function. In generic shell, task commands flow down the hierarchy (the T-axis). In Figure 3, a PL reads task command messages from its superior node and writes to EX to execute the tasks. This process repeats top-down.

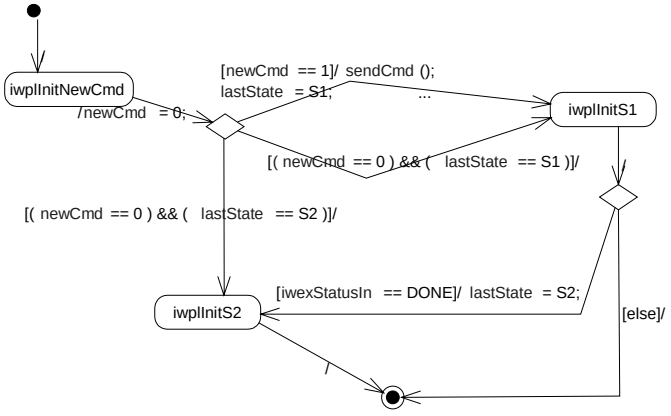

Figure 5: A Behavior in PL

On the other hand, the sensed information flows upwards. In Figure 3, SPWM reads the information from below and integrates and perceives it for a feature with the corresponding level of resolution.

For example, the knowledge that "the part is ready for inspection" integrates multiple pieces of low level information:

- the part has been placed on the inspection table,
- its location has been determined, and
- the knowledge for inspecting the part, i.e., its inspection plans, has been retrieved. This further implies that the geometrical knowledge has been obtained.

4 Summary and Future Work

We described our ongoing investigation of applying UML to a software development process. Our current results have indicated that UML is able to support an efficient process for modularizing and sharing software under the ISAM/RCS generic real-time control system architecture. UML may allow us to reuse legacy code and external software libraries. It may support a scenario driven development process that is consistent with the ISAM/RCS methodology. UML also provides a standard representation that is critical to open systems and to facilitate understandability.

References:

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

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

3 Garlan, D., "Software Architecture: a Roadmap," The Future of Software Engineering, Finkelstein, A., Ed., ACM, New York, New York, 2000.

4 Kosanke, et al., "CIMOSA: enterprise engineering and integration," Computers In Industry, Volume 40, Elsevier Science, 1999.

5 Tarr, P. , et al., "N Degrees of Separation: Multi-Dimensional Separation of Concerns. "Proceedings of the International Conference on Software Engineering (ICSE'99), May, 1999.

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

7 Maximov, Y. and Meystel A., "Optimum design of multiresolutional hierarchical control systems," Proceedings of IEEE International symposiumm on intelligent control, pg 514-520, Glasgow, UK, 1992

8 Albus J. S. and Meystel A., "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.

9 Balakirsky, S.B., Lacaze, A., World Modeling and Behavior Generation for Autonomous Ground Vehicles, Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, April 24-28, 2000.

10 Proctor, F.M., "Practical Open Architecture Controllers for Manufacturing Applications," Open Architecture Control Systems: Summary of Global Activity, ITIA Series, Vol. 2, pp. 103-114, 1997.

11 Moore, M., et al., "Complex Control System Design and Implementation Using the NIST-RCS Software Library," IEEE Control Systems, Volume 19 Number 6, December 1999.

12 Albus, J.S., McCain, H.G., Lumia, R., NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM), NISTTN 1235, National Institute of Standards and Technology, Gaithersburg, MD, May 1989.

13 Barbera A.J., et al., "A 3-D Control And Simulation Analysis Tool For The United States Postal Service", ASME 1992 Japan-U.S.A. Symposium on flexible Automation Conference, San Francisco, CA, July 13-15, 1992.

14 Huang, H., Michaloski, J., Tarnoff, N., and Nashman, M., "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.