To appear in the April 1996 Issue of IEEE Expert.

AN ARCHITECTURE AND A METHODOLOGY FOR INTELLIGENT CONTROL

Hui-Min Huang
Mechanical Engineer
National Institute of Standards and Technology
Gaithersburg, MD 20899
email: huang@cme.nist.gov

Abstract

We outline a multiple dimensional reference model architecture and a methodology for representing and developing intelligent systems. The reference model architecture features multiple dimensions enabling modeling the multiple aspects of complex systems. The canonical form within this framework facilitates an open and scalable system architecture. The well-defined structures facilitate efficient knowledge engineering processes. We describe a submarine automation model performing real-time control to illustrate the application of this reference model architecture.

Keywords: automation, functional decomposition, hierarchical systems, intelligent control, methodology, multiple dimensional architecture, object oriented, task analysis.

1. Introduction

Large scale intelligent control systems pose unique challenges in computer software and hardware technologies for researchers. These systems often conduct critical missions. They commonly require the capability of real-time access to knowledge bases to meet the millisecond level control cycle requirements. Researchers have begun to address some aspects of this complex problem domain.

There have been considerable contributions in the area of hierarchical control architectures. Saridis introduced a three-level hierarchy in [1]. Albus, in [2], described a reference model architecture called the Real-time Control System (RCS). RCS includes six basic levels of authority with the controller nodes all represented via an intelligent machine model. RCS forms the basis of this paper. Acar and Özgüner, in [3], provided an alternative architecture that organizes the system into a multi-resolutional hierarchy by identifying its components and examining the physical relationships among them. Antsaklis and Passino, in a chapter of [4], described a hierarchical control architecture within which a hybrid approach was proposed to model systems with a high degree of autonomy. Successive delegation of duties from the higher to lower levels is among the important characteristics of the hierarchy. Meystel, in another chapter of [4], described a nested hierarchical control theory that included the concept of treating design and control as a design-control continuum.

In the direction of implementing intelligent control systems, a research focus has been the areas of software technologies and computer-aided software engineering environments. Sweet et al., in [5], identified the key software technologies for the Aerospace Industries Association (AIA). Simmons, in [6], described a Task Control Architecture (TCA). Scalability might be a limitation of TCA as it is not intended to model multiple cooperating agents. Object oriented paradigms are becoming popular for handling the representation problems of software systems. However, they are not suitable for all problems. Schneider, et al., in [7], developed a flexible object oriented real-time software

implementation tool called ControlShell[1]. However, this tool does not intend to address the issue of architecture and it seems as if a reference model architecture can complement the capability of ControlShell.

Intelligent system control has been the research focus of the Intelligent Systems Division (ISD) of the National Institute of Standards and Technology (NIST). NIST ISD proposes that a comprehensive approach toward this intelligent control system problem should cover all of the following critical issues:

* A scalable and open architecture.
* A rich and representative reference model.
* A distributed and efficient structure for organizing system knowledge.
* A rigorous knowledge engineering process and modeling paradigm.
* A comprehensive computer-aided rapid development and deployment environment.
* Real-time control and operator interaction capability.

The approach that the NIST ISD has been using is the Real-time Control System (RCS) reference model architecture [2]. Researchers have been applying RCS to various large scale intelligent control systems, including [8, 9, 10], since two decades ago. The ultimate goal for the NIST ISD is for RCS to evolve into a unified solution paradigm to the problem domain of intelligent system control.

In particular, this paper attempts to describe RCS as a multiple dimensional reference model architecture. We attempt to integrate different notions of hierarchy into one unified framework for use with large scale intelligent system control.

2. Multiple Dimensional Reference Model Architecture

The term hierarchy can mean different things to different people. In an object oriented paradigm, a hierarchy can mean a tree describing class derivation. In a functional decomposition paradigm, a hierarchy can mean layers of subfunctions representing a system. RCS contains, and is not limited to, both of these two aspects. However, these two aspects have not yet been explicitly described in an integrated fashion in previous RCS literature. Task decomposition and level of authority have been among the important characteristics in the RCS applications [8, 9, 10]. NIST ISD has also been developing generic software templates and libraries that future RCS applications can inherit. Efforts are required to integrate all these aspects together to form a unified paradigm. This is a major issue that we intend to address in this paper. We propose to use an integrated coordinated system, comprising the three

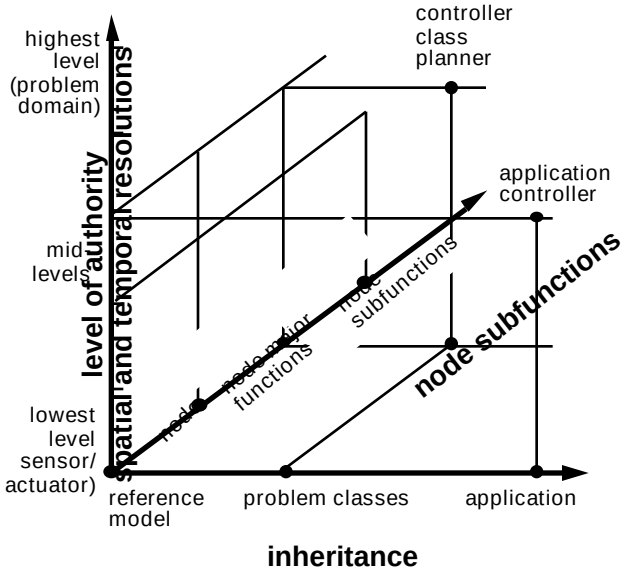


Figure 1: A Multiple Dimensional Reference Model Architecture for Intelligent Systems

[1] References to company or product names are for identification only and do not imply NIST endorsement.

paradigms of: level of authority, functional decomposition, and inheritance, to form a multiple dimensional reference model architecture. The origin of the coordinate system contains a generic controller node, which serves as the building block of RCS and is described by an intelligent machine model [2]. The structure is shown in Figure 1. Sections 2.1 through 2.3 describe the three axes. Section 2.4 shows a resulting view and describes how an implementation architecture is identified within the reference model architecture. This system characterizes RCS. This multiple-disciplinary and integrated paradigm facilitates rich and representative system models. Applying simplistic modeling paradigms may fail for large systems. Booch, in [11], describes two perspectives of a system: algorithmic decomposition and object-oriented decomposition, with the latter being the driving perspective. These perspectives correspond to the functional decomposition and the inheritance perspectives of the reference model architecture that this paper describes. The most significant difference in our concept is that the level of authority perspective drives the system design while referencing the generic reference model.

2.1 The functional decomposition dimension--an intelligent machine model

An intelligent system must be capable of making complex decisions, based on its assessment of the current environment, and taking actions to accomplish its goals. The intelligent machine model proposed by Albus in [2] contains the required functions that allow for such capabilities. We attempt to briefly describe the intelligent machine model in a functional decomposition notion and to describe how the model performs the intelligent decision making and situation assessment activities.

In RCS, the term functional decomposition means that a node is decomposed into finer and finer functions. We use a coordinate axis, as shown in Figure 2, to describe this effect. A generic controller node resides at the origin of the axis. This node permits interaction from an operator. The node is functionally described by the behavior generation (BG), sensory processing (SP), world modeling (WM), and value judgment (VJ) functions, as shown at the second unit of the axis. This is the basic model of the RCS intelligent machine model. Extending farther along the axis means further decompositions of the node functions and subfunctions. The following describes this model in detail:

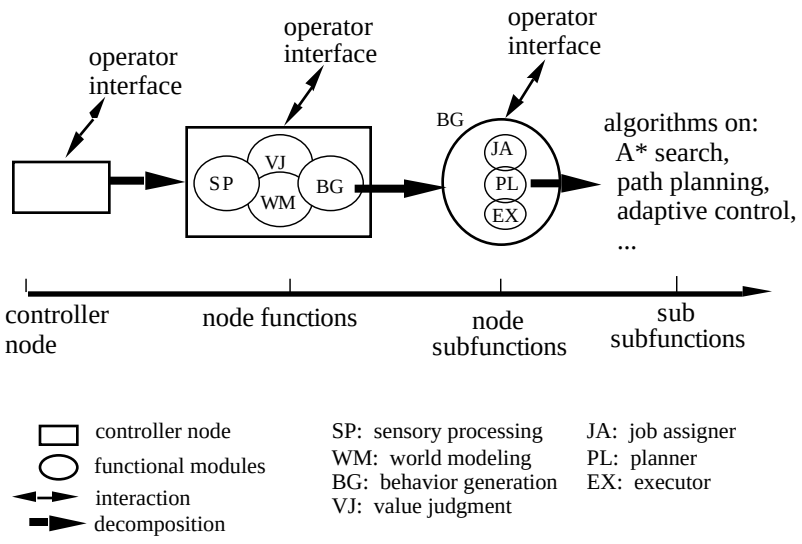


Figure 2: Functional Decomposition of a Controller Node

2.1.1 The behavior generation (BG) function

BG is responsible for planning and executing the tasks that a node receives from its superior node at the higher hierarchical level. The resulting output is sent to the node's subordinate nodes at the lower level as their input commands.

The third unit along the axis shows that BG contains the following subfunctions:

The job assignor (JA). JA spatially decomposes the node's input commands based on the subordinates that the node has. The results will be used for the temporal planning purposes.

The planner (PL). Planning typically requires forming, evaluating among, and selecting from alternative hypothetical sequences of subtasks ordered in the temporal sense.

The executor (EX). EX executes the selected plans. The execution is done by comparing the commanded values, provided through PL, and the observed values, provided through WM, of the state variables and computing the output commands for the next level nodes.

The fourth unit along the axis shows another layer of functional decomposition. In Figure 2, we illustrate some algorithms possibly used in a PL. For example, particular path planners may be used as a part of the submarine navigation planner. An adaptive control algorithm may be used for the position control of a flexible manipulator.

2.1.2 The sensory processing (SP) function

SP contains the following major functions:

Data acquisition. This function samples, filters, and validates sensory data.

Data integration. This function integrates sensory information over space and time. The data may be provided by multiple subordinates.

Data assimilation. This function may recognize patterns and detect events.

The subfunctions typically contain individual algorithms in these categories.

2.1.3 The world modeling (WM) function

WM contains a set of functions both maintaining the associated knowledge base (KB) and servicing the other RCS major functions. The major WM functions include:

Knowledge base management. This function uses the data provided by SP to update the KB in real-time. This function also keeps the knowledge content consistent across the system.

State estimation and prediction. The results may be used by PL to plan the next move or by SP to expect certain sensory input.

Query management. PL may query WM as "what if I make this move" during the planning stage. EX may query WM as "what is the current value of X" during execution. This function is responsible for retrieving the information from the knowledge base and generating the responses.

The subfunctions of WM, at the third unit along the functional decomposition axis, may include particular algorithms such as a least-squares based estimator.

The KB supports the BG decision making functions. The KB is distributed within all the controller nodes of a system. The information stored in the KB consists of two parts, models and data. The models include the static and dynamic models of the system and of the environmental objects of concern, such as a hydrodynamic model of a submarine or a geometric representation of a mechanical component. The following information is a typical set of the data knowledge maintained by each node and used to support real-time task execution:

- The plan information. RCS typically uses state transition diagrams to model plans. The state transition based decision making process requires the information including the set of plans that a node is capable of performing, the name of the plan that is being executed, and the current state of execution. Quintero described this, in detail, in [12].

- The node status. Typical status values are: Reset, Executing, Done, Waiting, Error, and Emergency stop. This information allows the control system to respond properly. The system developer may specify an execution paradigm such that, unless in emergency situations, a node must be at the Done state before the node can accept the next input command.

- The error code. Typical execution errors include: a mismatch of a command or status between a sender and a receiver, correspondent nodes not responding, time out, etc. The occurrence of certain error might point to certain recovery procedures.

- The performance indices: Timing performance is typically the most critical index. A node can maintain the following timing information: maximal and minimal cyclic execution time and execution time trends in various forms. During the real-time decision making processes, the performance indices might indicate that a controller node is being overloaded and proper attention is needed.

- The data to be shared by other nodes, such as object position, fuel level, etc. This data must be maintained to allow consistent execution throughout the entire system.

This set of information also provides a snapshot of the system under execution for the operator.

The integration of individual node knowledge bases and the information common to the system constitutes the control system knowledge base. In this sense, the system knowledge base is organized by referencing the structure of the control hierarchy.

2.1.4 The value judgment (VJ) module

VJ determines the costs, risks, and benefits of the hypothesized plans and actions that a planner may generate. VJ includes the following major functions, although we feel that further investigation is required to fully elaborate VJ:

Criteria computation and update. Value criteria must be computed from the given tasks. Safety, time, or precision may be of the highest value in different situations. Albus also described the emotional aspect of the values in [2]. This function may need to update or recompute the criteria as situations change.

Value judgment. This function computes the costs and benefits of the hypothesized plans based on the established criteria. The results are sent to PL for the plan selection purposes.

2.1.5 Interactions among the node functions and operator interface (OI)

The four functions, BG, SP, WM, and VJ form a closed control loop for a node. In addition, a node is subject to operator interaction (OI). The following is a summary of the specific interactions among these functions:

- The PL in BG may query WM for predicted results of the hypothesized plans that the PL generates. The predictions may be sent to VJ for cost, benefit, or risk analysis. The PL uses the results to select a plan.

- The EX in BG may query WM for the current values of certain state variables and compare them against the desired values that the PL has planned for. EX then generates output commands for the subordinates to correct the errors.

- The SP may require predictions and estimations of certain state variables or any a priori knowledge, from the WM, in order to validate sensory data or to detect events or features. The SP then provides the WM with the current results for the WM to update the knowledge base.

- An operator may send commands to the BG, request WM data for display, change the gains in the control or filtering algorithms, respond to a BG request for recovering errors, or override the automatic control during emergency conditions.

In [2], Albus provides further details of the node functions and their interactions.

2.2 The level of authority dimension--hierarchical levels

RCS is a hierarchical architecture. Controller nodes are distributed across all the predefined levels. The nodes are also authoritatively connected . We establish a coordinate axis in the multiple dimensional reference model architecture to describe these RCS levels. We also characterize this dimension by a set of tenets.

2.2.1 Levels of authority

In RCS, the following levels are predefined as the guidelines for partitioning a hierarchical system:

Level 6 -- Problem Domain Level, also called Facility or Mission level. This is the highest level. The controller receives overall commands, from an operator, for the entire control system. The BG function of this node decomposes these commands and outputs the results to the responsible next level controllers.

Level 5 -- Group Level. Multiple physical entities may exist in a hierarchical system and they must be coordinated at this level. In a manufacturing environment, a workstation may coordinate multiple pieces of equipment. These workstation controllers, then, belong to the group level. In a Defense environment, a group level controller may coordinate a fleet of naval vessels and/or a squadron of air planes.

Level 4 -- Equipment, or Task Level. A node at this level typically models a major physical entity, for example, a submarine. Tasks received by the controllers at this level concern how each piece of equipment is expected to perform to accomplish a system goal.

Level 3 -- Elementary Move (E-move) Level. The e-move level is the kinematic control level. Any task is decomposed into a series of subtasks that are free of kinematic limits, singularities, and obstacles. Sensor data submitted from the primitive level (see the next paragraph) may be combined to produce surface features, feature distance and relative orientation, etc.

Level 2 -- Primitive (Prim) Level. The primitive level is the dynamic control level. The kinematically sound tasks are computed for subtasks that are dynamically smooth. The SP

function integrates and fuses data gathered from individual sensors and produces linear features for objects.

Level 1 -- Actuator Level. The controller nodes at this level interact with the environment. The BG generates electrical, hydraulic, or mechanical commands to activate the actuators. The SP function for this level is to receive signals from each individual sensor and to process them.

2.2.2 Tenets of this dimension

Flexibility of the number of nodes at the levels: Each level can have none or multiple nodes except for the highest level where there is one node. Some problems may require only on-off types of control and may not require a dynamic (prim) level. On the other hand, some problems may need multiple sublevels within a predefined level. This may happen when the tasks are complex enough to warrant another level of decomposition between a pair of predefined parent and child levels. It may also happen when the physical environment contains multiple layers of natural boundaries between a pair of predefined parent and child levels. For example, a manufacturing facility may have multiple production lines. A production line may have multiple workstations. A workstation may have multiple pieces of equipment. The group level may contain two sub levels: production and workstation.

Canonical form: The generic controller node structure and inter-node interface repeats and extends in the context of the intelligent machine model to a level sufficiently high to describe a system. This facilitates the scalability of a hierarchical control system. A unified execution behavior exhibits across all the controller nodes at all the levels. In implementation, RCS utilizes commonly available computer platforms, including components such as the C or C++ languages, the PC or VME busses, and the UNIX or DOS operating systems. These features, together, facilitate the openness of an RCS control system.

Execution of system goals via task decomposition: The behavior generation (BG) function of the node(s) at each level receives the commands from the level above, decomposes them, and outputs the results to the responsible next level controllers. In other words, controllers at a particular level coordinate the execution of the next lower level controllers.

Resolution, temporal span, and spatial span: High levels deal with tasks and data that have less detail but longer time and wider spatial span. Lower levels deal with tasks and data that have more detail but shorter time and narrower spatial span.

2.3 The inheritance dimension--reference model to application

The concept depicted by Figure 3 is that the desired functionality of the models to the left of the axis is inherited by the models to the right. The fact that RCS has been developed as a reference model architecture implies that the properties of the generic node, shown at the

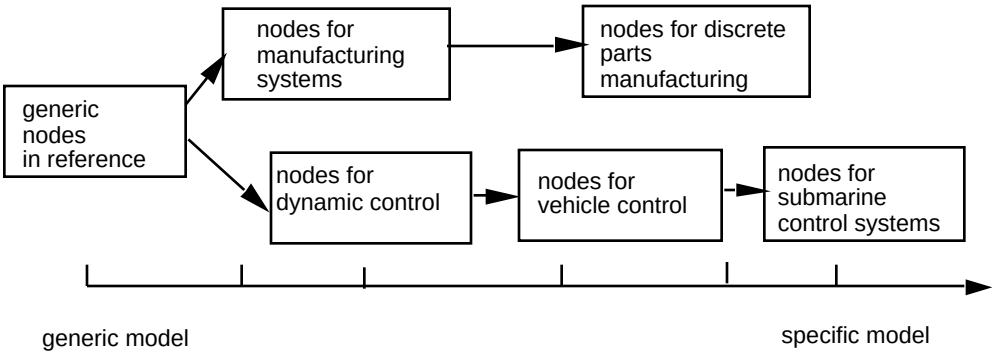


Figure 3: The Inheritance Dimension

origin of the axis in the figure, may be inherited by any class of problems adopting the architecture. The dynamic control systems and the manufacturing control systems are two examples of the problem classes. Any vehicle control RCS may inherit the properties developed for the dynamic control system RCS. This inheritance relationship can extend to many layers. Each layer may contribute commonly useful software library sets. The concept described here is consistent with that in the object oriented paradigms. This feature makes the architectural implementation process efficient and makes the RCS development environment rich.

In the case of developing a software template for a controller node, the base structure, which resides at the origin of the inheritance dimension, may include the following generic functions:

- Reading input data. BG, WM, SP, and VJ may all require this function.
- Evaluating the command status. This is a part of JA.
- Writing the output data to the proper subordinates. This is a part of EX.

This base structure may be common for all the nodes at all the RCS authoritative levels. At the task level, a derived structure for the vehicle control problem class may inherit all these functions (called methods in some object oriented paradigms) and may add vehicle position explicitly as a part of the WM data to be read in and written out. A subsequently derived controller node for a submarine may yet add another piece of data, water pressure, as a part of the SP data to be read in, written out, and processed. The essence is that the common properties, data or functions, can easily be identified using the RCS construct described along the other two axes of the multiple dimensional framework. The identified properties are then inherited along the inheritance dimension. This makes the RCS architecture efficient.

Unlike the other two dimensions, NIST ISD had not systematically investigated this inheritance dimension until recent years when the object oriented paradigms became popular. NIST ISD, just as many other research organizations, is exploring the feasibility and efficiency of migrating from traditional paradigms to the object oriented paradigm.

Therefore, our current accomplishment along this inheritance dimension of RCS is limited and this subject would be a major focus of the NIST ISD's future research.

2.4 The resulting framework

We obtain the resulting multiple dimensional reference model architecture after integrating the three dimensions, namely, level of authority, node subfunctions, and inheritance, together, as illustrated in Figure 4. The traditional RCS levels are identified along the level of authority axis. The generic major and subfunctions of a node, including BG, WM, SP, and VJ, are identified along the node subfunction dimension. A generic prim level planner may include a control algorithm. A vehicle control problem class may have common features, such as path planning, in the planner. These common features may further be inherited by subsequent problem classes or individual applications along the inheritance axis, such as the submarine application illustrated in Figure 4.

We further illustrate the submarine automation control architecture in Figure 5. This architecture has multiple levels. The nodes, shown as boxes on the figure, are distributed along the levels (horizontal dotted lines) and are authoritatively connected (solid lines). The highest level node of this hierarchy coordinates two subordinates. An RCS architectural implementation can be viewed as a hierarchy tree rooted on a notch on the inheritance axis and growing in parallel with the authority axis.

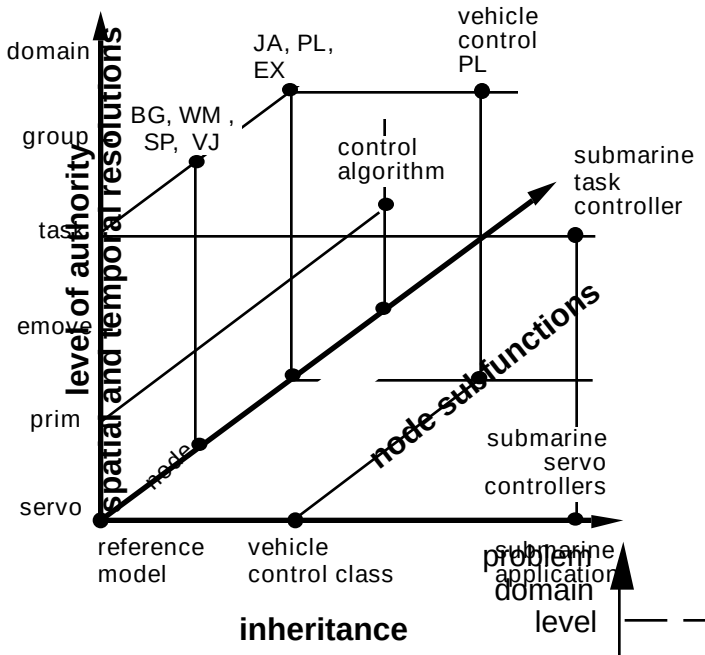
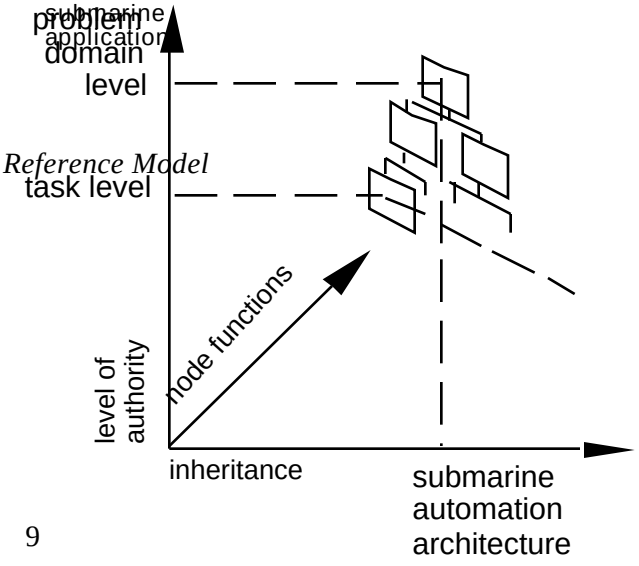


Figure 4: The RCS Multiple Dimensional Reference Model Architecture

In the three-level hierarchical model that Saridis proposed, in [1], the execution level is responsible for executing control functions. The coordination level is responsible for short range decision making and learning. The organization level is responsible for long range planning, decision making, and the management and handling of the information. The hierarchical model that Antsaklis and Passino described in [4] referred to a similar structure. These levels are consistent with the levels along the authority axis in this multiple dimensional framework. In general, the organization level corresponds to the group level and up in RCS. The coordination level



corresponds to the task level through the prim or emove levels in RCS. The execution level corresponds to the servo level or up to the prim level in RCS.

3. RCS Methodology: A Task Oriented Knowledge Engineering Process

3.1 knowledge evolution process

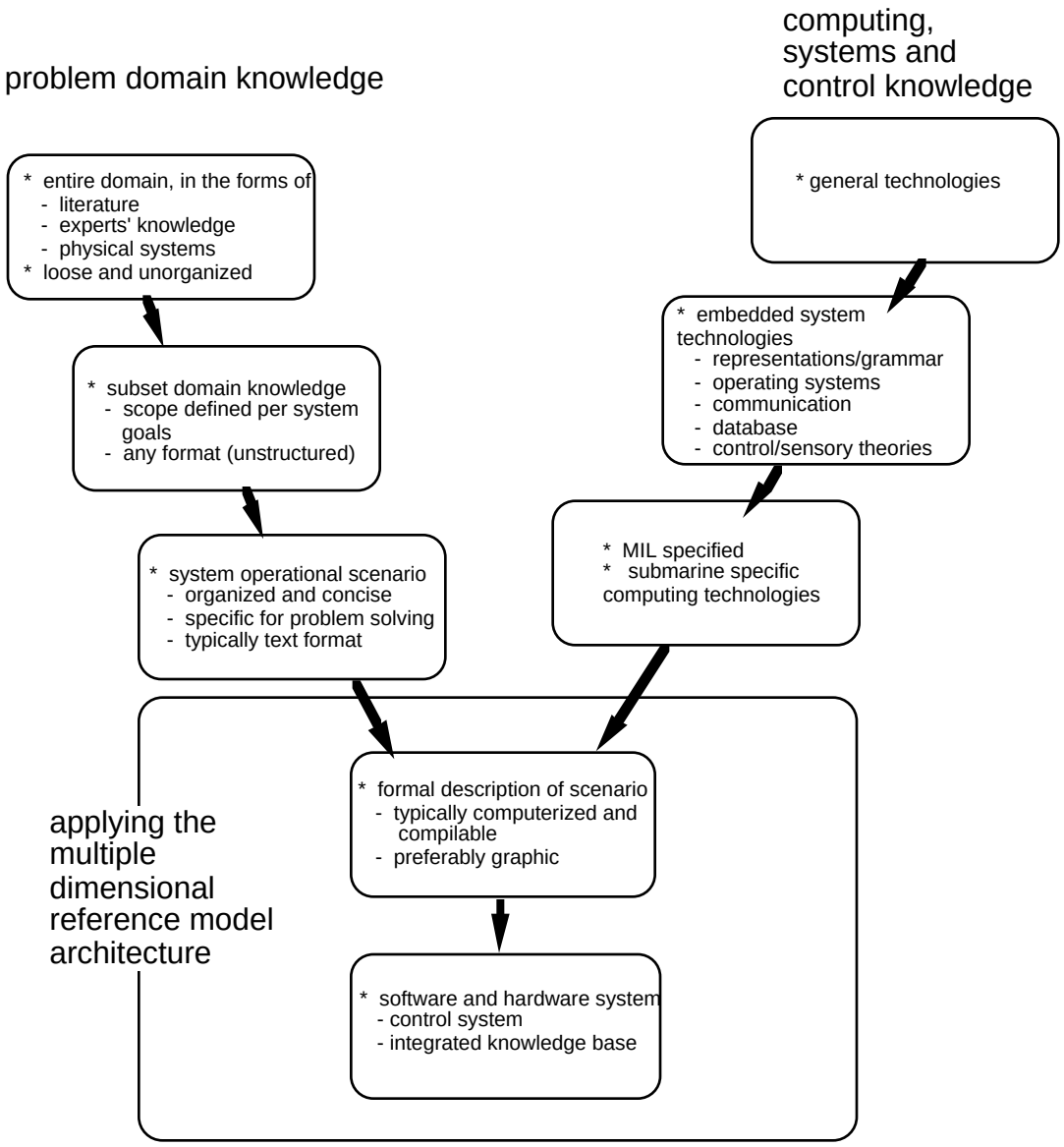


Figure 6: A Knowledge Evolution Process

Architectural implementation involves understanding, sorting, assimilating, and integrating the domain knowledge and the systems, computing, and control knowledge using a systematic approach. RCS prescribes a task oriented knowledge engineering process, which is highlighted in Figure 6 as a knowledge refinement process. Note that, the RCS multiple dimensional reference model architecture does not become involved until the middle stages of the process. Knowledge for a problem domain and knowledge in the

computing, systems, and control fields are, to a large extent, conceptually independent in their raw forms, shown as the two separate branches at the upper half of the drawing. In the second box of the left branch, the RCS methodology calls for interactions with domain experts to identify a subset of the knowledge that is within the scope of project requirements. These domain expert interactions may result in clarifications or modifications to the pre-established project requirements. Another step of knowledge refinement process further deduces the domain knowledge to a set of inclusive written operational scenarios. The developers perform a task analysis based on the scenario descriptions. Quintero [12] also describes these aspects in detail.

Meanwhile, at the right-hand side branch of Figure 6, a separate knowledge refinement process occurs. Note, we have not yet fully explored how the task oriented method is applied to this particular area. Appropriate computer platforms (for both development and operations), modeling and implementation languages, operating systems, CASE tools, military (MIL) and other standards or specifications, etc., must be investigated and appropriately incorporated in the RCS architectural implementation environment.

The two knowledge areas merge after the operational scenarios are obtained, as shown in Figure 6. The developers perform task analysis to further formalize the system knowledge as described in the scenarios. We have developed a logical software structure, described in the next section, facilitating the task analysis and system design of RCS applications.

3.2 A proposed analysis and implementation structure

This RCS methodology proposes a generic and comprehensive logical structure to facilitate the analysis and implementation of the multiple dimensional reference model architecture. This structure identifies and integrates the following five hierarchies (Figure 7): control, simulation, animation, operator interface (OI) for control, and OI for simulation. The reasons for requiring these hierarchies are:

- The control hierarchy performs real-time system control to accomplish missions. This is the hierarchy described in Figure 5. The lowest level nodes send electrical, hydraulic, or mechanical control signals to physical actuators. However, in the laboratory development or simulation situations, a simulator is required.

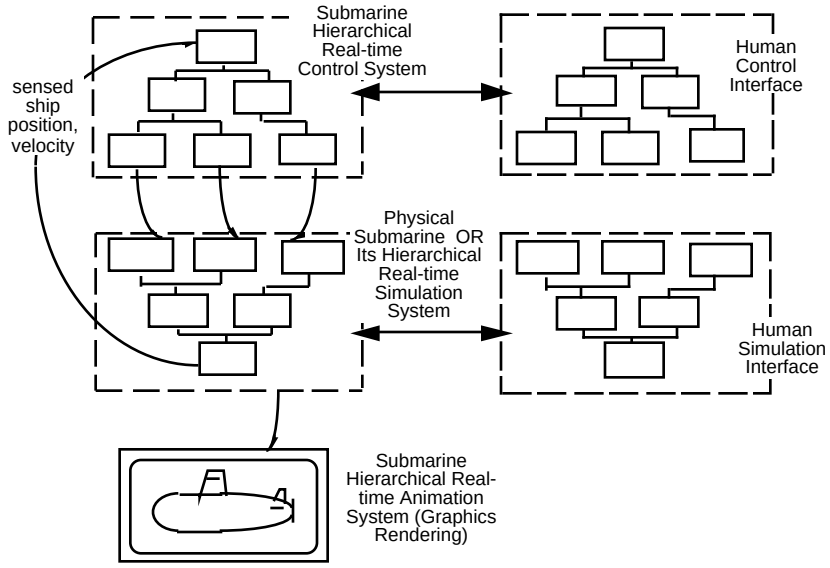


Figure 7: An Analysis and Implementation Software Structure

- A hierarchically structured simulator provides an advantage that the simulated physical system behavior also presents multiple levels of abstraction. The lowest level simulator nodes, the actuator simulators, receive control signals and compute for the mechanical movements of each actuator. The higher level nodes integrate the simulated actuator movements and compute the ship level dynamics. The resulting ship state variable values are fed back to the higher levels of the control hierarchy via the simulated sensors. The dynamics of the relevant environmental objects is also simulated.

- The RCS methodology proposes a conceptual distinction between simulation and animation. Simulation involves the kinematics and dynamics of the objects. Animation performs graphic rendering of the simulation results, at, or close to, real-time. The animator may also be hierarchically structured to form a close correspondence to the simulator.

- The RCS architecture specifies that, conceptually, all the nodes permit operator interaction. This is why, in Figure 7, the OI is described as one-to-one correspondent to the control and simulation hierarchies. In implementation, however, it is likely that the one-to-one relationship is not followed. There may be a keyboard input device responsible for the interaction with multiple control nodes.

- We also propose a logical distinction between the control and the simulation OI. Control OI may allow switching among multiple control modes: manual, autonomous, or semi-autonomous. It may allow a control override from an operator in emergency situations. Simulation OI involves setting up parameters or triggering external events for the simulator software to generate desired simulation behavior. This Simulation OI capability allows us to test the performance of the control systems.

3.3 Task analysis and implementation

In performing task analysis, the researchers essentially identify and order the control system tasks, simulation events, and control nodes on the appropriate parts of the logical structure described in Figure 7. The tasks and events correspond to the verbs of the scenario descriptions. The control nodes correspond to, but are not equal to, the agents performing the operations as stated in the scenario descriptions. The RCS methodology then uses a formal modeling representation, namely, finite state machines, to model the control system behaviors. What need to be modeled, graphically, are the sentences that the

verbs are associated with, in the scenario descriptions. These state diagrams are also called RCS plans. The control system operates by selecting and executing proper plans based on the system goals and the environmental situations.

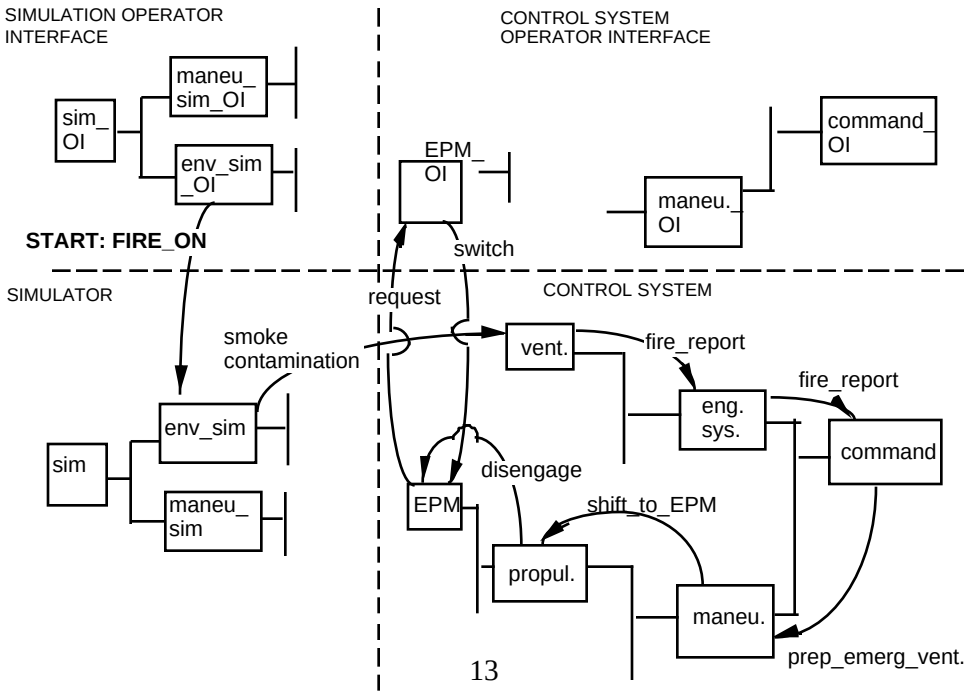
The state diagram development process also reveals the data and the data processors that are required to support the state transition based decision making process. Finally, a C or C++ language based controller node software template [13] is used to code the software. These activities are shown in the bottom two boxes in Figure 6. The templates are a part of the NIST ISD effort to explore the inheritance dimension of the multiple dimensional reference model architecture.

4. The Submarine Automation Application

We illustrate the RCS methodology using an automation model of a submarine maneuvering system. In this project, the NIST ISD researchers collaborated with a retired submarine commander. Based on the pre-established goals of the simulated submarine, he described to us, in detail and in every aspect, how a submarine would operate to achieve the specified goals. The following are some of the methods used to maximize the relevant knowledge output from the domain expert. ISD researchers did not impose any structural constraints during his description. The commander was encouraged to use the submarine terminology. The main role of the ISD researchers at this stage of interaction was to be goal driven, to be intelligent listeners, and to ask stimulating questions sparsely to catalyze his stories. The following section provides an illustrative submarine demonstration scenario. Note that, this scenario serves as the basis for the illustration of the submarine automation knowledge revolution process, given in the sections that follow.

4.1 An illustrative scenario for the submarine automation model

A submarine is conducting a submerged transit of the open ocean at its standard speed (15 knots or nautical miles/hour, equivalent to 7.7 meters/second) and at a keel depth of 200 meters. A watchstander (a submarine term, meaning a crew member who is assigned to a designated onboard location to perform the pre-specified duties) reports that there is a lube oil fire in the lower level Engine Room. The Officer of the Deck (OOD) directs the Ballast Control Panel (BCP) operator to pass the word on the general announcing system. The OOD completes the following actions for coming to periscope depth: Clearing baffles, Checking for sonar contacts and close contacts, Slowing and changing depth, and Raising



the periscope.

The damage control party fights the fire in the engine room. On indication of decreasing main lube oil pressure the OOD orders "All stop, shift propulsion to the EPM (emergency propulsion motor)." The shaft rotation is stopped and the clutch is used to disengage the shaft from the turbines and the EPM circuit breaker is closed. The Engineering Officer of the Watch (EOOW) reports to the OOD that he is prepared to answer bells on the EPM. The OOD orders "Ahead two thirds" which maintains enough speed for depth and steering control.

The damage control party reports that the fire is out. The BCP selects the ventilation lineup and sets it to emergency ventilate the engine room using the diesel engine. When the lineup is proper, the OOD directs "Commence snorkeling."

4.2 Task analysis

To begin the submarine automation task analysis, the researchers establish the four quadrants corresponding to the four hierarchies described in Figure 7. This is demonstrated in Figure 8.

Arrows show the sequence of the tasks, nodes, and events that we have identified. Note that the goal is to develop an intelligent control system to replace the manual operations described in the scenario. The first significant event, as described in the scenario, is the report of a lube oil fire. Accordingly, at the upper left corner of Figure 8, a fire_on event is identified. During simulation, this event is designed to be injected through a simulation operator interaction node and to be sent to the env_sim simulator node. This node computes the changes in the air constituents (see smoke_contamination in the figure) which are detected by the simulated sensors installed in the vent controller. This illustrates how developers realize that particular sensors and controller are required. Within the control hierarchy, that is located at the lower right quadrant of the figure, the fire report is sent up the hierarchy to the highest level controller which has the responsibility of coordinating the engineering system (labeled as eng. sys.) and maneuvering (labeled as maneu.) controllers.

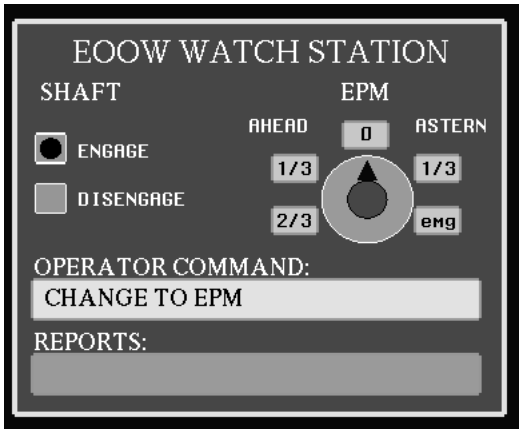


Figure 9: A Graphic Operator Interface Panel

Per the scenario, one of the commands that Maneuver will receive is to prepare for emergency ventilation (prep_emerg_vent). Maneuver decomposes this command and requests the propulsion controller to switch to the EPM, or the emergency propulsion motor, control. In our control system configuration, this particular switching operation is to be done manually. Therefore, such a request is sent to the controller operator interface hierarchy, shown in the upper right quadrant of Figure 8. A graphic display of this operator interface module is illustrated in Figure 9. The operator reads the message, disengages the main shaft, and reports a done status. The operator will receive another command later describing the required EPM speed, which he will dial the knob, as shown in the figure, accordingly.

Many iterations of these event, node, and task identification activities are required to finalize the multiple hierarchies. The resulting control hierarchy is partially shown in Figure 10, with some of the developed controllers beyond the scope of this paper omitted. Note that, this hierarchy further illustrates the architectural framework, shown in Figure 5, and the methodology, shown in Figure 7. The command controller, at the highest level, coordinates two subordinates: maneuver and engineering systems. Maneuvering involves the coordination of the propulsion and depth controllers. The submarine depth is further controlled by a dive/rise controller which coordinates the sail and stern planes. The propulsion controller coordinates the turbine controller (which provides propulsion control in regular conditions), the DC motor controller (which provides the control in emergency conditions), and the clutch allowing switching between these two mechanisms. The parts of the engineering systems of our concern include a diesel engine and the ventilation system.

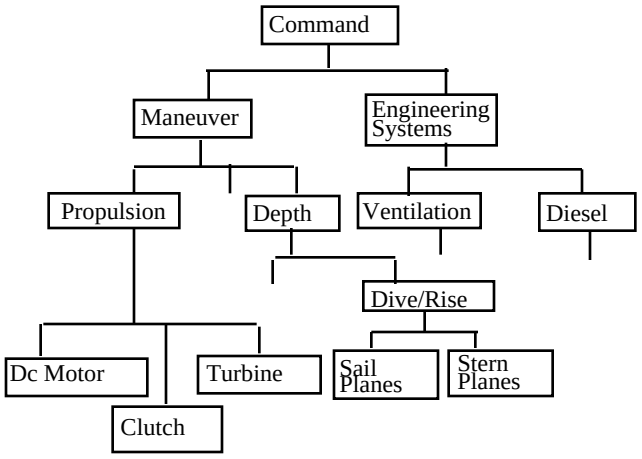


Figure 10: A Submarine Architectural Implementation

The tasks and events are identified before the plans are described for each controller node. The illustration for these activities is omitted in this paper since state transition based modeling is fairly well known.

4.3 Real-time control and simulation

This section illustrates some real-time depth control and simulation activities for the submarine automation model. In Figure 11, a mission command given to the commander includes the depth requirements. They must be converted to the electrical signals for the sail and stern planes. Intermediate levels are needed to provide smooth transitions between these two extremes. The intermediate levels facilitate human understanding, computation efficiency, and control stability. The command controller passes the depth requirements of the mission, through a series of way points, down the hierarchy to the Maneuver controller. Maneuver computes the required ship depths accordingly and passes them down for the Depth controller. Depth computes a series of bubble angles required for controlling the ship to the specified depths. The Dive/Rise controller computes required plane angle maneuvers, for the Sail and Stern Plane controllers, to achieve the required bubble angles. The plane controllers generate required electrical signals for the control valves to move the planes to the commanded angles.

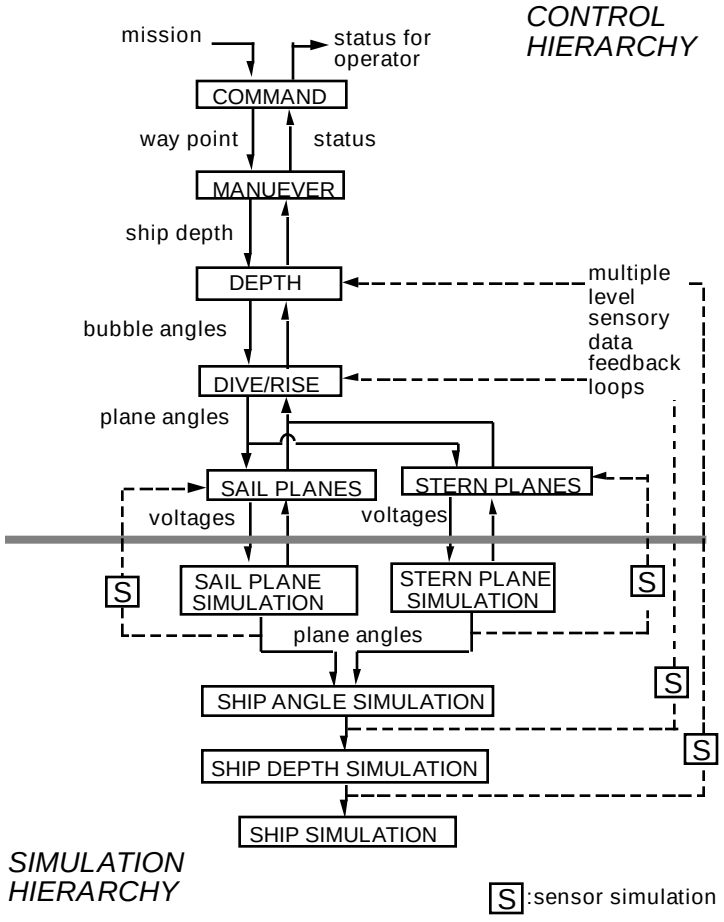


Figure 11: Nested Depth Control and Simulation in Submarine Automation

The submarine depth simulator is developed as a reverse of the control hierarchy, as seen in the lower portion of Figure 11. The only input that the simulator receives from the controllers is the commanded electrical signals. The hydrodynamic model for the submarine is decomposed and distributed in the simulator hierarchy. At the lowest level (shown at the top of the simulator), the electrical signals are used to compute for the simulated plane angles, which are integrated at the next level to form simulated ship bubble angles. At the next level, the dynamic model uses the ship angles to compute the ship depth. All these intermediate results may be used as sensor input to feed back to the appropriate controllers.

This example shows a rigorous control system behavior as a result of the rigorous system analysis and design paradigm.

5. Conclusion and Future Directions

The multiple dimensions in the architecture facilitate the descriptions of complex systems from multiple perspectives. This provides a framework for an open and scalable system architecture. A knowledge engineering methodology has been described and illustrated. This process describes systematic structures to organize system knowledge and describes a series of smooth transformations to assimilate system knowledge from raw forms to computer executable forms. A submarine automation model is used to illustrate the application of this RCS reference model architecture and its knowledge engineering methodology. The simulation demonstrated the model performing intelligent tasks by using the well-structured operational knowledge.

Researchers at NIST ISD and elsewhere has begun working to automate this process. A Joint Architecture project has been ongoing to describe, in detail, a generic architecture for discrete parts manufacturing facilities. We are currently investigating a specification for a RCS computer-aided control system development (CACSD) tool. The achievement of such a tool would greatly enhance the inheritance aspect of the architecture. Software templates and reusable libraries have been generated to speed up implementations and to further explore the inheritance property of RCS. The ultimate goal is an automated environment facilitating the development of intelligent systems covering from the very early conceptualization stages to the final critical mission real-time operation stages.

Acknowledgments

Dr. James Albus and Dr. Anthony Barbera have been leading the research and application of RCS since they originated it at NIST two decades ago. Captain Robert Lowell and Dr. Ed Carapezza of ARPA have been the sponsors of the submarine project at various stages and provided valuable technical insights to the problem. Keith Young was our principal submarine domain expert. Richard Quintero, Ron Hira, Ross Tabachow, and Will Shackleford of NIST and M. L. Fitzgerald, Nat Frampton, Philip Feldman, and Clyde Findley of the Advanced Technology Corporation have participated in various stages of the submarine project.

References

1 Saridis, G.N., “Foundations of Intelligent Controls,” in *Proceedings of the IEEE Workshop on Intelligent Control 1985*, IEEE Computer Society Press, Washington, D.C., 1985, pp. 23-28.

2 Albus, J.S., “Outline for a Theory of Intelligence,” *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 3, May/June 1991, pp. 473-509.

3 Acar, L. And Ozguner, U., “Design of Knowledge-Rich Hierarchical Controllers for Large Functional Systems,” *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 20, No. 4, July/August 1990, pp.791-803.

4 Antsaklis, P.J. and Passino, K.M., eds., *An Introduction to Intelligent and Autonomous Control*, Kluwer Academic Publishers, Norwell, MA, 1993, pp. 1-26 and pp.129-161.

5 Sweet, W., et al., Recommendations from the AIA/SEI Workshop on Research Advances Required for Real-Time Software Systems in the 1990s, Special Report SEI-89-SR-18, Cargenie-Mellon University Software Engineering Institute, Pittsburgh, PA, September, 1989.

6 Simmons, R., et al., "Autonomous Task Control for Mobile Robots," *in Proceedings of the Fifth International Symposium on Intelligent Control*, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 663-668.

7 Schneider, S. A, et al., “ControlShell: A Real-Time Software Framework,” AIAA Conference on Intelligent Robots in Field, Factory, Service, and Space, American Institute of Aeronautics and Astronautics, Washington, D.C., 1994.

8 Huang, H., Hira, R., and Quintero, R., "A Submarine Maneuvering System Demonstration Based On The NIST Real-Time Control System Reference Model," in *Proceedings of The 8th IEEE International Symposium on Intelligent Control*, Chicago, Illinois, 1993, pp. 376-381.

9 Huang, H., Quintero, R., and Albus, J.S., "A Reference Model, Design Approach, and Development Illustration toward Hierarchical Real-Time System Control for Coal Mining Operations," Book Chapter in *Control and Dynamic Systems, Advances in Theory and Applications*, Volume 46, Academic Press, 1991, pp.173-254.

10 Szabo, S., Scott, H.A., Murphy, K.N., Legowik, S.A., Bostelman, R.V., "High-Level Mobility Controller for a Remotely Operated Unmanned Land Vehicle," *Journal of Intelligent and Robotic Systems*, Vol. 5, No. 199, Kluwer Academic Publishers, Norwell, MA, 1992, pp. 63-77.

11 Booch, G., Object-Oriented Analysis and Design with Applications, 2nd Edition, The Benjamin/Cumming Publishing

12 Quintero, R. and Barbera, A.J., "A Software Template Approach to Building Complex Large-Scale Intelligent Control Systems," in *Proceedings of The 8th IEEE International Symposium on Intelligent Control*, Chicago, Illinois, 1993, pp. 58-63.