1. INTRODUCTION 5

2. THE RCS REFERENCE MODEL 8

2.1 A NODE 8
2.2 MULTIPLE RESOLUTIONS 9
2.3 BEHAVIOR AND BEHAVIOR GENERATION 11
2.4 BEHAVIOR AS A PRINCIPLE OF ABSTRACTION 12
2.5 ORGANIZATION AND COMMAND AUTHORITY 12
2.6 OBJECT-ORIENTED PERSPECTIVES: CONTROL NODES AND MAPPING BEHAVIOR TO NODE HIERARCHY AS A MEANS OF ENCAPSULATION 12
2.7 OPERATOR INTERFACE 13
2.7.1 Basic Principles 13
2.7.2 Operator Levels of Commanding Authority 13

3. RCS METHODOLOGY--THE DEVELOPMENT PROCESS 15

3.1 BEHAVIOR ORIENTATION, OBJECT ORIENTATION, AND DATA/FUNCTION ORIENTATION 15
3.2 THE METHODOLOGY STEPS 15
3.3 BEHAVIOR ANALYSIS, SCENARIO DEVELOPMENT, AND DOMAIN EXPERT INTERACTION 18
3.4 STRUCTURING BEHAVIORS IN HIERARCHY 20
3.5 TEMPLATE BASED SYSTEM IMPLEMENTATION 21
3.5.1 The Functional View, Multiple Granularity, and Current Scope 22
3.5.2 The Base Class Generic Process Template 23
3.5.3 Command and Status Templates 25
3.5.4 Template Inheritance 25
3.5.5 Generic Shell: Using Templates to Build RCS Nodes and Functions 26
3.5.6 Combining Behavioral-oriented, Object-oriented, and Functional Decomposition Paradigms 28
3.5.7 Analysis and Implementation structure 28

4. CASE STUDY I, FINITE STATE MACHINES, SIMULATOR TEMPLATES, AND OPERATOR INTERFACE FOCUSED IMPLEMENTATION 30

4.1 SCENARIOS 31
4.1.1 Maneuvering Mechanisms 31
4.1.2 Traverse through Bering Strait Scenario 33
4.1.3 Open Sea Maneuver and Casualty Responses Scenario 33
4.2 BEHAVIOR ANALYSIS 35
4.3 ILLUSTRATIVE BEHAVIOR DESCRIPTIONS 39
4.3.1 Command Level 39
4.3.2 Helm Control and Illustrative Result 40
4.4 SOFTWARE STRUCTURE 43
4.5 FINITE STATE MACHINE BASED PROCESS TEMPLATE 45
4.6 SIMULATION: TEMPLATE AND IMPLEMENTATION 46
4.6.1 The Structure 47
4.6.2 Sensor 47
4.6.3 Actuators 47
4.6.4 Physical System 48
4.6.5 Environmental 49
4.7 OPERATOR INTERFACE 50
4.7.1 Watchstation Displays: Mapping Control Activities to the Operating Environment to Facilitate Operator Interaction 50
4.7.2 Operator Interface Data Types 54
4.7.3 Levels of Operator Involvement During Operations 54
4.7.4 User Expertise and Hierarchical Situation Perception 56
4.7.5 Supporting Other System Functions 57
4.8 KNOWLEDGE INFERENCE 58
4.9 ANIMATION 59

5. CASE STUDY II, MANUFACTURING DOMAIN, TOWARD IMPLEMENTATION SPECIFICATION AND ON-LINE BEHAVIOR GENERATION 60

5.1 APPROACH........61
5.1.1 Generic Shell for ISAM........61
5.1.2 Application Domain: TestBed and Prototype System Design........62
5.1.3 Leveraging Component and Interface Technologies........64
5.2 METHODOLOGY DEVELOPMENT: GENERIC SHELL, INTERFACE DEFINITIONS, AND TASK EXECUTION........65
5.2.1 An Internal Execution Model of a RCS Based Manufacturing Workstation........65
5.2.2 The Executor Template........68
5.2.3 The Planner Template........69
5.2.4 Interface Definitions........70
5.3 INITIAL APPLICATION SYSTEM: INSPECTION WORKSTATION........71
5.3.1 Behavior Generation........71
5.3.2 Inspection Workstation Control System Implementation........73

6. CASE STUDY III, INTELLIGENT AUTONOMOUS VEHICLES, EXERCISING FULL ARCHITECTURAL CAPABILITY........76

6.1 SCENARIO DEVELOPMENT........77
6.2 CONTROL HIERARCHY AND TASK DECOMPOSITION........81

7. SUMMARY........83

FIGURE 1: AN RCS NODE 8
FIGURE 2: RCS CONTROL LEVELS BASED ON THE PRINCIPLE OF MULTIPLE LEVELS OF RESOLUTIONS 10
FIGURE 3: RCS DEVELOPMENT PROCESS 16
FIGURE 4: MAPPING BEHAVIOR ON HIERARCHY 21
FIGURE 5: FUNCTIONAL LAYOUT OF RCS PROCESS TEMPLATES 22
FIGURE 6: PROCESS TEMPLATES MAY COVER DIFFERENT FUNCTIONAL SCOPES OF RCS NODE 23
FIGURE 7: AN BASE CLASS INDEPENDENT PROCESS GENERIC PROCESS TEMPLATE 24
FIGURE 8: TEMPLATES INHERIT COMMON RCS CHARACTERISTICS 26
FIGURE 9: APPLYING TEMPLATES TO SYSTEM ARCHITECTURAL IMPLEMENTATION 27
FIGURE 10: A SOFTWARE STRUCTURE FOR ANALYSIS AND IMPLEMENTATION 29
FIGURE 11: SUBMARINE HYDRODYNAMIC CONTROL MECHANISMS 31
FIGURE 12: TASK ANALYSIS DIAGRAM 36
FIGURE 13 ENVIRONMENTAL SIMULATION OPERATOR INTERFACE 36
FIGURE 14: SUBMARINE AUTOMATION RCS CONTROL HIERARCHY 37
FIGURE 15: SUBMARINE AUTOMATION TASK TREE FOR CASUALTY SCENARIO 38
FIGURE 16: SUBMARINE AUTOMATION TASK TREE FOR MANEUVERING AND ICEBERG AVOIDANCE SCENARIO 38
FIGURE 17: A MISSION PLAN FOR THE COMMAND CONTROLLER 39
FIGURE 18: THE HELM CONTROL PLAN 41
FIGURE 19: RESULTING PATH CONTROL IN SIMULATED ENVIRONMENT 43
FIGURE 20: COMPREHENSIVE SOFTWARE STRUCTURE 44
FIGURE 21: FINITE STATE MACHINE BASED PROCESS TEMPLATE 45
FIGURE 22: THE SIMULATION HIERARCHY 46
FIGURE 23: ACTUATOR SIMULATOR TEMPLATE 48
FIGURE 24 THE SHIP SYSTEM SIMULATOR PROCESS TEMPLATE 49
FIGURE 25: MAPPING CONTROL ACTIVITIES TO OPERATING ENVIRONMENT TO FACILITATE OPERATOR INTERACTION 50
FIGURE 26 THE OFFICER OF THE DECK WATCHSTATION DISPLAY 51
FIGURE 27: THE ENGINEERING OFFICER OF THE WATCH WATCHSTATION DISPLAY 51
FIGURE 28 THE BALLAST CONTROL PANEL WATCHSTATION DISPLAY 53
FIGURE 29: DIFFERENT DEGREES OF OPERATOR INVOLVEMENT 54
FIGURE 30 REQUEST FOR OPERATOR ACTION 55
FIGURE 31: HIERARCHICAL SITUATION EVALUATION 57
FIGURE 32: KNOWLEDGE PROCESSING 59
FIGURE 33: OVERVIEW DISPLAY FOR ANIMATION AND OPERATOR INTERFACE 60
FIGURE 34: NAMT CIM TESTBED 63
FIGURE 35: RCS BASED MANUFACTURING WORKSTATION EXAMPLE 66
FIGURE 36: TIME-LINE BASED RCS HIERARCHY INTERNAL EXECUTION MODEL 67
FIGURE 37: INTERNAL FUNCTIONAL FLOW OF EXECUTOR TEMPLATE 69
FIGURE 38: INTERNAL FUNCTIONAL FLOW OF PLANNER TEMPLATE 70
FIGURE 39: INSPECTION BEHAVIOR ANALYSIS 72
FIGURE 40: MAPPING BEHAVIOR ON RCS MODULES 73
FIGURE 41: INSPECTION WORKSTATION 74
FIGURE 42: INTERNAL FUNCTIONALITY OF THE IMPLEMENTED CONTROL NODES 75
FIGURE 43: ISAM TESTBED CURRENT HARDWARE CONFIGURATION 76
FIGURE 44: MILITARY UNMANNED VEHICLE CONTROL HIERARCHY 83

INTELLIGENT SYSTEM CONTROL: A UNIFIED APPROACH AND APPLICATIONS

Hui-Min Huang, Harry Scott, Elena Messina, Maris Juberts, Richard Quintero

Intelligent Systems Division

National Institute of Standards and Technology

Gaithersburg, MD 20899

{huang, scott, messina, juberts, quintero}@cme.nist.gov

Abstract

The hierarchical real-time control system (RCS) reference model architecture that is under development at the National Institute of Standards and Technology (NIST) aims at designing and developing intelligent control for large and complex systems. RCS is based on several general and fundamental principles of complex systems. RCS is a reference model architecture, supporting systems or system components that are developed using Artificial Intelligence (AI) and expert systems techniques, neural nets, fuzzy logic, reactive behavior, or other innovative solutions. RCS provides a mechanism to support knowledge engineering process, an important aspect of expert systems. The RCS methodology, the process of applying RCS to control system development, is described. This process unifies several engineering principles including: multiple spatial and temporal resolutions, behavior-oriented, object-oriented, and functional decomposition. The methodology combines a knowledge-based system-design procedure and a template-based implementation approach. The authors describe three control systems, either completed or still under development, that progressively describe the advances on the implementation approach and demonstrate the richness of the RCS architecture.

Keywords: architecture, automation, behavioral-oriented, engineering methodology, hierarchical systems, intelligent control, object-oriented, software

1. INTRODUCTION

Software architecture and its associated engineering methodology are among the predominant factors that determine the success and the performance of large, complex, intelligent, real-time control systems. These systems may contain many subsystems that execute in a distributed environment. The subsystems may execute independently, yet communicate with the rest of the system in a defined and integrated manner so that the entire system pursues its overall goals. The subsystems may be implemented with different planning and control strategies and yet communicate through standard interfaces that are consistent with the rest of the system. The control system may be reconfigured, in that subsystems may be taken off-line for maintenance or integrated on-line for particular jobs or missions. In addition, information is becoming a critical piece in the current and next generation system control. In the manufacturing environment, part models, process plans, production statistics are just a few of the many pieces of information that need to be computerized, organized, and interfaced to allow easy and timely access so that system goals can be achieved and desirable performance can be delivered. All of these requirements call for an architecture that is scalable and open, incorporating a standard interfacing method [1].

It is desirable to have an architecture that allows developers to employ innovative solutions that may be based on AI techniques such as expert systems, neural nets, fuzzy logic, or reactive behavior for the individual subsystems. The architecture should also allow a hybrid construct [2] that combines discrete-event components such as finite state automata [3] or Petri Nets and continuous-time based components such as adaptive and learning control [4]. To address these issues systematically, it is desirable to employ a particular type of architecture, reference model architecture, that provides generic system static and dynamic models but yet integrates individual component solutions.

Researchers have been addressing certain aspects of these system problems, including flexible manufacturing systems (FMS) [5], hybrid systems [2], intelligent control [6, 7], and open system architecture [8, 9]. Lima and Saridis described a Learning Stochastic Automata based intelligent control system [10]. Antsaklis and Passino, in a chapter of [6], described a hierarchical control architecture that emphasizes successive delegation of duties from the higher to lower levels. Meystel, in another chapter of [6], described a nested hierarchical control theory that included the concept of treating design and control as a design-control continuum. Acar and Özgüner, in [11], provided an architecture that organizes the system into a multi-resolutional hierarchy by identifying its components and examining the physical relationships among them.

The High Performance Computing and Communications (HPCC) Program was formally established following passage of the High Performance Computing Act of 1991 by Congress. That program has been organized more recently into the Computing, Information, and Communications (CIC) R&D (Research and Development) program. The Systems Integration for Manufacturing Applications (SIMA) program [12] is an activity under one of the major components of the CIC. Since 1994, the SIMA program has partially funded work on the development of a standard reference model architecture for the control of complex, intelligent systems. This effort builds on earlier related work in several projects at the National Institute of Standards and Technology (NIST) dating back to the early 1980s and leverages current efforts in other programs.

In particular, the recent NIST efforts have included developing, besides the Real-time Control System (RCS) architecture described in this document, the Manufacturing Systems Integration (MSI) architecture [13], and the Quality in Automation (QIA) architecture [14]. Each of these was developed with specific attention given to different areas of application in the manufacturing domain. For example, MSI focused on manufacturing activities above the shop floor. Application of QIA concepts focused on problems of in-process and process-intermittent inspection on machine tools.

The NIST Intelligent Systems Division (ISD), under the leadership of Dr. James Albus, has been developing, evolving, and applying a reference model architecture called the NIST Real-time Control System (RCS) [15] for the last two decades. RCS is the ISD’s proposed solution to the aforementioned issues and requirements. RCS is a hierarchical control structure with each level assigned specific responsibilities. Controllers are employed at each level capable of performing designed behaviors to fulfill the level's responsibilities. The authors will describe the methodology and show how to apply RCS to control system design. There have been earlier reports describing various states of the architecture and its applications [16, 17, 18, 19, 20, 21]. The authors describe some advances to the architecture and the methodology.

A key to the development methodology is a consistent software engineering paradigm to facilitate the realization of RCS applications. Since RCS emphasizes controlling engineering systems to perform physical work, RCS prescribes a behavior or task oriented system design and analysis [21]. Object-oriented (OO) paradigms [22, 23, 24, 25] are emerging as a major methodology for system development. Although there are skeptics, it is generally believed that these object-oriented paradigms provide several significant advantages over some traditional methods, such as structured design and analysis and information engineering in that OO paradigms focus on the depiction of the real world. This makes the OO paradigms consistent with RCS. The authors will address the object aspects of RCS.

Knowledge engineering is an important issue in the field of expert systems. Knowledge engineering may include the processes of acquisition, perception, organization, management, and inference to support decision making. The RCS reference model provides a mechanism to support all these processes. The RCS methodology focuses on a knowledge based system development procedure. During operations, the RCS based systems exercise all these processes to perform desired behaviors.

The authors will describe three case study control systems with two objectives in mind. First, the descriptions progressively demonstrate the richness of the RCS architecture, from a generic process template based system to real-time rich sensing and planning systems. Second, the case studies describe the evolution and the road map of the methodology, from finite state machine models to process and interface definitions and to a project plan for fully exercising the architecture and methodology. Case study I is a submarine automation demonstration system project, which is one of the first test cases for the methodology. In this project, most the behaviors are modeled as state diagrams at the system development stage. This project provides a base execution model for RCS applications.

Case study II applies RCS to the manufacturing domain. An instantiation of RCS, called the Intelligent Systems Architecture for Manufacturing (ISAM) [26, 27], is applied to the overall SIMA project. A NIST testbed called the National Advanced Manufacturing Testbed (NAMT) [28] is used for implementing and testing the prototype systems. In this case study, more RCS capabilities are planned to be implemented, including real-time planning and rich sensory interaction. The ultimate resulting system would exercise the full architectural capability. The project also calls for specifying processes and interfaces, based on the reference model, to facilitate architectural implementation and integration. This specification process will involve industrial and other government agencies' efforts and results. The results will facilitate component technology integration in the architectural framework.

Case study III applies RCS to the intelligent automous vehicles domain, covering from the Army next generation unmanned vehicle (XUV) to the U.S. Department of Transportation Crash Avoidance program. 4-D/RCS, an instantiation of RCS [29], has been created for these purposes. The implementation effort will leverage the ISAM project results. Full RCS capability is anticipated in the resulting control systems.

2. THE RCS REFERENCE MODEL

A close study of complex systems, including human reasoning processes and behaviors, manufacturing systems, and military command structures, reveals some common principles of intelligent processing and control. RCS is based on a generalization of these principles.

2.1 A Node

The basic operating unit in RCS is a node, or control node, as shown in Figure 1. The model contains a behavior generation (BG) function that makes decisions based on the received task commands and on the current state of the world. BG is supported by world modeling (WM), value judging (VJ), and sensory processing (SP) functions [1, 19].

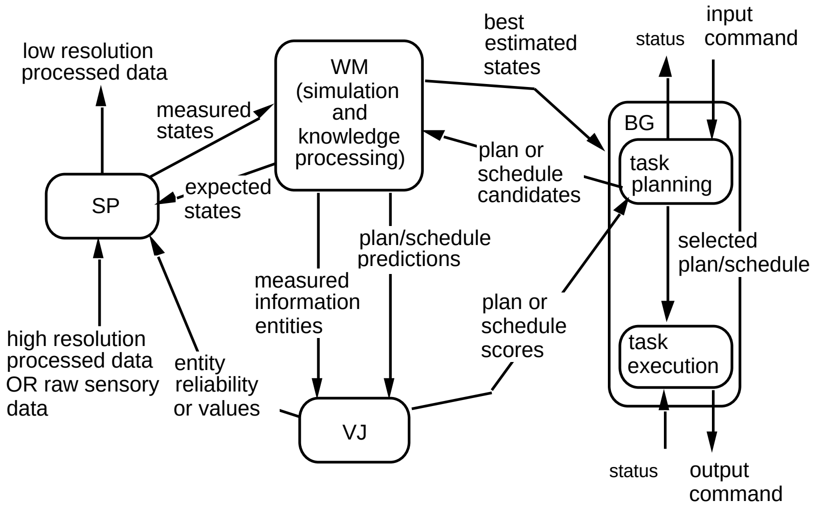

low resolution processed data
measured states
WM (simulation and knowledge processing)
best estimated states
status
input command
BG
task planning
plan or schedule candidates
SP
expected states
plan/schedule predictions
selected plan/schedule
measured information entities
plan or schedule scores
task execution
high resolution processed data OR raw sensory data
entity reliability or values
VJ
status
output command

Figure 1: An RCS Node

BG employs a task planning function that generates a set of possible plans or schedules. WM simulates these candidates and generates the predicted results. VJ uses a set of cost functions to judge the simulation results and provides the values of these schedule candidates for the BG planning function to determine a final schedule for execution. Note that, a schedule is the timing or ordering specifications for a plan [29].

SP processes sensory data and generates measured states for the system. Particular data sets may require different processing algorithms or different gain values for the selected algorithms. The selection process may involve WM and VJ, as the left side of Figure 1 shows. Figure 1 also shows that SP processes data from lower levels at higher resolutions--with more detail but with narrower scope--and output to higher levels data at lower resolutions--with less detail but larger scope.

2.2 Multiple resolutions

RCS is a hierarchical architecture that focuses on modeling systems through spatial and temporal decompositions. Higher levels operate in a state space with larger spatial and temporal span but coarser resolution. Rules that guide the hierarchical decomposition with respect to resolution of space and time are based on control theory and biological evidence. In his "Outline for a Theory of Intelligence [30]," Albus proposed a principle that states,

"In a hierarchically structured, goal-driven, sensory interactive, intelligent control system architecture:

1. control bandwidth decreases about an order of magnitude at each higher level,
2. perceptual resolution of spatial and temporal patterns decreases about an order of magnitude at each higher level,
3. goals expand in scope and planning horizons expand in space and time about an order of magnitude at each higher level, and
4. models of the world and memories of events decrease in resolution and expand in spatial and temporal range by about an order of magnitude at each higher level."

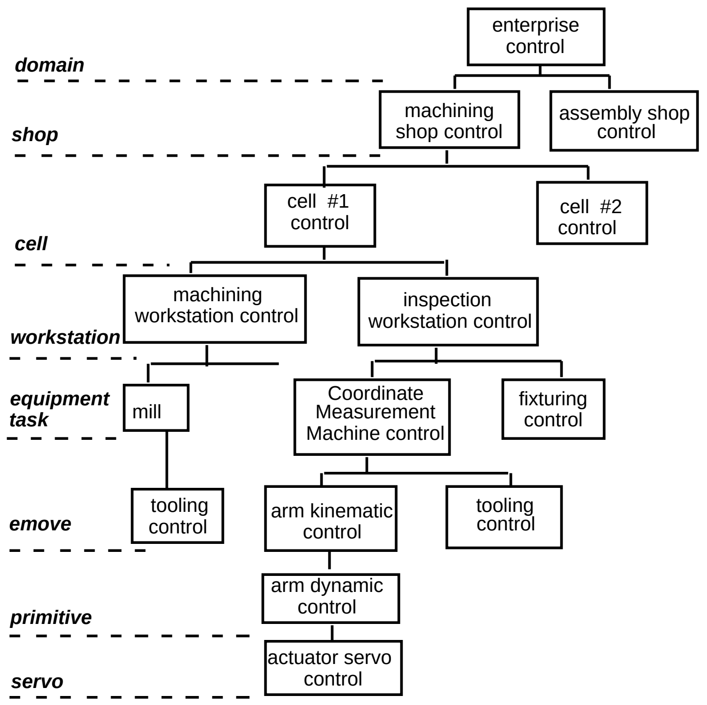


Figure 2: RCS Control Levels Based on the Principle of Multiple Levels of Resolutions

By following these guidelines, the RCS methodology have a sound basis for decomposing a system at its constituent, resolution levels. RCS provides as many control levels as required to cover the scope of individual problems. These levels form a smooth transition of spatial and temporal resolution from the highest to the lowest levels. Figure 2 illustrates an application of these levels to a manufacturing facility. The levels include application domain (the highest), shop, cell, workstation, equipment, elementary move, or emove (kinematic), primitive (dynamic), and servo. See [1, 17] for a detailed description of these levels. Each level may have zero or multiple control nodes (except for the highest level which has one node) all using the same model as described in section 2.1. At each level of the hierarchy, the control for individual and collective physical entities, such as robots, vehicles, workstations, manufacturing shops, and robotic motors, are modeled. Note that if the implementation problem were for the control of a machining center only, then the upper levels of control become unnecessary.

These guidelines also help operators to understand and anticipate the way a high-level goal can be decomposed into low-level detailed behavior.

2.3 Behavior and behavior generation

RCS applies to intelligent systems that make decisions and perform physical work in real-time. Therefore, behavior, describing actions, as opposed to data, is the most critical aspect driving the control system design. Albus and Meystel defined behavior as "an ordered set of consecutive or concurrent changes among the states registered at the output of the system or subsystems [31]." In a complex system, agents or subsystems are capable of performing certain behaviors. A mechanism is required to coordinate the individual behaviors to form system behavior. This mechanism does not necessarily have to be external to the agents, but can be embedded within the agents themselves and manifest itself as "cooperation."

In RCS, agents or nodes generate behavior through planning and execution processes. The objective of the generated behavior is to command and coordinate the sub-behaviors of all the nodes' subordinates. The system behavior may be initiated when a user enters a high-level goal. An incoming task for a manufacturing system may be to "complete the product order." The activity of the enterprise level node is to decompose the order into the language of the next lower-level group of nodes and assign and schedule the involved groups.

The workstations decompose their received tasks into the languages of their subordinate individual pieces of equipment. The equipment decomposes the incoming tasks into "elementary moves" that are (or are equivalent to) a series of physical motions of the equipment to achieve the task goals. The elementary moves are further decomposed into "primitives" that are (or are equivalent to) dynamic motion commands for the equipment. Finally, the primitives are decomposed at the servo level into the electrical or mechanical signal commands for the equipment actuators.

Manufacturing orders are completed through this behavior generation process that results in the desired machine actuation. Refer to [1, 17] for a detailed description of these levels.

Note that Object Oriented (OO) techniques also use the term behavior but in a different context [22]. In this paper, the term behavior refers to the definition given here.

2.4 Behavior as a principle of abstraction

Abstraction is defined as denoting the essential characteristics of an object that distinguish it from other objects, while suppressing non-essential details. Different types of abstractions are described in various OO paradigms, including object, class, data, function, and process [22]. The abstraction relationship among objects within a system leads to system hierarchies.

RCS views goal-oriented behaviors as the most important aspect of an intelligent system. Therefore, RCS uses behavior analysis to derive control system hierarchies. Each control node is highlighted with a finite number of tasks that it can perform. This may be referred to as the “task abstraction” or “behavior abstraction.”

2.5 Organization and Command authority

Modeling a system through the combination of the multiple levels of resolution and behavior generation concepts forms a control hierarchy. Figure 2 gives an example. These concepts further establish a relationship of command authority within the control hierarchy. The superior control nodes command the subordinate nodes. The subordinates report back the status of command execution.

2.6 Object-Oriented Perspectives: Control Nodes and Mapping Behavior to Node Hierarchy as a Means of Encapsulation

Encapsulation publicizes the interface of a model and hides the implementation (procedures and internal data). An RCS hierarchy is composed of the control nodes with a generic functional model as described in Section 2.1. The generic control node is considered one of the base types of objects. Additional base types are also being developed [32]. Nodes at different control levels are derived from this base type when extra characteristics that are specific and necessary are added.

In a class declaration within an object-oriented model, the interface is in the public portion, and implementation is in the private portion of the model [22]. In RCS, the task structure is integrated with the node hierarchy. Each node has a set of tasks that it is capable of performing. An input command set, corresponding to the task set, is used by the node’s superior to command the node’s behavior. In the same manner, the node’s behavior results in the generation of sub-commands for its subordinates.

In this way, in RCS, the nodes' behavioral capabilities are encapsulated via the task set that it is capable of processing. A given node within an RCS hierarchy needs to be aware only of its immediate subordinates' interfaces. It does not need to be concerned about how the subtasks the node passes down are carried out or if the subtasks are further decomposed.

2.7 Operator interface

RCS is generic in that it supports a full range of system operation modes, from teleoperation to autonomous. An operator interface is allowed at every level and at any node to facilitate these modes.

2.7.1 Basic Principles

Our basic principles for operator interface design are:

* to have a well-defined overall structure and well-defined roles for individual operators,
* to integrate with the system control,
* to integrate with the current operating environment,
* to provide a systematic model of conducting man-in-the-loop control and migrating legacy subsystems,
* to match user requirements and expertise,
* to allow emergency procedures,
* to optimize both real-time performance and operator workload,
* to inform intuitively, in real-time, and to provide as much information as operators require, without overloading, and
* to support system service and development, including testing and simulation.

2.7.2 Operator Levels of Commanding Authority

To support hierarchical real-time control, in RCS, operator interface functions are divided into six types of levels of commanding authority. Specific applications may not contain all the levels. From the highest to the lowest levels of authority, the level types are:

Level 6 -- application domain level operator. This is the highest level. The operator enters and monitors the overall command for the entire control system. In case study I, section 4, the command controller operator belongs to this level.

Level 5 -- group level operators. They handle control activities involving either multiple coordinated control entities or multiple coordinated groups of control entities. For example, machines, material handlers, and/or vehicles are often grouped according to the degree of coordination necessary to accomplish a set of tasks. The operators would be interacting with such groups at this level.

Level 4 -- equipment or task level operators. These operators deal, through their corresponding controllers, with the control activities of individual major physical entities, such as a single machine or vehicle.

Level 3 -- elementary move (emove) or subsystem level operators. The operators may monitor or command such elementary move control activities as avoiding obstacles and kinematic limits, including, in case study I, the possible instability of the submarine caused by large bubble (pitch) angles. At this level, an operator is interacting with a major subsystem of a larger machine or vehicle.

Level 2 -- primitive (prim) level operators. At this level, an operator might be interacting to control dynamic motions of machine appendages, perhaps setting machine acceleration or directing motion in a particular direction.

Level 1 -- actuator level operators. Through their corresponding controllers, these operators deal with direct environmental interaction, by controlling the electrical and mechanical signals that drive the movement of individual actuators toward the goals.

This hierarchical organization also allows a problem with a high level of abstraction to be logically and smoothly transitioned to a set of sub problems with lower levels of abstraction. In this way, the operator interface for a complex problem can be shared by a set of operators with well defined and limited responsibilities. The operators may act on different levels of abstraction at different times by controlling the modes of operator interaction.

3. RCS Methodology--The Development Process

3.1 Behavior Orientation, Object Orientation, and Data/Function Orientation

Behavior is considered part of an object's model in most of the object-oriented methods. For example, Coad and Yourdon describe parameters such as frequency and location, that control a radar system's behavior [24]. Shlaer and Mellor describe real-world things as having stages in their behavior patterns [25]. However, RCS seems to place heavier focus on the behavior than OO methods. In OO methods, objects are the defining element and "behavior is a part of objects." in the object-oriented methods. The RCS methodology is "behavior-oriented." Behavior is the most critical aspect in the control system software. The analysis of behaviors and tasks drives the system development effort while the related physical subsystems or components serve as a part of the constraints for the analysis.

From a global perspective, the system behavior determines an RCS hierarchy and the underlying control flow. In RCS, developers typically analyze and identify system behavior before designing control nodes, or "objects," to perform the behavior. This concept has not been emphasized in most object-oriented methods. The authors believe that this behavior orientation could significantly strengthen the general object-oriented methods, particularly in the large, complex, real-time control problems dealing with physical tasks where the RCS architecture excels.

In the RCS methodology, the analysis of data and data processing are driven by the task performance requirements. Data and processors are also organized to support task performance.

3.2 The Methodology Steps

RCS is intended for systems that generate intelligent behaviors to achieve desired goals. The first issue in the system development effort is understanding the problem area, i.e., what is the physical operating environment for the system to be developed and how will the system operate in that environment. The RCS methodology prescribes a behavior analysis and scenario development process to address this first issue. The development process is shown in Figure 3. Designers may develop the operational scenarios through reviewing literature, such as operating manuals, and through interacting with domain experts. The latter is the most effective to obtain first-hand knowledge. Among the RCS applications, in the submarine automation project, the researchers interacted with several retired submarine commanders throughout the project period. In the NAMT Inspection Workstation project, the researchers interviewed shop floor inspectors. In the Army Next Generation Unmanned Vehicle (XUV) Demo III project, the researchers coordinated with the Army Mounted Maneuver Battlefield Laboratory (MMBL) in Ft. Knox, Kentucky. The details are described in the later sections.

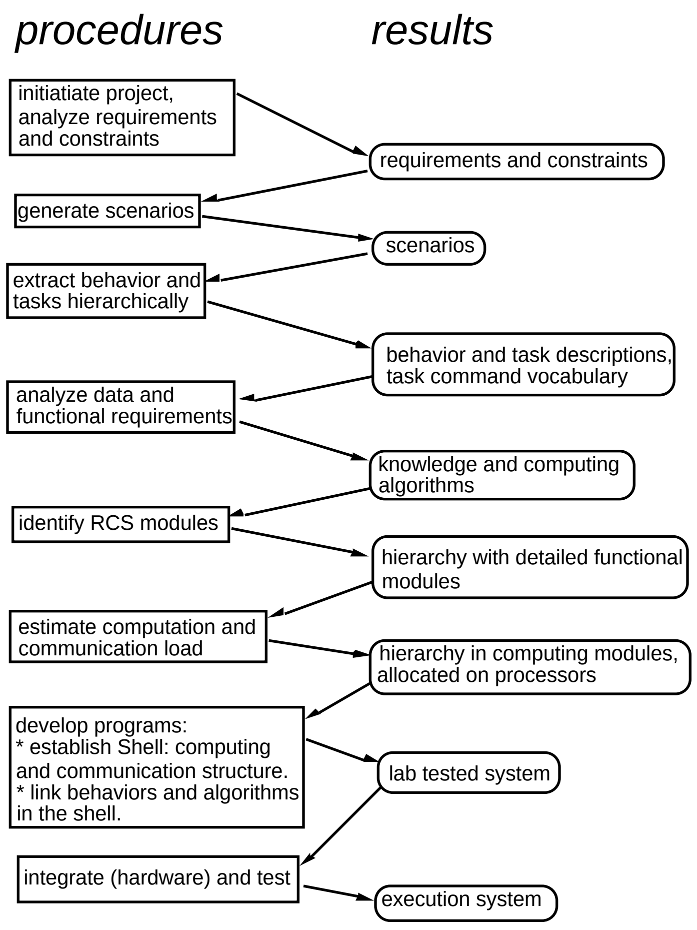

procedures
results
initiatiate project, analyze requirements and constraints
requirements and constraints
generate scenarios
scenarios
extract behavior and tasks hierarchically
behavior and task descriptions, task command vocabulary
analyze data and functional requirements
knowledge and computing algorithms
identify RCS modules
hierarchy with detailed functional modules
estimate computation and communication load
hierarchy in computing modules, allocated on processors
develop programs: * establish Shell: computing and communication structure. * link behaviors and algorithms in the shell.
lab tested system
integrate (hardware) and test
execution system

Figure 3: RCS Development Process

In the behavior analysis and scenario development process, the system goals and functional requirements are analyzed in terms of how the users expect to use the systems and what they expect the systems to perform. Goals drive behaviors and scenarios. Behaviors and scenarios define system requirements, including those of subsystems, information, communication, and integration. This behavior analysis process is constrained by factors including conformance to standards, legacy system compatibility, technology limitations, and programmatic requirements.

In the behavior analysis process, the “action verbs” in scenario descriptions are identified as task commands. The task commands are structured, using the multiple resolution principle, as described in section 2.2, as task trees. These task commands and parameters serve as the vocabulary to model the intelligent behavior of the control system.

Corresponding the reference model with the task knowledge results in a control hierarchy. Iterations are required to obtain a control hierarchy with an appropriate number of control levels and nodes and to map the task trees onto the hierarchy at the appropriate levels of abstraction.

The task knowledge is further analyzed to identify the explicit data, communication, and computation requirements. It is emphasized that only explicitly and specifically stated requirements can be programmed into executable software. This process also scopes the system capability in the sense that the developers must determine how high a priority it is to implement a particular task and whether the task performance is autonomous, semi-autonomous, or manual. To execute a task, developers determine whether a certain piece of information is provided a priori or generated on-line. Next, the developers must determine which sensors, processing algorithms, computing platforms, and databases to use.

For example, in a manufacturing scenario that involves a machining sequence, one specification is whether a schedule is generated off-line for selection and interpretation into the machine native language by the machine controller, or, alternatively, whether the schedule is to be generated by an on-board, real-time, planning system that operates on, for example, a CAD (Computer-Aided Design) model of the part.

In a military scenario, when multiple next generation unmanned vehicles (XUVs) are to perform a reconnaissance, surveillance, and target acquisition (RSTA) mission, the developer needs to determine whether an area map is given and, at what resolution, or whether the vehicles are to build the map as they survey the area. The differences would affect the sensor selection and other design decisions.

The results of this scenario-development and behavior-analysis process include the design of a control hierarchy, the input and output of each node in the hierarchy, the behavior of the nodes, and all the processing and data requirements. These results are organized per RCS reference model.

Computation and communication load is estimated to form computing processes. A template-based approach is used to first build a system shell for the hierarchical control nodes before knowledge, processing modules, and behavior modules are linked in the shell. Laboratory and field tests are performed and results are used to modify the behaviors, the processing algorithms, and the computation and communication load distribution among the computing processes.

3.3 Behavior Analysis, Scenario Development, and Domain Expert Interaction

In a military scenario, a behavior to be analyzed would be for the XUV scout platoon to "Establish Observation Post," in which the first sub-activity is to "Receive Mission." The RCS developers determine that the mission statement would be implemented as a task command in the control hierarchy. In preparing interactions with a crew of Army soldiers, the developers must be prepared to explore, or verify, with the soldiers the data included in the mission statement. The developers are to generate a list of questions to stimulate the interactions, to catalyze the soldiers' stories, and to maximize the scenario descriptions useful to system development. The questions related to the mission statement may include:

- The mission statement may list Named Areas of Interest (NAIs). What are the purposes and how are they specified?
The answer is that the upper level commanders would identify areas that potentially contain useful features as NAIs. For example, areas containing rough terrain may be good for defensive purposes. NAIs are specified with four coordinates to enclose the interested areas.

- The mission statement may list Observation Post (OP) sites. Is it sufficient to describe them as a list of coordinates? Is there any other information that needs to be specified, such as time to reach the sites? What are some considerations in selecting an OP?
The answers are that, an OP can be specified via either coordinates or landmarks. Additional information is often required when an OP is assigned. Timing constraints are critical when fire suppression at a certain time on the OP

is planned. The upper level commander must associate this information with the OP for the XUV to avoid the fire. Some considerations for selecting an OP are whether it is cleared, it allows an escape route, and it supports the planned areas of advances (AOAs).

On the opposite side of the interaction, architectural developers may provide soldiers some flavors of how the behaviors may be implemented, such as:

The section level Behavior Generation planner module may contain a main flow chart describing the "Establish Observation Post" activity. At various steps in the flow chart, the planner queries the map and the knowledge base (both are parts of the world model) to either make a decision or invoke a specific planner. For example,

```
Query WM <Has area been cleared?>
    <Yes>  (1) Invoke "formation planner" to determine a formation for the vehicles to proceed.
           (2) Invoke "path planner" to generate waypoints for the vehicle formation to follow.
    <No>   (1) Invoke "path planner" to generate vehicle coordinated paths to conduct area reconnaissance.
           (2) Command vehicles to conduct certain RSTA subsystem activities, including target identification.
           (3) Command vehicles to conduct certain weapon subsystem activities, including preparing for fire suppression support.
```

The section level would have an executor for each vehicle in the section. The executors execute the generated schedules containing steps involving various vehicle behaviors (navigation, RSTA, etc.).

All of the proposed activities need to be verified with or obtained from the soldiers in order to be credible. Some consensus can be reached as to steps that are essential and must be implemented, or are less significant and designated for future expansion.

The results of this scenario development and behavior analysis process include the design of a control hierarchy, the input and output of each node in the hierarchy, the behavior of the nodes, and all the processing and data requirements. Later sections provide implementation examples.

3.4 Structuring Behaviors in Hierarchy

Scenarios are described in a natural language form. The next step toward a computational system in the RCS methodology is to structure the descriptions in a hierarchical way. The following example (see Figure 4) illustrates the process.

In a manufacturing inspection scenario [33], the current activity is to perform system initialization to prepare for inspection tasks. The initial control hierarchy has been drafted (see the bottom of the drawing for the middle levels of the entire hierarchy). The workstation (WS) receives a "initForInspect" command. The first subtask that WS decomposes for the subordinates, Measurement (MS) and Fixturing (FX), is "equipSetup." The arrowhead pointing downward shows the command process. MS and FX decompose their received task command but such activities are omitted from the diagrams. MS and FX report to WS a "Done," illustrated by an upward arrowhead, when the subtasks are completed. In WS, the next step for initialization is "calibrateCMM (coordinate measuring machine)," for MS only. MS decomposes the task into a sequence of subtasks including "calibrate Reference Sphere" and "calibrate Probe" for its Tooling subordinate. These activities include align all the coordinates and use the reference ball to calibrate the probe on the CMM. See [33] for a detailed explanation of these activities. In performing these activities, the Tooling operator is given instruction displays as an aid. The displays include: what to do (to define the coordinate system) and how to do (orientation of the probe). Pictures provide further assistance.

Once Tooling and MS are done with calibrating the CMM, WS issues the final activity for the initialization task, "calibrate Camera." Once the done is rippled back from the low levels to WS, it sends a done to its superior.

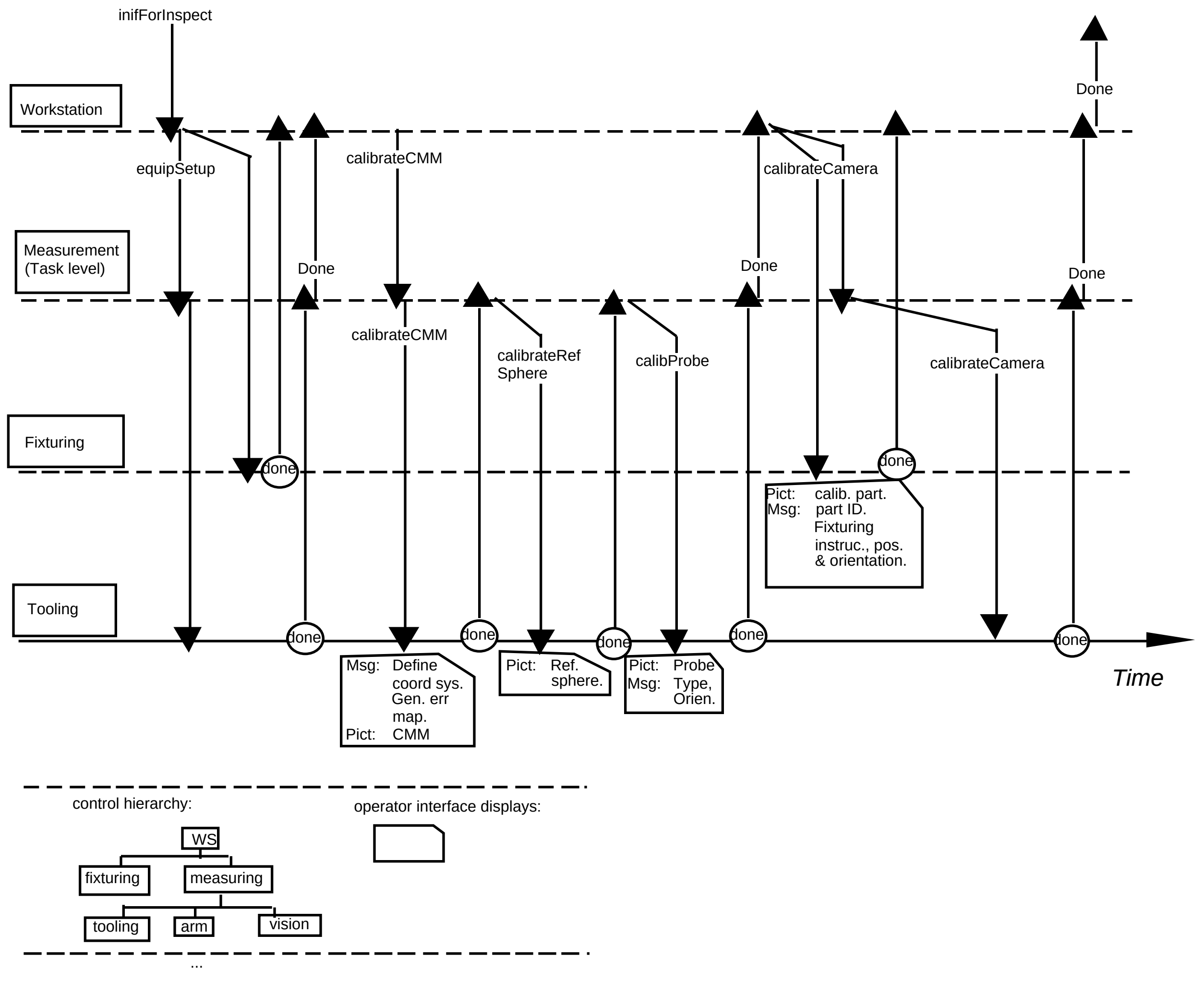


Figure 4: Mapping Behavior on Hierarchy

The later sections on implementation cases use either this method as is or a modified version to perform RCS development.

3.5 Template Based System Implementation

In RCS, control nodes perform designed behaviors. The generic structure of the control nodes warrants the creation of process-template sets for the control nodes and their sub-modules. As such, the control nodes are designed via the behavior paradigm and yet implemented in an object-oriented notion. The generic command and status communication protocol in the RCS hierarchy warrants a separate set of templates.

3.5.1 The Functional View, Multiple Granularity, and Current Scope

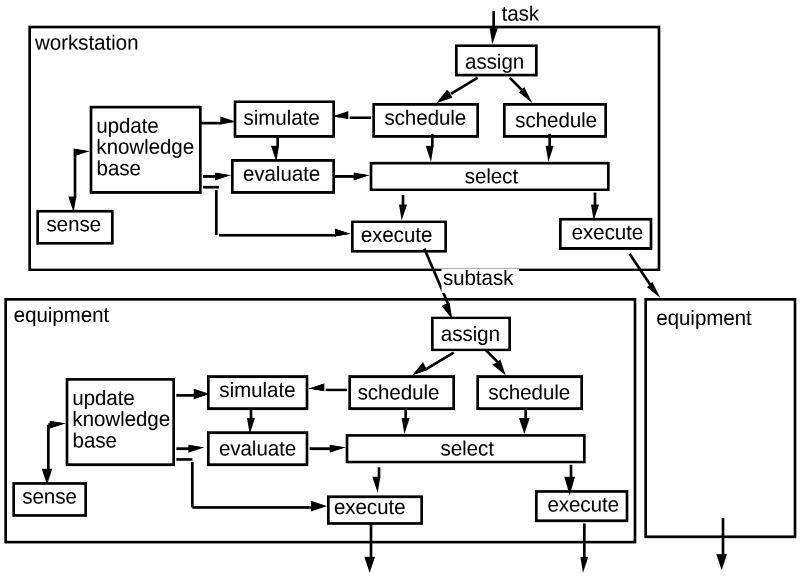


Figure 5: Functional Layout of RCS Process templates

Figure 5 shows a processing view of an RCS control node. The approach is to model each box with a process template. For example, there may be a node template. There may also be templates for individual node functions, such as behavior generation and sensory processing. Figure 6 illustrates this effect. The process templates contain generic functionality as described in the RCS reference model, but would be further derived and instantiated at different control levels and applications. The process templates for the different RCS functions, as seen in Figure 5, may be different. The behavior generation functions may be suitable to be modeled using a finite state machine (FSM) based process template. Such a FSM model may not be suitable for a knowledge-base maintenance function. See [29] for a description of the sensory processing template.

In software implementation of a control hierarchy, the nodes and their functions are grouped, according to computation and communication load distribution, to form independent computing processes. All the control nodes should contain all the functions as described in Figure 5. In a situation when a control node performs a significant amount of planning and scheduling work for a number of subordinates, the node’s Job Assignor (JA), Schedulers (SC), Plan Selector, simulator models, and state estimation algorithms may all be complex and executed independently in a distributed environment. When a particular planning algorithm calls for intensive communication among the JA, SC, simulation, and value judgment modules, they may be grouped as an independent computing process, which is referenced as a planning (PL) process in this report. Note that, this report focuses on the behavior generation process templates. The sensory processing and world modeling aspects are left for the future development.

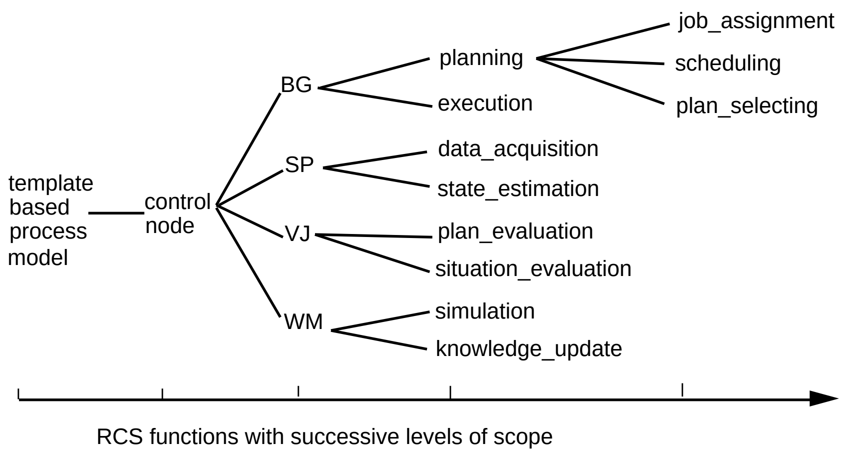


Figure 6: Process templates May Cover Different Functional Scopes of RCS Node

3.5.2 The Base Class Generic Process Template

The template approach has been an evolutionary process in the NIST Intelligent Systems Division. An earlier paper [17] described a previous implementation. This report describes the latest [34]. Figure 7 shows the generic process template, which, in the context of C++, is called a base class. The template describes the basic processing flow of most of the RCS modular computing processes, or abbreviated as processes. The arrows indicate the process' communication, based on a non-blocking command and response protocol. This template is further expanded to form specific RCS functions, such as the finite-state-machine-based execution model, the planner (PL) template, and the executor (EX) template. The next section describes the template expansion process, as a part of the methodology, whereas the later case-study descriptions show the template expansion results.

In the base-class, generic, process template, a process has the following communication channels:

- A pair of command input and status output channels to the superior processes.
- As many pairs of command output and status input channels as the number of subordinate processes the process has.

- A pair of operator input/output channels.
- As many data read/write channels as necessary to support the decision making process.

This model is consistent with RCS that addresses the one-writer multiple-reader paradigm for each data entity.

This process executes the following functions:

- Preprocessing:

  * Read in required globally shared data. Read input data from all the coordinating processes, typically including the superior and subordinate processes and the operators. Internet or any network may be used to implement the operator interface function to facilitate distributed system activities.

  * Evaluate thresholds, flags, or status or process any other necessary prerequisites to support the following computing process.

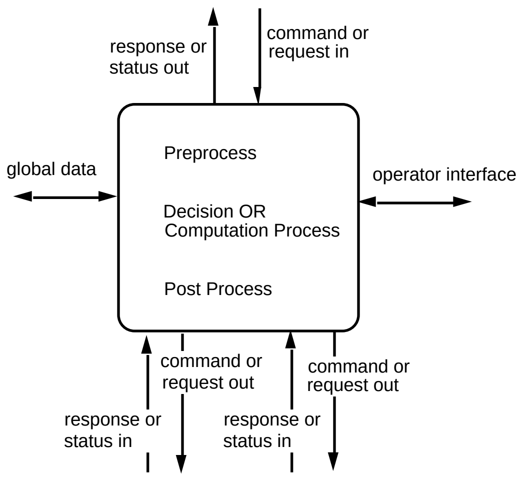


Figure 7: An Base Class Independent Process Generic process template

- Computing or Decision process:

  Invoke a computing algorithm based on the world state and the execution status of the subordinates. In the situation when this base class is used to implement finite state machines (FSM), this process would select and execute a state table. The first implementation case, in a later section, will describe this FSM model in further detail.

- Post processing:

  Write decisions (sub-task commands) to subordinate processes. Write to operator when required. Post global data. Post responses or status reports to superior or upstream process in general.

- Support Facility:

  Real-time performance algorithms compute features such as the percentage of the total execution time that is spent on particular states. An operator interface function allows accessing the command and status communication channels to monitor input and output.

3.5.3 Command and Status Templates

RCS prescribes a command and status execution model. Each command is associated with a serial number. The subordinates who receive the commands are to verify the numbers against the numbers associated with the last received commands. The subordinates then echo the current serial numbers back. The superior is to verify the echoed numbers to determine whether its commands have been successfully received.

In addition to the echo, the subordinates also send the superior the information on command execution status: executing, done, or error. In a finite-state-machine execution model, the subordinates also send the state number information to indicate the particular segments of the state tables that the subordinates are executing.

The authors use a set of base classes [34] as the templates for implementing system command and status messages. This serial-number-and-echo model is also extended to general request-and-response activities among the node processes. Note that the message implementation must use the same communication system as in the generic process base class.

3.5.4 Template Inheritance

The RCS methodology employs multiple layers of inheritance [23] for the templates. A base-class, generic-process template establishes the communication and processing flow. These properties are inherited to form the executor and planning templates. The executor contains additional functionality of invoking emergency action when normal execution fails. The planning template contains additional functionality of handling planning horizon. These two templates are further inherited when they are used by specific applications. Figure 8 shows this effect. In this way, common RCS characteristics are implemented consistently. This method also reuses common code systematically.

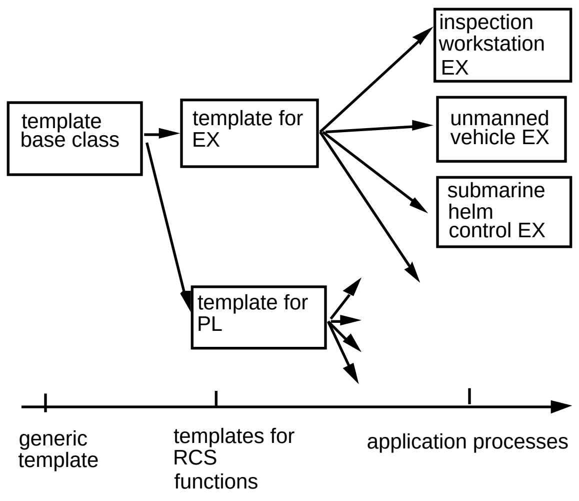


Figure 8: Templates Inherit Common RCS Characteristics

The base-class, process template is also used to implement a finite state machine model. This model reads all of its inputs in the preprocessing section of the template. The decision process selects and uses a state table or graph to decide its next state and performs its data processing based on the computed state transition. The post processing writes all of its outputs, and waits for all its next execution cycle which is usually based on a time-based execution clock.

3.5.5 Generic Shell: Using Templates to Build RCS Nodes and Functions

Developers use the templates to build their applications. The process templates contain standard RCS interface channels. When implementing a node or a function, e.g., the machining control node planning module shown in Figure 9 as the “mc PL/Sim/VJ/PS” box, the developers specify this box’s interfacing modules. The interfacing modules include the workstation, “mc SPWM (sensory processing and world model),” “mc EX Motion,” and “mc EX Tooling.” These modules are connected via the generic interfaces as provided in the base-class, process template. The developers, then, specify the data that go across each of the interface lines. The following is a list of RCS interfaces that is being used and yet evolving:

* The PL/Sim/VJ/PS sends a schedule to EX. The schedule can be a manufacturing process schedule, a navigation path for mobile robots, a pointer to a state graph, etc.

*	The EX sends a task command to the subordinate PL. The sub-task command is typically a step in the schedule. The EX may send more than one schedule step to meet the subordinate's planning-ahead requirements.

*	The SPWM sends supporting data to PL and EX. PL and EX may require the same types of data but PL requires the data that apply through the planning horizon while EX may need the data applied to the current time. In a manufacturing environment, a planner may need to know whether the resources are available for the next X minutes in order to generate a schedule for the X minutes, while an EX needs to know whether the resource is available right now in order to control the next plan step. In a path planning situation, the PL and EX may need map data or obstacle lists.

A linking facility is employed to allow specific algorithms to be integrated to perform required functions. A control module template is associated with a linker template that lists the kinds of software library or processing algorithms that the control module may need. A PL template typically links different types of software from what an EX does. In a UNIX operating system environment, the linking procedure is done via the Makefile facility. In the Figure 9 example, the planner module employs an A* search based planner [35], shown in dotted lines. The EX may also link in a pre-generated, state table-based plan for execution. These are shown as dash lines. The linking process also facilitates code sharing.

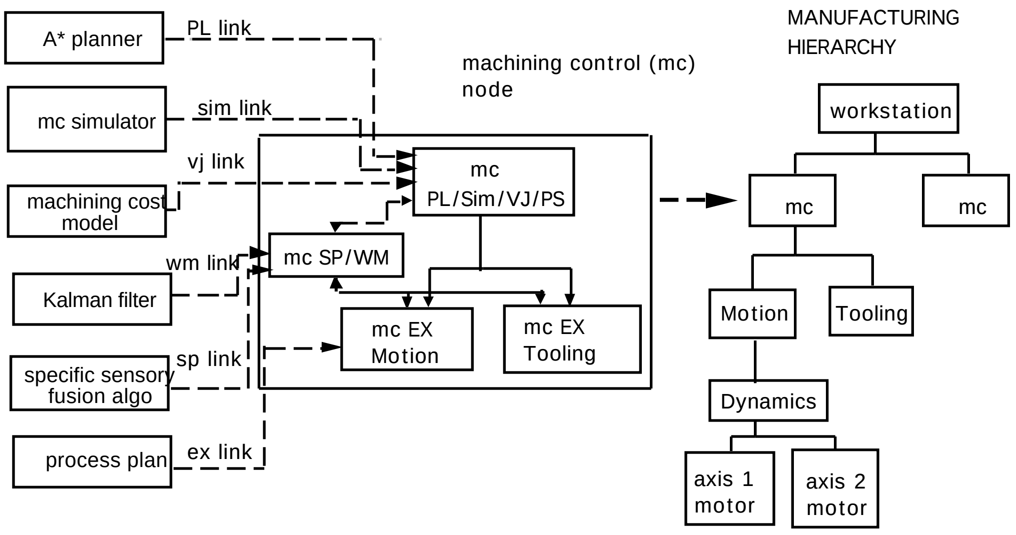


Figure 9: Applying Templates to System Architectural Implementation

Developers can apply the process templates at an early stage and construct a skeleton control hierarchy capable of interfacing, linking, and executing individual component technologies. This is called the generic shell for the targeted control system. This shell serves as the backbone for the system operations. The developers construct a control-node “shell” and the control-hierarchy “shell” by specifying the interfaces, shown in Figure 9 as connecting the solid lines, and by identifying the required algorithms, shown in Figure 9 as connecting the dash lines. Note, again, that the authors plan to revisit, in the future, the issue of implementing the SPWM process template.

3.5.6 Combining Behavioral-oriented, Object-oriented, and Functional Decomposition Paradigms

The method described in the above section shows how RCS applies the three prominent system development paradigms: behavioral orientation drives system analysis and design; Object-orientation facilitates software design and implementation; and Functional view describes control nodes explicitly and in detail. Finally, in system operations, behaviors are generated by objects serving as system control nodes.

3.5.7 Analysis and Implementation structure

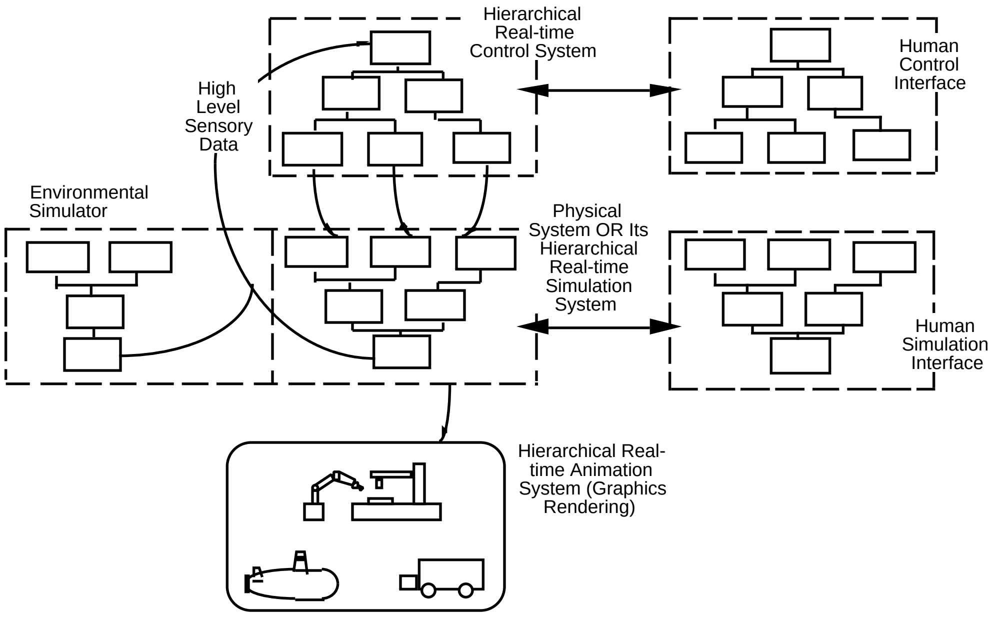
The RCS methodology proposes a generic and comprehensive logical structure to facilitate the analysis and implementation. This structure identifies and integrates the following five hierarchies: control, simulation, animation, operator interface (OI) for control, and OI for simulation, as seen in Figure 10. The reasons for requiring these hierarchies are:

The control hierarchy performs real-time system control to accomplish missions. The lowest level nodes send electrical, hydraulic, or mechanical control signals to physical actuators. However, there are many situations during which simulation is either extremely useful or typically required. The situations include laboratory development, testing, training, and maintenance functions. Inside the control nodes, simulation is also used to facilitate planning.

Simulation may take different context or cover different scope in these different situations. Simulators typically employ kinematic or dynamic models of the to-be-controlled physical systems or other environmental objects that are within the controlling scope. At different hierarchical control levels, control nodes output task commands at different levels of abstraction and anticipate effects at the corresponding levels of abstraction. Therefore, it is beneficial to have the simulation also hierarchically structured to provide the anticipated effects. The lowest level simulator nodes, i.e., the actuator simulators, receive control signals and compute the mechanical movements of each actuator. The next higher level simulation nodes integrate the simulated actuator movements and compute the system dynamics. The same process continues up the simulation levels that correspond to the control levels.

As such, the resulting simulated states at different levels can be fed back to the higher levels of the control hierarchy via the simulated sensors.

The environmental simulation is treated in the same manner.



Hierarchical Real-time Control System
Human Control Interface
High Level Sensory Data
Environmental Simulator
Physical System OR Its Hierarchical Real-time Simulation System
Human Simulation Interface
Hierarchical Real-time Animation System (Graphics Rendering)

Figure 10: A Software Structure for Analysis and Implementation

Animation performs a 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. Animation does not compute the dynamic nor kinematic behaviors of the objects.

The RCS architecture specifies that all the nodes permit operator interaction. This is why, in Figure 10, the OI is described as a one-to-one correspondence to the control and simulation hierarchies. Note that, this shows a functional relationship. In implementation, it is possible not to follow the one-to-one relationship. A single operator may use a keyboard as the input device to interact with multiple control nodes. In case study I, the authors demonstrate that an operator display interacts with multiple 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.

4. Case Study I, Finite State Machines, Simulator Templates, and Operator Interface Focused Implementation

RCS has been applied to many control systems [36, 37, 38, 39, 40, 21]. In this section, the authors describe the submarine automation demonstration system project that was supported by the Defense Advanced Research Projects Agency (DARPA) Maritime Systems Technology Office. This project was one of the first test cases for the described development steps, section 3. The software demonstration system contains multiple levels in its control hierarchy, simulation, and animation [18, 41]. The authors studied and extended the base-class, process template to all these software aspects, i.e., control hierarchy, operator interface, simulation, and animation.

Submarines often conduct missions in hostile and uncertain environments in both peacetime and in wartime. Crew members routinely make critical decisions at many levels of authority in a coordinated and concurrent fashion while the submarine is underway. A huge volume of information, including sensory data, mission and systems status, a priori knowledge, and mission requirements, must be processed, organized, and made available to support the decision making process in real-time. In today's submarines most of the information produced by the sophisticated submarine subsystems is manually organized, evaluated, and communicated between watchstations by crew members. As submarine subsystem technology has advanced, crew size and, in turn, submarine size has increased in order to handle the information processing load. Therefore, the current challenge is how to process and communicate the information in a smarter and more efficient way without increasing crew size. The ultimate challenge is crew size reduction. As the U.S. is trying to realign its economic and defense priorities, cost effectiveness and operational efficiency are vital to the strength of its defense forces. Automation, computer, and control system architecture technologies have advanced enough to meet these challenges.

This project demonstrated, in simulation, automatic maneuvering and engineering service systems for a 637 class nuclear submarine. The maneuvering system involved steering, trim, depth, and propulsion control. A rudimentary engineering service system demonstrated the ship's ability to respond to a casualty report. Ventilation line-up's are reconfigured in real-time. Sophisticated maneuvers were performed to clear baffles, to raise the depth to enable the diesel engine to perform emergency ventilation, and to engage the ship’s emergency propulsion control. Human computer interaction was described in detail.

In this project, all the behaviors are described as state diagrams and tables at the system development stage except for ice avoidance capability. Therefore, the template utilization is limited to a finite-state-machine model. The other RCS functions, such as sensory processing, world modeling, and the ice avoidance planning, are embedded in the base-class, process template as an implementation simplification.

4.1 Scenarios

ISD researchers collaborated with retired submarine commanders to develop the demonstration scenarios. Based on the pre-established goals of the simulated submarine, they described, in detail and in every aspect, how a submarine would operate to achieve the specified goals. The following are some of the guidelines used to maximize the relevant knowledge captured from the domain experts. The developers did not impose any structural constraints during their descriptions. The commanders were encouraged to use the submarine terminology. The main role of the developers at this stage of interaction was to be goal driven, to be intelligent listeners, and to ask stimulating questions sparsely to catalyze their stories.

A brief introduction of the submarine mechanisms is given first to facilitate understanding.

4.1.1 Maneuvering Mechanisms

A submarine has a number of mechanisms for hydrodynamic control of its depth, buoyancy, orientation, and speed (see Figure 11). The authors briefly introduce a selective set of them below:

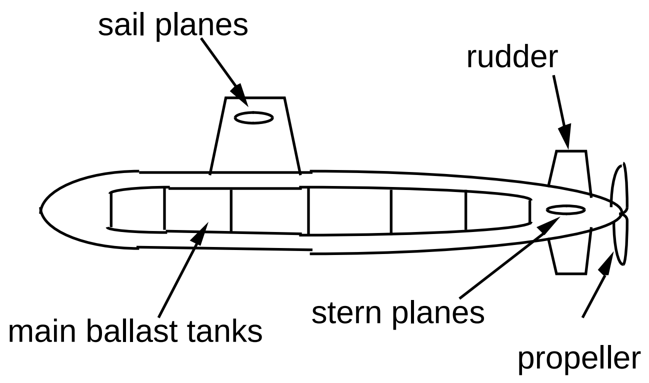


Figure 11: Submarine Hydrodynamic Control Mechanisms

* main ballast tanks

* variable ballast tanks
* fairwater or sail planes
* stern planes
* rudder planes
* propulsion mechanisms

The sailors use the Main Ballast Tanks (MBTs) for gross control of the ship's buoyancy, specially for submarine submerging and surfacing. If the six tanks are flooded (completely filled), the submarine gains negative buoyancy and the ship submerges. If the tanks are blown (emptied), the submarine gains positive buoyancy and surfaces. The tanks are blown by admitting high-pressure air through valves at their tops. Conversely, they are flooded by allowing the air at the top of the tanks to leave through the vent valves while seawater floods through ports at the bottom of the tanks. The submarine may gain its neutral buoyancy when the MBTs are full and the variable ballast tanks are at their prescribed levels.

The Variable Ballast Tanks are used for small adjustments in buoyancy and orientation. "Variable ballast" as used in this paper refers to any of the following tanks: forward, aft, water round torpedo, auxiliary, and depth control. An example of buoyancy control via variable ballast follows. At greater depths the pressure of the water outside the vessel is much higher and causes the pressure hull to contract. The weight of the ship remains the same; however, the volume of the water it displaces decreases. The result is negative buoyancy and the ship will start to sink. Blowing water from the variable ballast tanks, which decreases the weight of the ship, may be sufficient to stabilize the submarine's buoyancy. The variable ballast tanks are also utilized for trim/orientation control. Weight distribution in a submarine may change after it has been at sea for some time; for example, the supplies that were brought on board might be consumed which results in a change in forward and aft weight distribution. Specifically, the submarine will experience a downward pointing pitch (the bow will be at greater depth than the stern). Water may be transferred between the forward and aft trim tanks to resolve the imbalance.

The Fairwater or Sail Planes are located on the conning tower and have a range of $\pm 22^\circ$. The Stern Planes are located at the rear of the submarine and have a range of $\pm 27^\circ$. These planes are used for depth control, either in a combination or separately. Their distance from the center of mass provides a means for changing the pitch of the submarine. The Rudder provides a means for steering the submarine left or right with limits on its range of ± 37˚.

The Turbine is the main propulsion mechanism, which enables the submarine to travel “ahead” at a commanded speed. A DC motor provides propulsion power during emergency situations.

4.1.2 Traverse through Bering Strait Scenario

In this demonstration scenario, a 637 class submarine traverses the Bering Strait in stealth mode. Much of the travel will be under ice, which creates a need for obstacle avoidance to be resolved by the control hierarchy. Fluctuations in water density may require maneuvering mechanism adjustments. The submarine encounters changes in seawater density at fresh water runoffs, where rivers of fresh water cause the water density to drop suddenly. The drop in the density of the seawater will cause the ship to have negative buoyancy and it will start sinking.

4.1.3 Open Sea Maneuver and Casualty Responses Scenario

In this demonstration scenario, the submarine is conducting a submerged transit of the open ocean at its standard speed (15 knots, or 7.7 m/s) and at a keel depth of 120 m. A watchstander--a submarine term, meaning a crew member who is assigned to a designated onboard location, which itself is called a watchstation, to perform pre-specified duties--informs the Maneuvering Room on the sound powered phone circuit that there is fire in the lower level Engine Room. The fire is reported to be in the vicinity of the main lubrication (lube) oil pumps. The Engineering Officer of the Watch (EOOW) passes the word to the Officer of the Deck (OOD) in the Control Room on both the sound powered phones and the intercom announcing system.

The OOD directs the Ballast Control Panel (BCP) operator to pass the word on the general announcing system (1MC) "Fire in the engine room. All hands on EABs (Emergency Air Breathing system masks)," and to sound the general alarm. The OOD completes the following actions for coming to periscope depth:

- Clearing baffles.
- Checking for sonar contacts, close contacts.
- Slowing and changing depth (Ahead one-third, keel depth 18 m).
- Raising the periscope.

Upon hearing the general alarm the crew proceeds to their assigned general emergency (battle) stations, relief of the section watchstanders occurs.

The damage control party fights the fire in the engine room. On indication of decreasing main lubrication (lube) oil pressure, the EOOW recommends to the OOD that propulsion be shifted to the EPM (emergency propulsion motor) in order to secure the main lube oil to the propulsion turbines.

The Officer of the Deck (OOD) orders, "All stop, shift propulsion to the EPM." 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 EPM operator operates the hand wheel to control the EPM and increase the motor speed to ahead two-thirds.

The damage control party reports to the Engineering Officer of the Watch (EOOW) that "The fire is out, the reflash watch is stationed." The EOOW relays this word to the Officer of the Deck (OOD). The OOD directs the word to be passed "Prepare to emergency ventilate the engine room with the diesel." The BCP selects the ventilation lineup setting it to emergency ventilate the engine room using the diesel engine. When the lineup is proper, the BCP operator reports to the OOD "Prepared to emergency ventilate the engine room with the diesel." The OOD directs "Commence snorkeling." The diesel engine is started and emergency ventilation of the engine room is commenced to remove the smoke and noxious gases from the engine room. The OOD directs that the atmosphere analyzer be used to sample the engine room atmosphere. The atmosphere sample shows that the carbon monoxide level in the engine room is 800 ppm. The Ballast Control Panel (BCP) operator uses the ventilation control panel to determine that with this level of carbon monoxide and ventilation configuration, it will take 80 minutes to reduce the CO level to an acceptable 5 ppm.

As the emergency ventilation of the engine room with the diesel continues, the atmosphere throughout the ship is checked in several locations. In areas where the atmosphere analyzer shows normal conditions, the Officer of the Deck (OOD) grants permission for the removal of Emergency Air Breathing system masks (EABs). When the atmosphere in the engine room reaches acceptable conditions the OOD will order "Secure emergency ventilation of the engine room with the diesel, recirculate." The BCP operator will use the ventilation control panel to line up for normal submerged ventilation. The OOD will order "Secure from General Emergency. Secure from fire in the engine room." The normal underway watch section will resume the watch. The diesel engine and generator will continue to be used to supply power for the emergency propulsion motor (EPM) until the main lube oil system is again ready to supply lubrication to the turbine bearings. When the main lube oil system is restored, the turbines are warmed up with steam, the EPM is ordered to "All stop" and then the clutch re-engages the turbine and the shaft. The EPM circuit breaker is opened. Propulsion orders are again answered using the main engines and propulsion turbines.

4.2 Behavior Analysis

Developers follow the scenarios and identify the events and entities including control system tasks, simulation events, and involved control nodes. These events and entities are put, in order, on the appropriate parts of the logical structure as described in Figure 10. This results in a system analysis diagram as shown in Figure 12. The tasks and events correspond to the verbs of the scenario descriptions. The control nodes correspond to, but are not equal to, the sailors performing the operations as stated in the scenario descriptions. In the scenario, the first event that occurred was a report on lube oil fire. In the system design, this malfunction should be initiated by an environmental simulator operator interface (OI, see Figure 13 for the implemented OI). Once the simulator receives the trigger, the smoke generation simulator ramps up the smoke contamination values. Following the arrows on the diagram, the control system employs a ventilation subsystem sensor that detects the contamination and reports it to the engineering system node. In this exercise, the scenario description is mapped onto the complete software system structure.

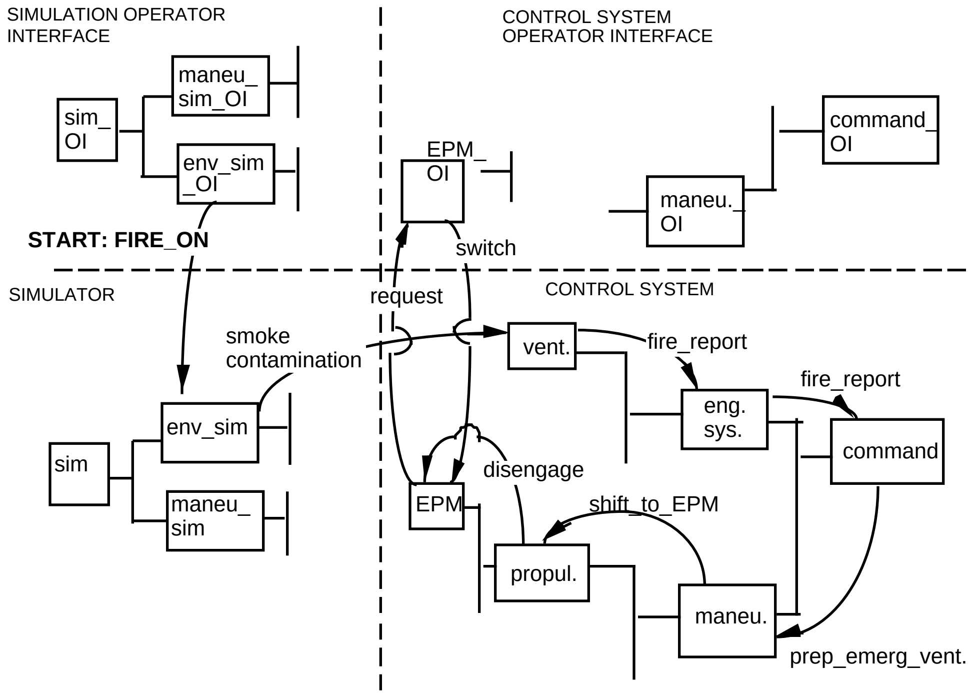


Figure 12: Task Analysis Diagram

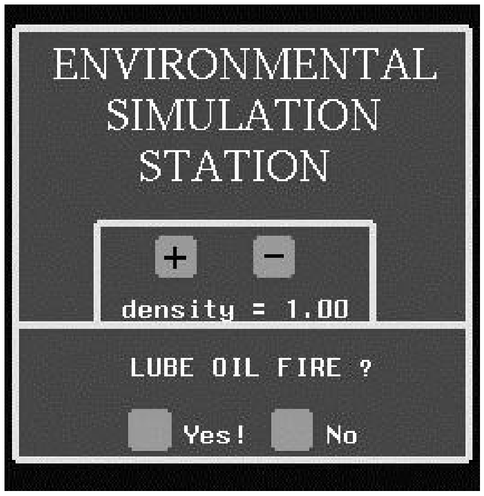


Figure 13 Environmental Simulation Operator Interface

The developers then model, via state diagrams, the sentences in the scenarios with which the verbs are associated. The state-diagram-development process also reveals the data and the data processors that are required to support the state transition processes. These state diagrams and data processors are associated back with the node tasks.

The control system behavior generator process selects and executes proper state diagrams based on the system goals and the environmental situations.

Figure 14 shows the resulting control hierarchy. Figure 15 and Figure 16 show the resulting task structures. RCS is scalable. Therefore, systems can be expanded or reduced. The shaded control nodes are added at a late project period to support the newly required open sea maneuver scenario.

A task tree provides a structure for organizing the system knowledge. The two task trees support the two scenarios, while task behaviors are shared for different scenarios when applicable. Multiple, higher plans may require the same set of lower-level plans. In these cases, the capability to avoid resource contention problems should be carefully built into the appropriate plans.

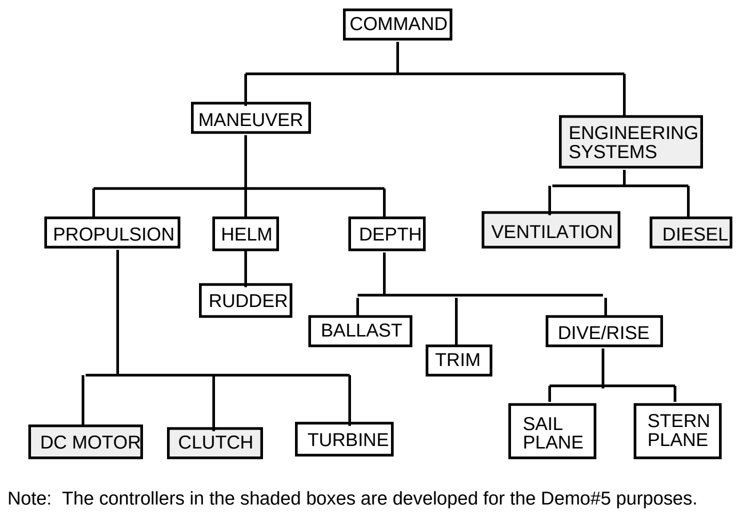


Figure 14: Submarine Automation RCS Control Hierarchy

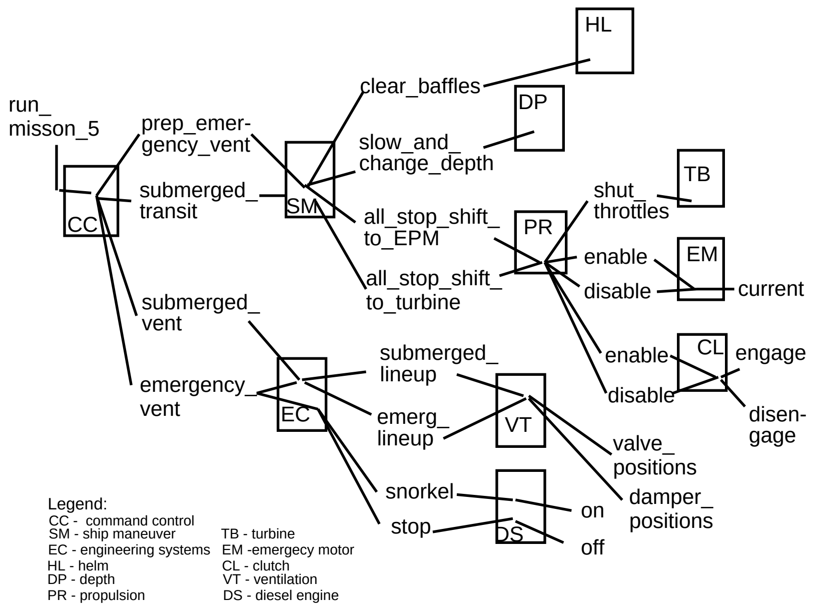


Figure 15: Submarine Automation Task Tree for Casualty Scenario

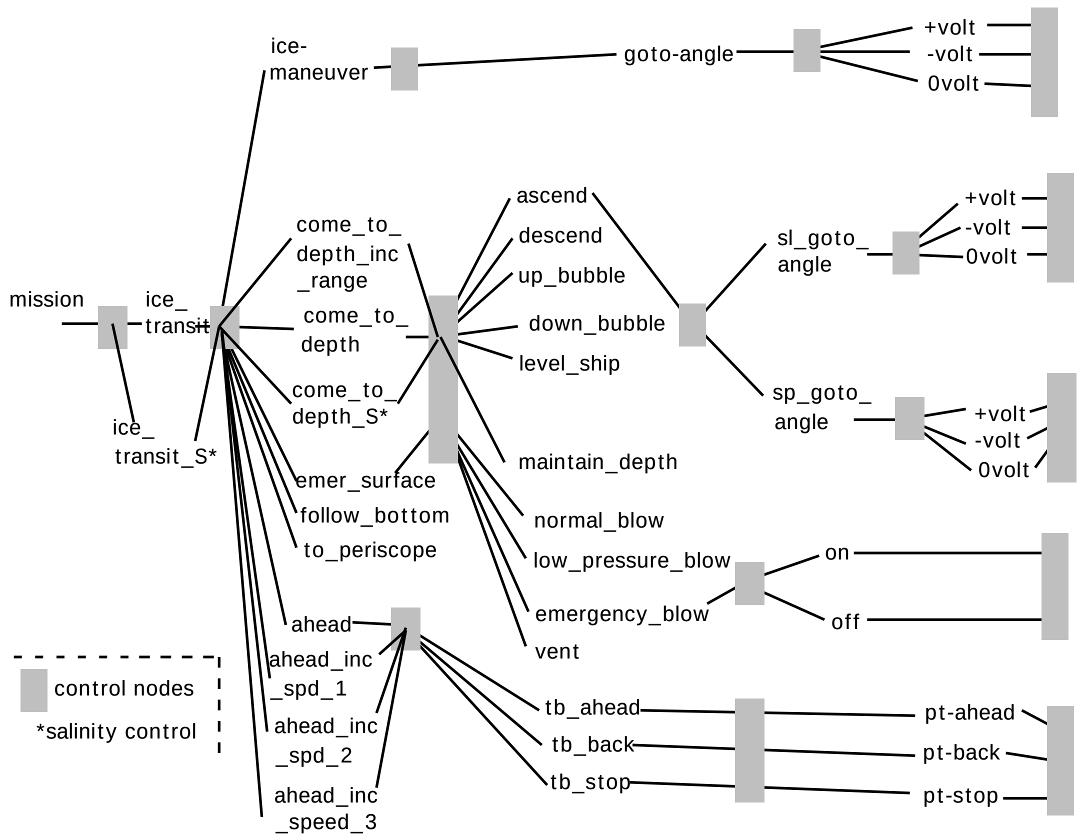


Figure 16: Submarine Automation Task Tree for Maneuvering and Iceberg Avoidance Scenario

4.3 Illustrative Behavior Descriptions

Behaviors for all the tasks shown in the task trees have been developed. This section describes two of them in detail to illustrate the development effort.

4.3.1 Command Level

From Figure 17, the Run_Mission command is decomposed into Prep_emergency_Vent, Submerged_Transit, Submerged_Vent, and Emergency_Vent commands. When the mission command is received, the command controller (CC) enters the state (S1). The submarine transits toward the next waypoint with CC ordering the ship's maneuver (SM) controller to execute the Submerged_Transit command and the Engineering system Controller (EC) to execute the Submerged_Vent command for normal open sea operations, see Figure 15. CC is in the state (S2) waiting for the execution status coming back from SM and EC. (S1) and (S2) describe a feedback control loop for CC under normal conditions. Once all the waypoints are reached, the submarine completes its mission and CC would be in the (done) state.

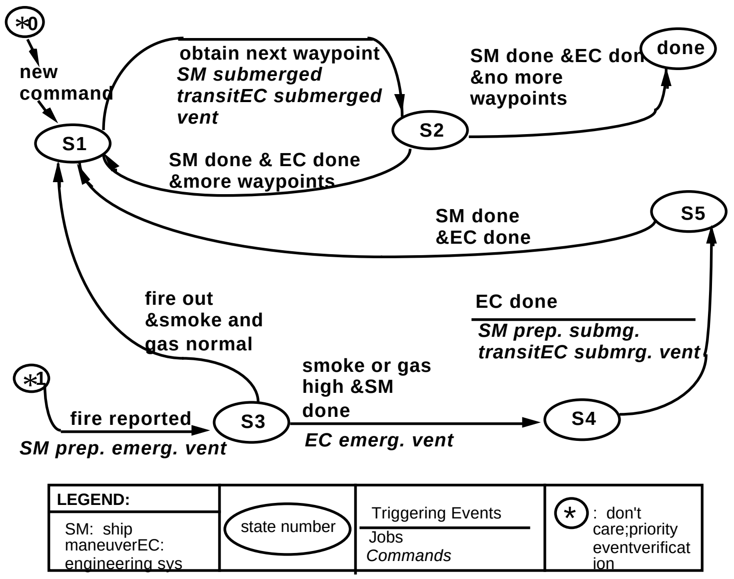

*0
new command
S1
obtain next waypoint
SM submerged transitEC submerged vent
S2
SM done &EC done &no more waypoints
done
SM done & EC done &more waypoints
SM done &EC done
S5
fire out &smoke and gas normal
EC done
SM prep. submg. transitEC submrg. vent
*1
fire reported
SM prep. emerg. vent
S3
smoke or gas high &SM done
EC emerg. vent
S4
LEGEND:
SM: ship maneuverEC: engineering sys
state number
Triggering Events
Jobs
Commands
*
: don't care;priority eventverification

Figure 17: A Mission Plan for the Command Controller

When a fire is reported, CC changes the normal behavior by ordering SM to Prep_emergency_Vent, which includes activities such as Clear_baffles by SM. Once SM is done, CC enters (S3) and orders EC to perform emergency ventilation. EC reconfigures the ventilation system (called line-up in submarine terms) and uses the diesel engine to snorkel. Once the contaminants are vented and the atmosphere is safe for breathing, CC enters (S4) and orders SM and EC to prepare to resume the normal open sea transit, (S1) and (S2).

The Run_Mission task has been explicitly decomposed and described using this state diagram, Figure 17. The same process is applied to each command on the task tree. This plan defines the initial state, the goal state, and all the intermediate states for the command controller during the execution of this command. In addition, all the required data, computation jobs, operator input, and subordinate status requirements are identified. This information enables the development of the associated sensory processing, world models, and human interface processes.

4.3.2 Helm Control and Illustrative Result

The Helm controller is solely responsible for the ship heading control. It receives the ICE_MANEUVER command and a next goal position from its superior, from which a desired course is computed (*0 and Evt0 in Figure 18. The required rudder angles are computed based on the course control error. If no obstacle (mainly ice keels) exists, the Helm controller will send the computed rudder angles to its subordinate, the Rudder controller, to approach the goal position (Evt3, Evt 4, and Evt5 in the figure).

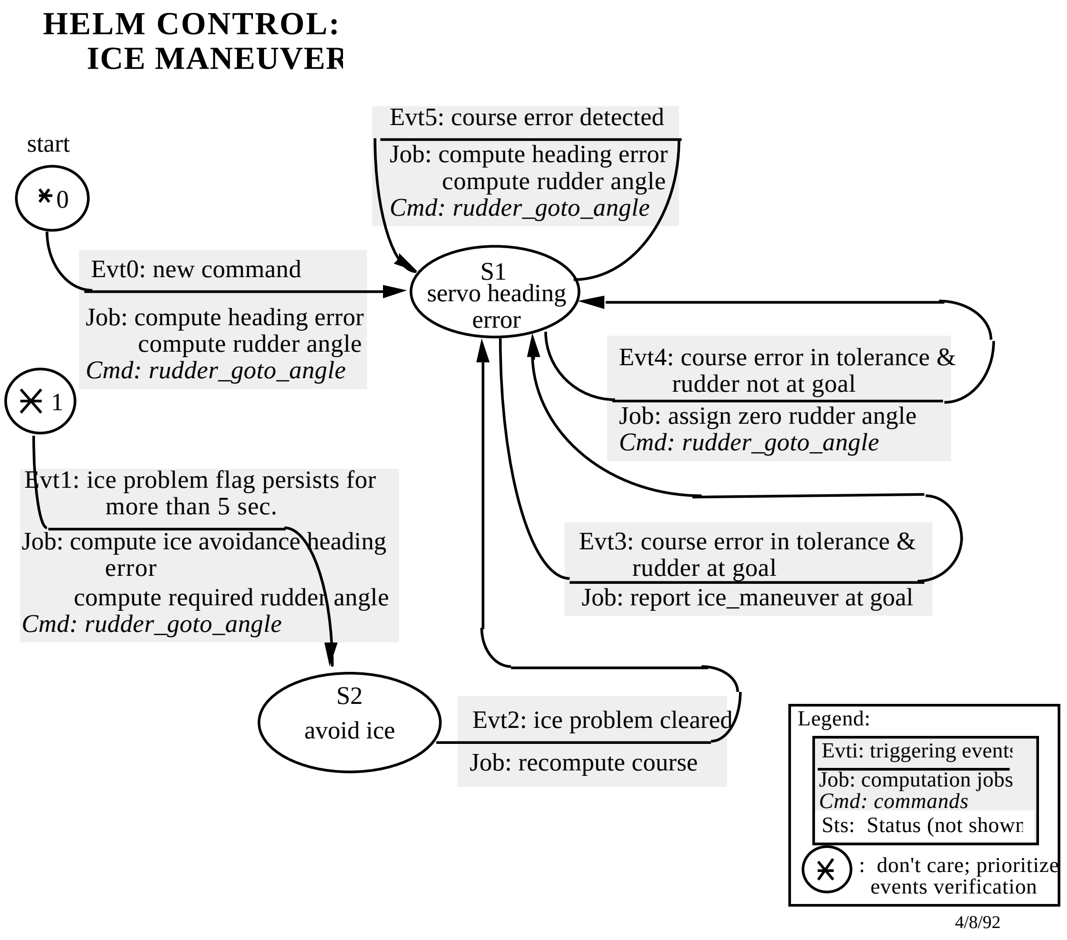


Figure 18: The Helm Control Plan

In this example, the Helm controller employs a CMAC (Cerebellar Model Articulation Controller) sensory processing algorithm [42] for evaluating the existence of the ice problem. Sonar data contain ice obstacle detection. The CMAC receives the data, generalizes for an estimated ice distribution map, stores the map in the control world model, and computes an ice avoidance recommended heading by taking into account the desired course. Inside the Helm controller, an ice_problem flag is raised if the CMAC recommendation differs from the desired course (Evt1 in the figure). The persistence of the ice problem for a predefined period of time warrants the specification of a “don't care state (*1).” The desired course will be omitted temporarily and the Rudder controller will be given an ice avoiding rudder angle (Evt1). A new course toward the original goal must be computed after the ice has been avoided (Evt2). Under the no-ice situations, the CMAC recommendations will be consistent with the desired course.

The computation of the desired rudder angles depends on the heading error. As the error reduces to be within some pre-calculated tolerance, a zero angle command must be sent in advance (before the heading error reaches zero) to account for the inertia of the submarine motion. An "ice_maneuver_at_goal" status would be reported (Evt3) to Ship Maneuver once Helm and Rudder are within the specified tolerances. The Helm controller continues in state S1, monitoring any possible heading deviation or the occurrence of the ice problem.

The Rudder controller always receives a GOTO_ANGLE command together with a target angular value resulting in a typical loop implementation. Simulated +VOLT, -VOLT, and 0VOLT signals are sent to the rudder simulator to generate the corresponding rudder angles.

As described in [1], one important feature a designer may discover as he steps from the higher levels down to the lower levels in an RCS hierarchy is the transition of coordinate systems from global systems with coarse resolutions to local systems with finer resolutions. The Helm controller refers to headings, a measurement global to the world, whereas Rudder refers to rudder angles, local to the submarine center line. The output of the Rudder controller, the voltage signals, is local to the electro-mechanical rudder mechanism.

Figure 19 illustrates a simulated execution of a scenario. The 637 class submarine transits through a section of Bering Strait that has ice problems. The planned path, shown as the dark line segments, is generated off-line and given to the controller prior to the transit. The islands, land, and Article Circle shown in the figure are, as such, not directly within the scope of this helm control illustration. The ice keels are generated randomly, in their sizes and shapes, and continuously (described also in section 4.6.5) throughout the paths. The actual paths of the submarine are shown in the light lines, which are jagged due to the local ice-avoidance activities. The box around the submarine is called its moving haven, meaning that the submarine is allowed to stay anywhere within the given area, but not outside it.

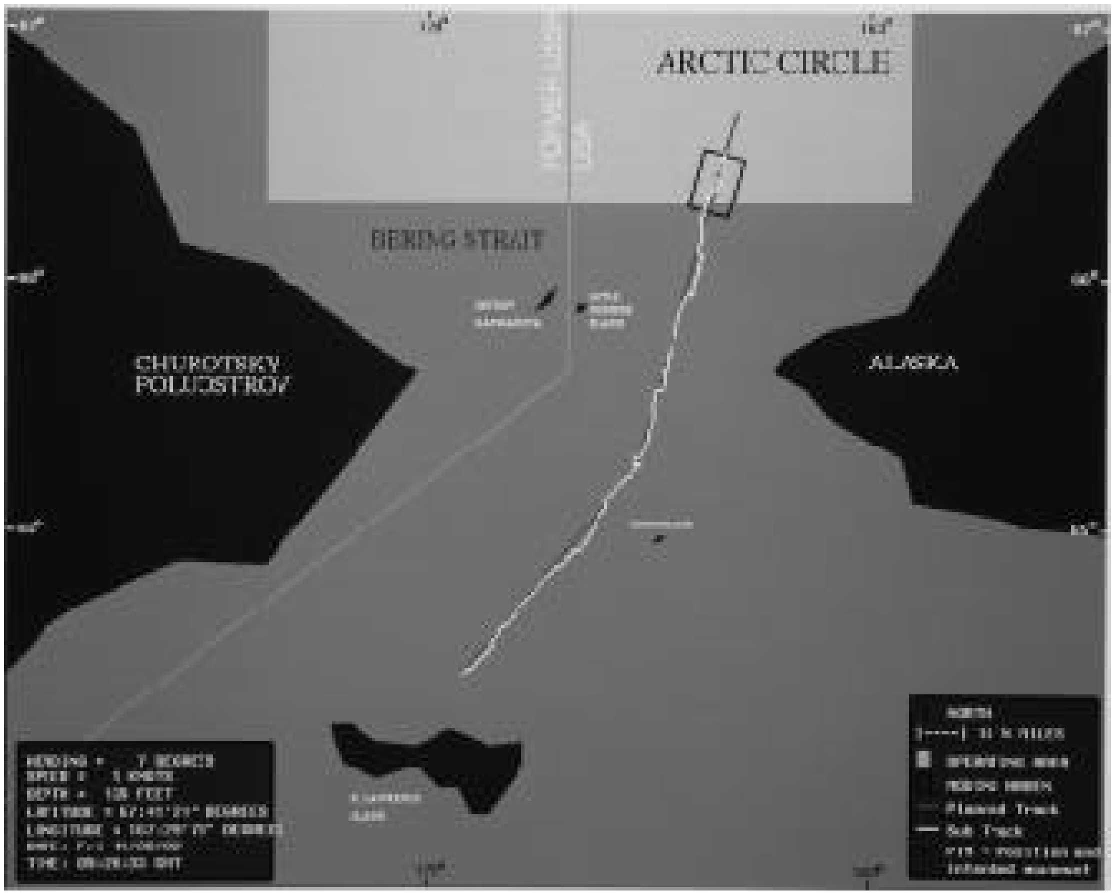


Figure 19: Resulting Path Control in Simulated Environment

4.4 Software Structure

This implementation employed a software structure as shown in Figure 20. Later sections describe each hierarchy, including its development--how templates are applied--and capability.

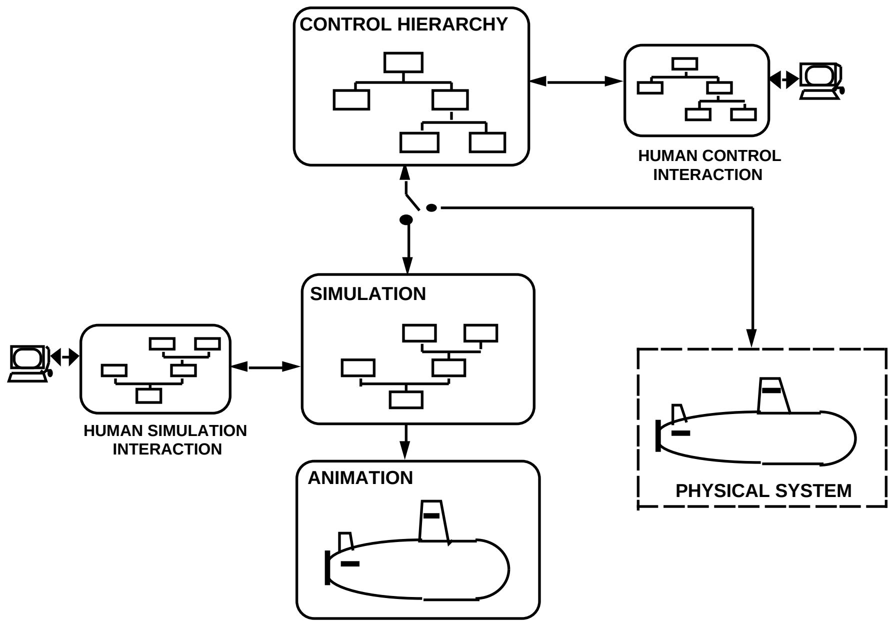


Figure 20: Comprehensive Software Structure

* The RCS Control Hierarchy (upper left in Figure 20): Executes the plans to perform real-time automatic control of ship maneuvering. Section 4.2 described the specific nodes. The finite-state-machine-based process template is used to implement the control nodes. Each node receives a command from its superior and selects an appropriate state table-based plan for execution.

* The Human Interface Hierarchy to RCS (upper right in the figure): Serves as the interface between the RCS hierarchy and various kinds of human operators who access system data and perform interactive control and system maintenance.

* The Simulation Hierarchy: Receives actuator commands and computes ship dynamics. Environmental objects and phenomena of concern are also simulated.

* The Human Interface Hierarchy to the Simulator (lower left in the figure): Allows the operator to change the simulation (including ship or environmental) setup to test the responsiveness of the RCS ship maneuvering system. System servicers may utilize this interface for maintenance and upgrade.

* The Animator (the lowest segment in the figure): Paints pictures in real-time on a graphic display based on the data computed in the simulation software.

The other four hierarchies all use the same format as the controller template. They all execute via the command and status paradigm. The entire RCS system results in five parallel sets of concurrent hierarchical execution.

Two distinct world models are used, one for the control system and one for the simulation, respectively. Each contains a set of state variables, dynamic models, and/or geometry models to represent the world.

4.5 Finite State Machine Based Process template

The authors applied the base-class, process template to implement the controller nodes. Behaviors were embedded in the decision process of the template in the form of state tables, as shown in Figure 21. The control hierarchy executes as a set of finite state machines.

The base-class, process template was used in a specific way to form the finite state machine execution model, as the following describes:

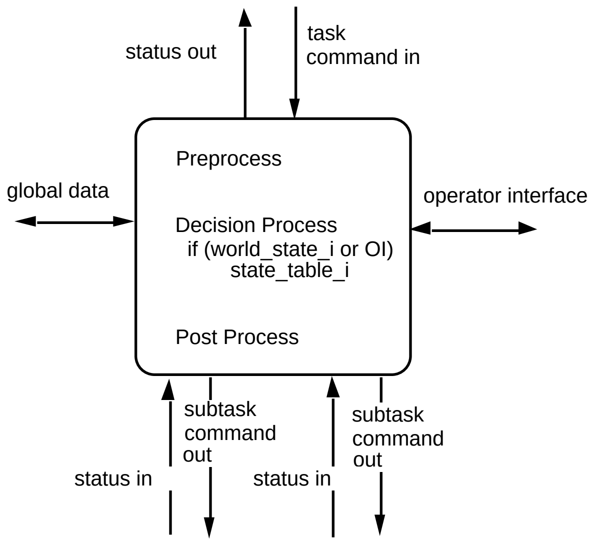


Figure 21: Finite State Machine Based Process template

· Preprocessing:

* Read task command input from the superior. Read operator requests. Read in the globally shared data required for decision process. Read status reports from the subordinates.

* Compute state values or evaluate event triggers from the following input data sets: environmental states, previous states for this node, states of other coordinating nodes, and required thresholds.

· Decision process:

Select a new task plan (state graph or table) if necessary and make a next state decision based on the latest task command input from the superior, operator requests, current world state, and subordinate execution status.

· Post processing:

Write sub-task commands to subordinates. Write to operator if necessary. Post WM data. Post status reports to superior. Collect performance data.

All the controllers in the control hierarchy are implemented using this template.

4.6 Simulation: Template and Implementation

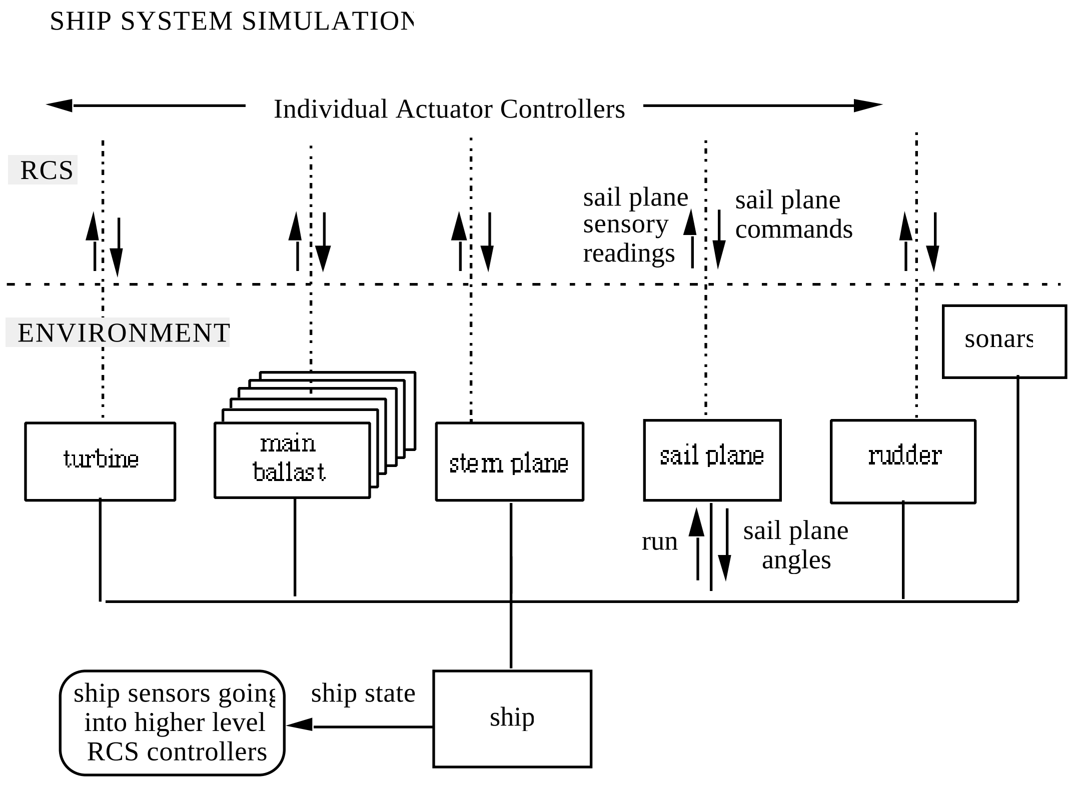


Figure 22: The simulation hierarchy

4.6.1 The Structure

The simulator hierarchy is shown in Figure 22. This hierarchy is developed to fit directly in the bottom of the RCS control hierarchy, Figure 14. The RCS actuator controllers send commands to their respective simulators. The lowest-level simulators compute the actuator movements accordingly and send the computed values back to the original controllers via the simulated sensors. The same values also are used by the ship simulator (i.e., the values are copied to the simulation world model) for the computation of the ship dynamics. The simulated ship state is fed back to the required higher level RCS controllers via another set of simulated sensors.

4.6.2 Sensor

The sonar simulator sends the data to the Helm control sensory processing routines. The submarine is equipped with 14 forward looking sonars. Sonar data are simulated by extending 14 lines, each simulating a sonar pinging in its fixed direction relative to the ship's center line, from the ship to the full sonar operating range (approximately 900 meters or 3000 feet). Intercepts with the simulated ice-keels are collected as sonar detections.

4.6.3 Actuators

The template for the ship system simulator, shown in Figure 23, assumes the same format as the generic base-class, process template. During execution, the simulator processes copy the commands from the actuators stored in the global memory and the current actuator positions values stored in the simulation world model. The simulators then compute new actuator position values according to the incoming commands. At the post-processing phase, the simulators copy the values back to both the simulator world-model and the global memory interface buffers. This information is consumed by the sensory processing function of the control hierarchy and by the high level simulation.

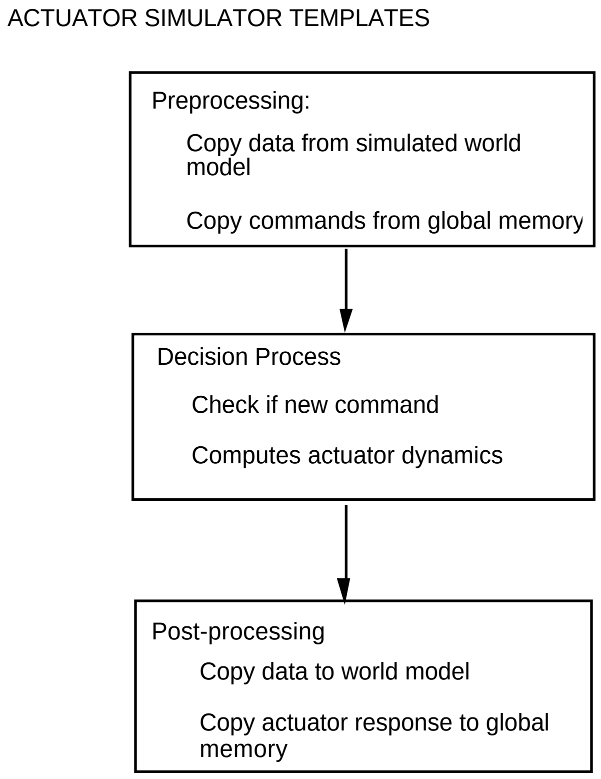

ACTUATOR SIMULATOR TEMPLATES
Preprocessing:
Copy data from simulated world model
Copy commands from global memory
Decision Process
Check if new command
Computes actuator dynamics
Post-processing
Copy data to world model
Copy actuator response to global memory

Figure 23: Actuator Simulator Template

These simulated actuators do not perform any closed loop control function but their corresponding controllers do. The desired actuator positions are used by the lowest level RCS controllers. The actuators normally receive and respond to commands such as ON, OFF, +VOLT, -VOLT, 0VOLT, etc. Dynamic models typically exist in the simulators to translate those commands to actuator positions.

4.6.4 Physical System

The template for the ship system simulator, shown in Figure 24, assumes the same format as the base-class, process template. In execution, this simulator module first copies in its incoming “commands,” an initialization or execution. The module then copies in all the data required for simulation, including the current ship state and the computed actuator positions.

The decision processing essentially involves a sequential execution of all the actuator simulators and a computation of the ship dynamics. In this example, the ship dynamics computes the position, speed, depth, bubble angle, and heading for the ship. There are two kinds of ship-depth simulation, namely, the actual depth and the depth that the ship-depth sensor reads. The former is regarded as the environmental simulation, whereas the latter is re-garded as the ship simulation.

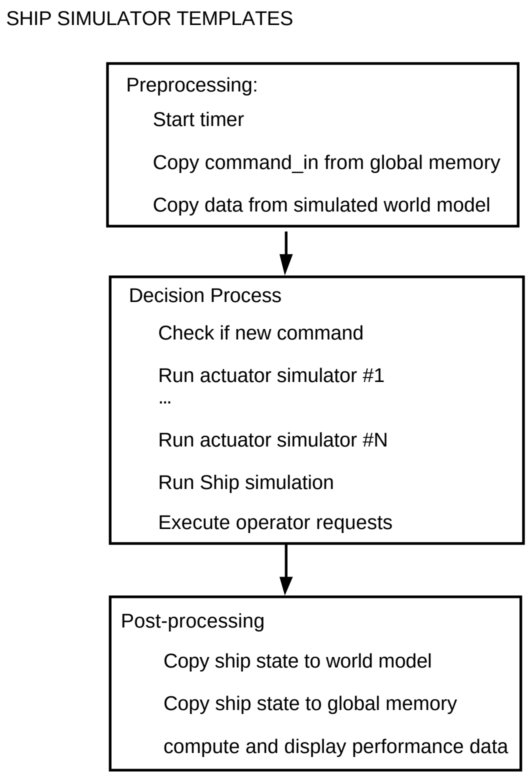

SHIP SIMULATOR TEMPLATES
Preprocessing:
Start timer
Copy command_in from global memory
Copy data from simulated world model
Decision Process
Check if new command
Run actuator simulator #1
...
Run actuator simulator #N
Run Ship simulation
Execute operator requests
Post-processing
Copy ship state to world model
Copy ship state to global memory
compute and display performance data

Figure 24 The ship system simulator process template

4.6.5 Environmental

The environmental simulation contains ice keels, sea bottom and depth, and seawater density. The ice keel and sea bottom profiles were generated using fractals [43], which allowed a complex structure of indefinite size and extent without the need for long and complex mappings. They are generated prior to test runs. The seawater is simulated in terms of fresh water run-off's, the true ship depth, and the water density. Unlike the depth simulation in the ship system, in this environmental simulation the depth model is sensitive to water density changes. Section 4.9, Ani-mation, demonstrates the graphic effects.

4.7 Operator Interface

4.7.1 Watchstation Displays: Mapping Control Activities to the Operating Environment to Facilitate Operator Interaction

In the process of upgrading existent systems from manual-operation-dominant to automation and intelligent control, it is beneficial to integrate the technology in an incremental and user-friendly manner. It is beneficial for the operators to be able to relate the new technology to their current operations. Therefore, the authors propose that the logical OI structure must be designed to be compatible with the current submarine operating environment.

The researchers have developed three, watchstation (WS), graphic panels to provide the input and output capability for operators during real-time control. The three displays are shown in the three shaded areas in Figure 25. They correspond to both the layout of actual submarine operational compartments and the RCS control hierarchy.

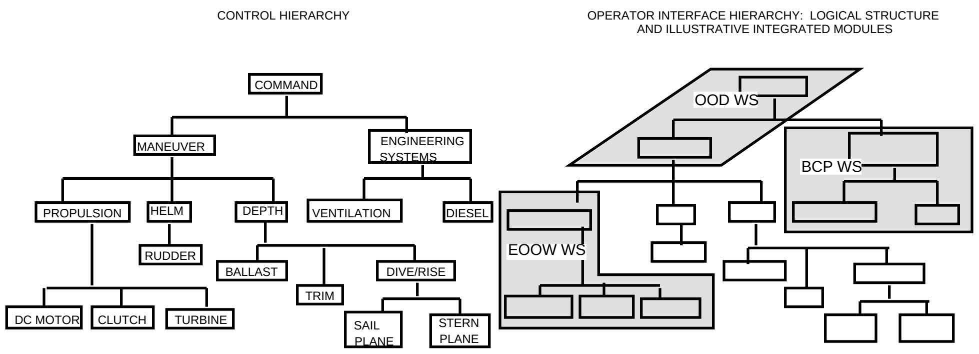


Figure 25: Mapping Control Activities to Operating Environment to Facilitate Operator Interaction

The three watchstation displays are:

1. The Officer of the Deck watchstation (OOD WS), which serves as the OI of the command and maneuvering controllers of the control hierarchy.
2. The Engineering Officer of the Watch watchstation (EOOW WS), which serves as the OI of the propulsion controller and all its subordinates.
3. The Ballast Control Panel watchstation (BCP WS), which serves as the OI of the engineering systems, ventilation, and diesel controllers.

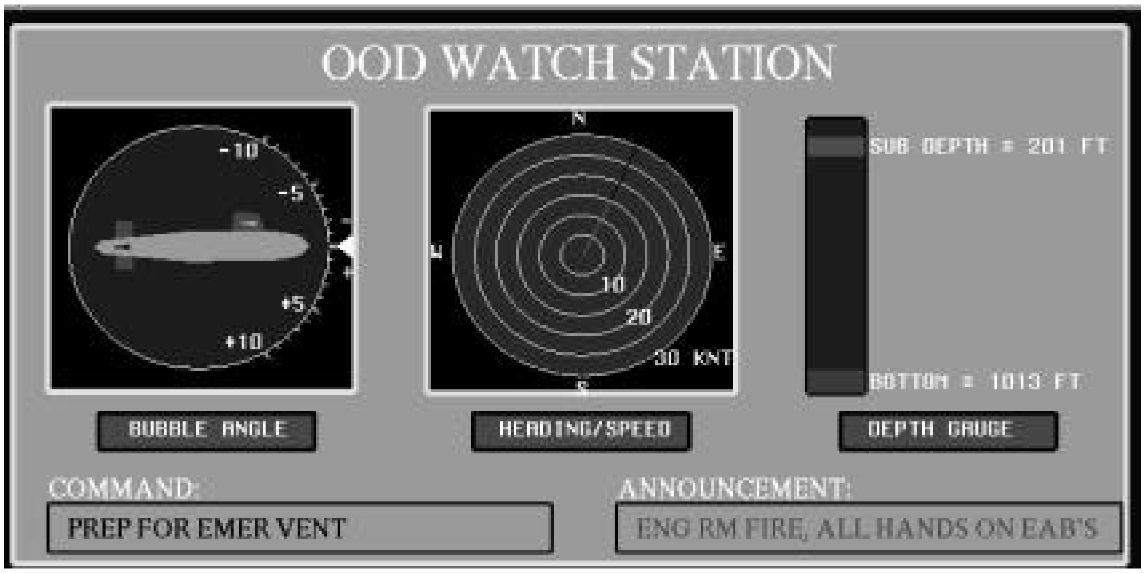


Figure 26 The Officer of the Deck Watchstation Display

The operator interface (OI) must display the necessary information for all the controllers, in real time, to enable the interaction between the control hierarchy and the submarine operators. Note that the objective of the OI is not to mimic the current submarine operating environment. Instead, the number and types of WS displays are determined by considering the following three factors: the operator work load, understandability and acceptability by the current submarine operation community, and the efficiency of hierarchical system control.

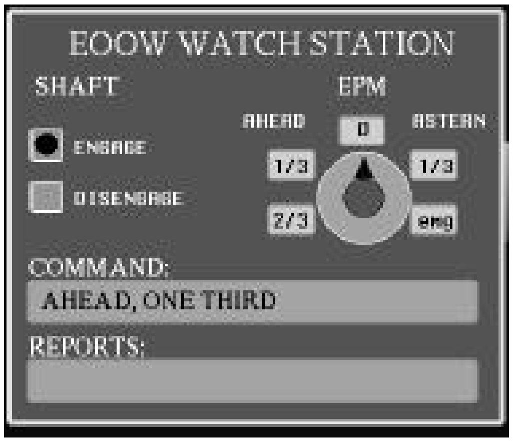


Figure 27: The Engineering Officer of the Watch Watchstation Display

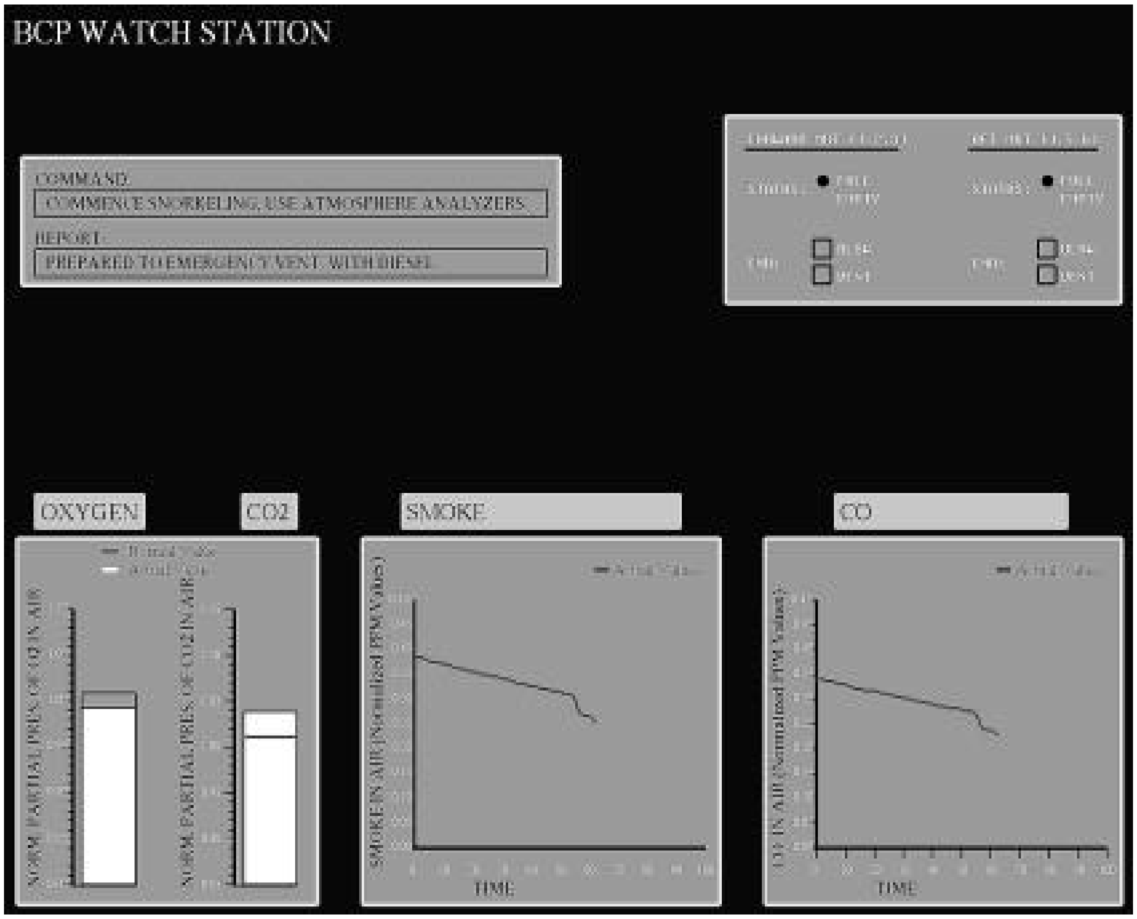
The OOD WS, illustrated in Figure 26, displays the crucial maneuvering data for the OOD, including (from left to right) the bubble angles, the heading and speed, and the depth. The WS also includes two, text-based, message areas: the left side showing the command that the command controller is outputting (for maneuver) and the right side showing the announcement that the controller is making. When a lubrication (lube) oil fire is reported through the commanding channels, the command controller immediately announces the message "ENGINE ROOM FIRE, ALL HANDS ON EABs (emergency air breathers)." Meanwhile, the COMMAND text area starts displaying "PREP FOR EMERGENCY VENT," meaning that the command controller is ordering the maneuver controller to execute the displayed command: to prepare for emergency ventilation.

The Engineering Officer of the Watch watchstation, shown in Figure 27, employs two buttons for engaging or disengaging the main shaft clutch. This WS also employs a speed control knob for the Emergency Propulsion Motor (EPM). In addition, this WS has two, text-based, message windows. The command-text window normally displays the command that the propulsion controller is executing. When the propulsion controller requests the operator to perform a command, this command appears in this window. Meanwhile, the window will shade in yellow. The REPORT message window displays useful messages for the Engineering Officer of the Watch operator. In Figure 27, EOOW must execute a ONE_THIRD_SPEED command, which is a subcommand of the “PREP FOR EMERGENCY VENT” command that Maneuver is executing.

The Ballast Control Panel watchstation, shown in Figure 28, contains the same types of text-message areas as in the Engineering Officer of the Watch watchstation. The Ballast Control Panel watchstation also includes: four atmospheric analyzer displays, the main ballast tank control buttons, and a ventilation line-up display.

At the beginning of the operation, the submarine is conducting an open sea transit. The Officer of the Deck watchstation displays a nominal zero degree bubble angle, a standard speed, and the nominal keel depth. The ANNOUNCEMENT message window is blank. At the Engineering Officer of the Watch watchstation, the COMMAND window displays a standard speed. Neither the SHAFT nor the EPM (Emergency Propulsion Motor) buttons are activated. The atmospheric analyzers in the Ballast Control Panel watchstation display normal levels of oxygen, carbon dioxide, smoke, and carbon monoxide. The ventilation diagram displays normal air circulation.

A lube oil fire is reported through the sensors in both the propulsion and the ventilation control systems. The REPORTS text window in Figure 26 displays the fire message. The command controller immediately announces the message of “ENGINE ROOM FIRE, ALL HANDS ON EABs” through the Officer of the Deck watchstation display. Meanwhile, the COMMAND window starts displaying “PREP FOR EMERGENCY VENT,” meaning that the command controller is ordering the maneuver controller to execute the displayed command. Maneuver decomposes this command into three commands: Clear_baffles, Slow_and_Change_Depth, and Shift_To_EPM for its subordinates, as seen in Figure 15. This task decomposition activity is displayed in the COMMAND window in real time. In other words, the displayed commands correspond to the actual states that the Maneuver controller is in. Meanwhile, the ventilation controller SP and WM algorithms update the abnormal concentrations of the modeled air constituents, namely, oxygen, carbon dioxide, smoke, and carbon monoxide. These data are displayed, in real time, in the Ballast Control Panel watchstation atmospheric analyzer displays. See Figure 28.


BCP WATCH STATION
OXYGEN
CO2
SMOKE
CO
NORM. PARTIAL PRES. OF O2 IN AIR
NORM. PARTIAL PRES. OF CO2 IN AIR
SMOKE IN AIR (Normalized PPM Values)
CO IN AIR (Normalized PPM Values)
Actual Value
TIME
TIME

COMMAND:
COMMENCE SNORKELING, USE ATMOSPHERE ANALYZERS

REPORT:
PREPARED TO EMERGENCY VENT. WITH DIESEL

Figure 28 The Ballast Control Panel Watchstation Display

The ventilation system is reconfigured automatically to prepare for emergency ventilation once the submarine is at the periscope depth. Once this command is completed, a message stating “Prepared to emergency ventilate with diesel” will be shown on the REPORT window on Figure 28. The Engineering Systems controller then receives a “Commence Snorkeling, Using Atmospheric Analyzers” command, as shown in Figure 28. The diesel engine extracts and exhausts the contaminated air and takes in the fresh air through the mast extending above the level of the water. This command completes when the atmosphere becomes safe to breathe again. At such point, the Command controller orders the submarine to resume the open sea transit.

4.7.2 Operator Interface Data Types

This project studied the data types developed for operator interactions with an objective of generalizing and standardizing the interfaces. The following summarizes the results:

* Input from the operators: Status, Command selections, Environmental variable settings, and Actuator override devices.

* Output to the operators: Continuous operational data, including ship depth displays and paths; Digital operational data, including ship speed displays; Discrete activities, such as commands, announcements, watchstation reports; Operator input requests; Schematic diagram; Errors and recommendations; Controller performance, such as execution time; and Debug data, including command/status, world data.

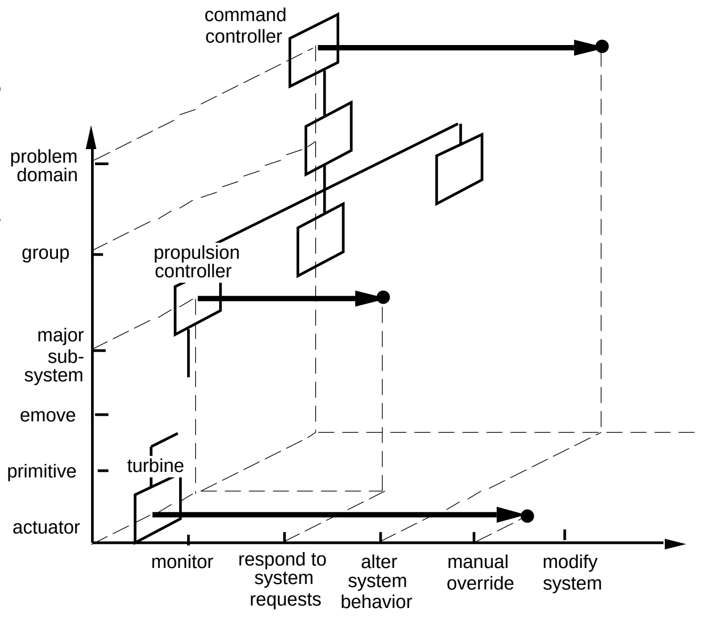


Figure 29: Different degrees of operator involvement

4.7.3 Levels of Operator Involvement During Operations

RCS allows operators to be involved in system operations at varying degrees. In this application, the researchers started experimenting with five degrees of “access control,” giving the operators from the least to the most amount of authority to interact with individual controllers in the hierarchy. This also provides a systematic model of conducting man-in-the-loop control and migrating legacy subsystems.

* Monitor, or to be informed.

* Respond to system requests.

* Alter system behavior by issuing new commands.

* Manual override.

* Modify system, or to reconfigure control hierarchy.

Operator access control should be applied to both the controllers and the tasks. Each controller in the hierarchy has a logical operator interaction function with assigned degrees of operator access control. Each step in a task plan may be assigned certain degrees of operator access control. For example, operator intervention may not be allowed during the execution of a safety related task. Our experiments focused on applying the operator interaction to controllers. Further research is required to integrate these two aspects.

The authors use a two-dimensional matrix to describe how the access control is applied to the hierarchy, as shown in Figure 29. Monitoring is the default degree of authority, which is available to all the operators.

The next unit on the horizontal axis, “respond to system requests,” does not allow operators to interrupt system operations. The system design does not allow the propulsion operator to issue new commands. Rather, the operator is only authorized to perform requested actions, such as CHANGE_TO_EPM, as shown in Figure 30. The operator, then, manually, disengages the main turbine shaft, engages the EPM shaft, and clicks the button on the display to report the completion. In this case, the operator serves as the EPM controller. This man-in-the-loop control operation also suggests that our method supports integrating the automation technology with legacy systems with manual operations.

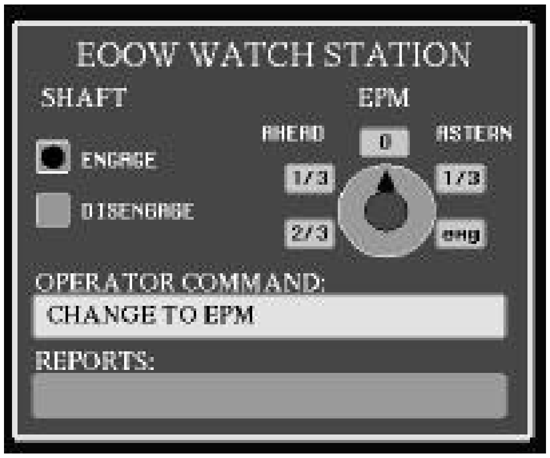


Figure 30 Request for Operator Action

Another example of “responding to system requests” is for operators to make decisions for the control system. The submarine may run into some salinity disturbances close to the coast where the depth controller can not maintain the ship's depth. The error messages and a set of options that involve either Sail/Stern Planes or ballast tanks are displayed for Maneuver. Blowing MBTs may be the last resort that disallows the submarine to re-submerge while adjusting the Planes are the normally selected option. The operator selects a best command for the controller to perform.

The next unit on the horizontal axis, "alter system behavior," means that the operator initiates graceful changes to system operations. In other words, operator commands are integrated with the ongoing system automatic operation. The command controller operator can alter system behavior by issuing a new mission command and having both the new and the existing mission commands scheduled and executed.

The next unit on the horizontal axis, "manual override," means that the operator takes over the control and the automatic control commands are ignored, suspended, terminated, or aborted. The lowest level operators can block the automatic commands and directly manipulate the actuators. When the submarine is about to hit ice keels, the helm operator must be allowed to enter an emergency turning command.

Reconfiguring the control hierarchy means that the operator can re-align the command chain of the hierarchy or can expand or reduce the control capability of the controllers. This advanced capability was not implemented.

4.7.4 User Expertise and Hierarchical Situation Perception

As stated in our basic principles, operator interface information must meet the user requirements and expertise [44]. All users are unique. They require different types of data. In one respect, data are processed and assimilated according to the levels of the operator command authority.

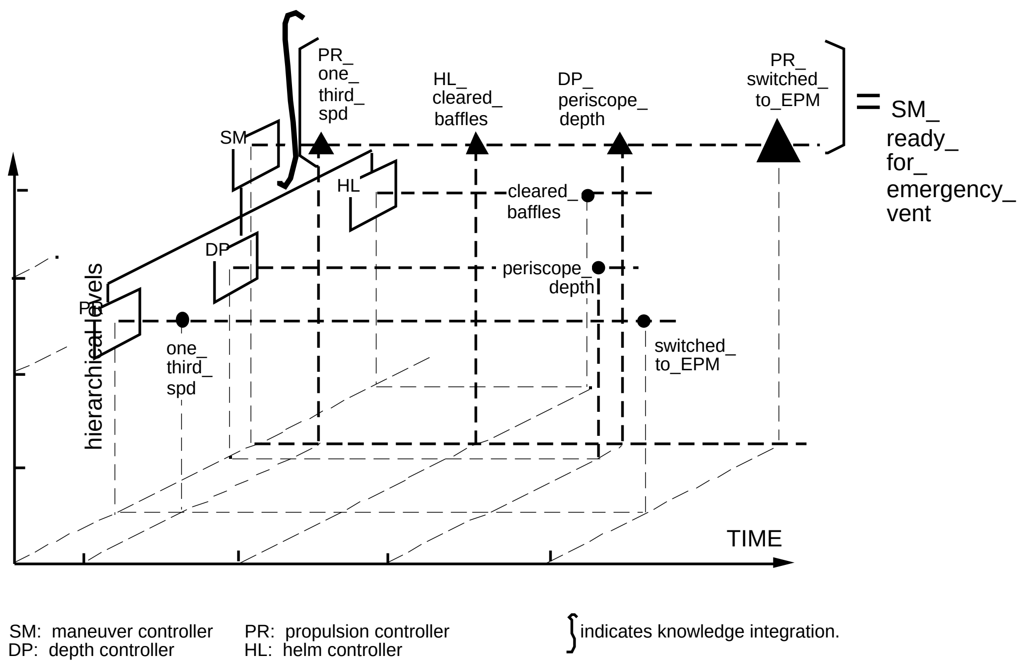


Figure 31: Hierarchical Situation Evaluation

In the RCS hierarchical environment, different levels deal with events and commands at different resolutions and time scales. Temporal and spatial integration must be performed when lower level information, either discrete events or continuous data, is delivered to higher levels. This integration process creates a different perception. Operators only need to understand the languages describing the situations at two levels of resolution: their own level and the level below.

Figure 31 illustrates this perception process. At the high control level, Maneuver is integrating the information received from the three spatially distinct subordinates, propulsion, depth, and helm. At the first instance, propulsion reports that the submarine has achieved the required speed. At the second instance, helm reports that the areas of concern have been cleared of hostile objects. At the following instance, depth reports that the submarine has reached the required depth. Next, propulsion reports that the emergency motor has been engaged. Maneuver summarizes all the information and concludes that it is ready to perform emergency ventilation.

4.7.5 Supporting Other System Functions

The focus of this example was on the operators' roles in system control. There are, however, additional types of operators that require control system support, including supporting systems development and service, simulation, and training. A system servicer may prefer to review raw data, as opposed to view concise and processed data often preferred by a control operator.

4.8 Knowledge Inference

The data that are useful to a submarine control operator may or may not be either useful or sufficient to a system developer or servicer. A system developer or servicer may need to access computer, machine-level, raw data during the debugging periods. However, in emergency situations, a control operator may not have time to read through all raw data to find out the problems, she/he needs to view concise and processed data. This implementation makes the raw data available, while transforming them to different formats to suit the user needs.

Therefore, knowledge processing is a part of our RCS world modeling functions. The knowledge transformation process can cover several layers of abstractions:

* From raw data to concise data.
* From the raw or concise data to conventional terms that match the operator's expertise. (Figure 32)
* From any of the aforementioned data formats to vocabulary that is consistent with the levels of abstraction in RCS hierarchical control and perception levels.

We have used submarine terms in our vocabulary, including "Fire in Engine Room, all hands on EABs (emergency air breathing units)" and "All stop, shift to EPM (emergency propulsion motor)." A particular message for a control operator may be an inferenced result of many pieces of raw data.

Note that, although we have made distinctions among these types of data, they are not intended to be hard boundaries. Some concise and processed data may be just as informative to a debugger as to a control operator. These different types of data may also share common criteria. For example, these data should, whenever possible, be displayed graphically to help human dissemination, be displayed in real time to facilitate situation evaluation, and share a common knowledge base to improve consistency.

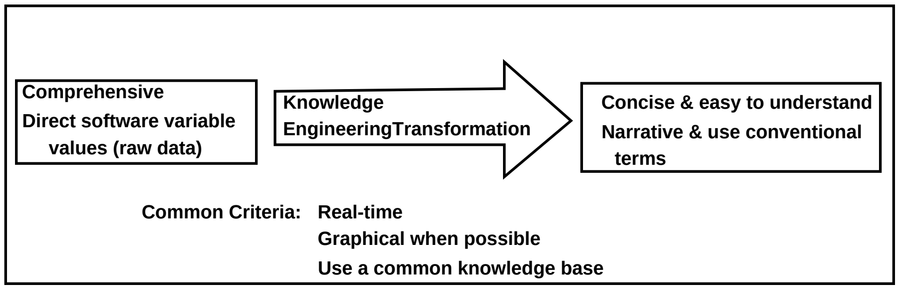
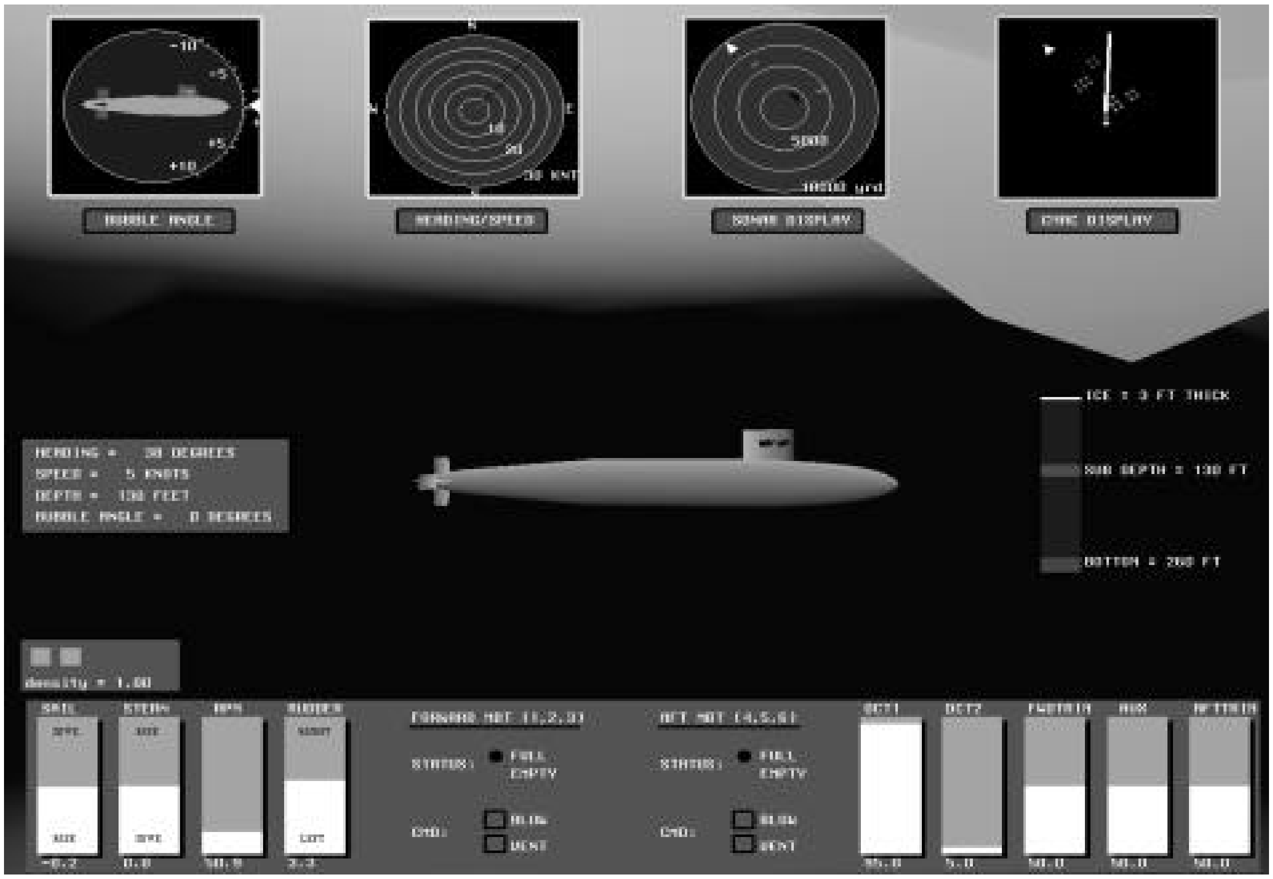

Comprehensive
Direct software variable values (raw data)
Knowledge EngineeringTransformation
Concise & easy to understand
Narrative & use conventional terms
Common Criteria: Real-time
Graphical when possible
Use a common knowledge base

Figure 32: Knowledge processing

4.9 Animation

Animation performs graphic rendering and allows the users to visualize the actions that the controller takes. Animation enhances the human understandability of a process. A human can comprehend many more variables pictorially than by reading numbers. Since RCS focuses on machines that do work, animating these machines is a logical step. It enables one to debug the algorithms that are used to control the machine and may avoid costly mistakes that occur when transferring software from simulated to real systems.

Figure 33 gives an overview of the demonstration computer screen including operator interface and animation. On the upper portion of the screen display, the ice keel is shown at the background. The foreground consists of four graphic displays: the bubble angles, speed and headings, current sonar detection, and ice map, from left to right. The middle portion of the screen display includes a digital, maneuvering-data display on the left and a depth display on the right. The submarine model was moving by the center of the screen when this snapshot was taken. On the lower portion of the screen display, a water-density simulation button is on the left corner, followed by the slider bars for the stern planes, sail planes, main turbine speed, and rudder, the control buttons and indicators for the main ballast tanks, and the slider bars for the depth control tanks, forward trim tank, auxiliary tank, and aft trim tank. All the displays are updated in real time.


HEADING = 30 DEGREES
SPEED = 5 KNOTS
DEPTH = 130 FEET
BUBBLE ANGLE = 0 DEGREES

ICE = 3 FT THICK
SUB DEPTH = 130 FT
BOTTOM = 260 FT

density = 1.00

Figure 33: Overview Display for Animation and Operator Interface

5. Case Study II, Manufacturing Domain, Toward Implementation Specification and on-line Behavior Generation

Various aspects of RCS have been applied to many manufacturing systems over the last two decades [16, 19, 46]. Under the SIMA program [12], a comprehensive reference model architecture called the Intelligent Systems Architecture for Manufacturing (ISAM) [26, 27] is being developed. ISAM intends to integrate RCS with important concepts from MSI [13] and QIA [14], as well as resolve architectural issues among them where differences exist. The objectives of ISAM include providing a framework for developing standards and performance measures for intelligent manufacturing systems and providing engineering guidelines for the design and implementation of intelligent control systems for a wide variety of manufacturing applications. ISAM is intended to anticipate the future needs of industry.

The implementation of a testbed to evaluate the applicability of the reference architecture is under way. The test-bed development effort leverages the work of several industry and government consortia. This ensures attention to current industry issues. These relationships are described briefly in the Approach section below.

These objectives and approach address the NIST Manufacturing Engineering Laboratory (MEL) goals of providing U.S. industry with state-of-the-art manufacturing architectures and models, fostering the development and implementation of advanced manufacturing systems, and anticipating and addressing the needs of U.S. manufacturing industry for the next generation of advanced manufacturing systems and standards.

5.1 Approach

The authors take the following approach in order to meet the project objectives:

* Specify processes and interfaces based on the ISAM reference model to facilitate architectural implementation and integration. This specification process will take into account industrial and other government agencies' efforts and results.
* Furnish a testbed to facilitate the testing.
* Leverage and integrate component technology, developed in house or outside, in the architectural framework.

The following elements describe the approach in detail:

5.1.1 Generic Shell for ISAM

An ISAM implementation contains many control nodes arranged in a hierarchy. Section 3.5.5 described a generic shell approach. In this project, the generic shell effort is to define a software structure, interface definitions, and task frame specifications that permit the core control node model to be standardized to the extent possible for manufacturing applications. As such, the authors are using the template based approach to examine and develop the software organization and development support for the Behavior Generation, World Modeling, Value Judgment and Sensor Processing components of a control node and their interfaces. A part of the effort is to tailor the RCS methodology to suit the manufacturing problem domain. This case intends to develop a shell structure that accommodates rich RCS capability, including sensory-interactive, real-time planning.

5.1.2 Application Domain: TestBed and Prototype System Design

The ISAM testbed is configured as a part of the NIST National Advanced Manufacturing Testbed (NAMT) testbed [28]. The NAMT is established to allow scientists and engineers from NIST, industry, other government agencies, and academia to work together to solve measurement and standards issues in information-based manufacturing. As a part of NIST's mission, NAMT also develops the needed tests and test methods for industry. Ultimately, the reference model architecture may be tested with the simulated and implemented factory components in the testbed.

NAMT has accommodated multiple sub-projects that address different manufacturing issues, and its scope is still expanding. A NIST Computer Integrated Manufacturing (CIM) Framework project [45] investigates a NAMT sub-area testbed that is described in a configuration shown in Figure 34. Many SIMA project elements are to be integrated in this CIM Framework testbed. An Inspection workstation and a Hexapod Controller are to be coordinated by a Shop controller. The communication follows a Common Object Request Broker Architecture (CORBA) client and server model. The shop controller can coordinate additional workstations when scenarios dictate, as the Turning Center and Simulated Workstation on the drawing indicate. The shop controller employs an operator interface function called a Guardian. Other features shown in the figure include a product database management system (PDM) and production information base (PIB), which are under investigation by other SIMA participants. Manufacturing activities including Design, Process and Inspection Planning, and Manufacturing Resource Planning (MRP) may produce information feeding the databases. An initial integration of PDM has been completed in that the shop controller has coordinated the PDM to provide certain inspection instructions to the inspection workstation.

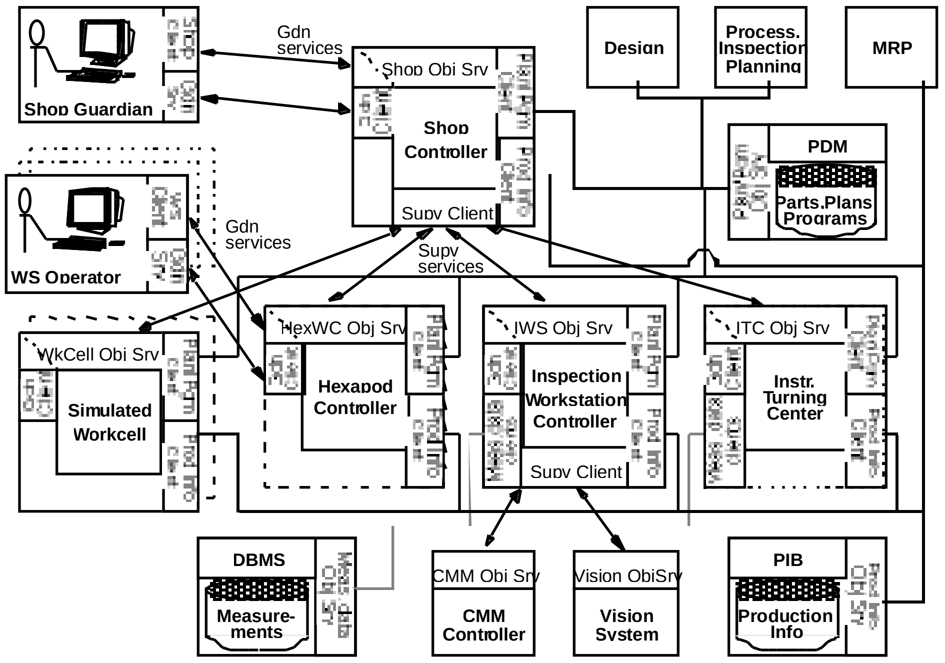


Figure 34: NAMT CIM testbed

As a starting point of ISAM experiments, the authors have applied the methodology to the chosen Inspection Workstation (IWS) that is based on a Coordinate Measuring Machine (CMM). Choice of this facility permits this project to take advantage of capabilities in advanced inspection currently being developed there under the Next Generation Inspection System (NGIS) [46] program. That project's use of vision- and scanning-probe technologies provides a sensory-rich environment for testing aspects of the architecture. Architectural and inspection (RCS, MSI, QIA, and NGIS) practitioners have joined their efforts in pursuing scenario sets for the activities of the IWS that can exercise the architecture. The first step is to generate a list of capabilities that will sufficiently exercise the architecture. These capabilities include on-line task planning, processing and handling errors, explicit quality loops, advanced human interaction, interface with legacy software and hardware, simulation, design, plan, and production data interface, administrative functions, learning, multiple simultaneous tasks per node, and interaction between CAD and sensed features.

Activities that required these capabilities are then included in the scenarios. The scenario identifies visible activities and data exchanges that reflect characteristics of typical inspection activities determined through various factory visits. A subset of the activities was used in the initial implementation. See the later section on project development. The scenario serves as an indispensable starting point in the implementation design. From this point, the researchers have been developing task descriptions and task trees, control node hierarchies and state representations for all tasks for each intended testbed mission. Part of this effort is the continued interaction and collaboration with software tool vendors that produce products that may be adapted to assist control system developers in this phase of implementation.

5.1.3 Leveraging Component and Interface Technologies

(1) EMC

The Enhanced Machine tool Controller (EMC) [16] project focuses on the development of implementation infrastructure. Our ISAM implementation heavily leverages their results, including the Neutral Manufacturing Language (NML) communication interfaces, an application programming interface (API) specification, a baseline development environment with a version control mechanism, Internet based operator interfaces, and advanced control system diagnostic capabilities.

(2) OMAC/TEAM Message Definition Efforts

The Open, Modular Architecture Controller (OMAC) users group [47] and the Department of Energy (DOE) Technologies Enabling Agile Manufacturing (TEAM) programs [48] have been working on Application Programming Interfaces (API) message definitions for manufacturing. These specifications support communication between control nodes, and, therefore, facilitate open architectures. A member of the ISAM Reference Model Architecture project team actively participates in that effort and applies and extends applicable results to the APIs used in the IWS testbed implementation.

(3) Planning

Planning at an ISAM node is conducted in the Behavior Generation module. An extensive study of planning/scheduling algorithms has been performed and performance experiments have been conducted. The current effort involves integrating the algorithms in the planner template, as described in section 5.2.3. Further work to develop standard interfaces to planners, plan simulators and evaluators, and schedulers is needed. These interfaces are to be tested in the IWS testbed. Issues in discrete event planning are also being examined.

(4) Feature-based Control

A focus on feature-based control involves applying part features described in solid models in STEP form to machining operations. Initial developments are being extended to the inspection domain.

(4) World Modeling/CAD

This effort seeks to formalize world modeling definitions within the generic process template and to design possible API access for world model information. Experiments are planned for an integration of product data and CAD API's with inspection controller code. This work will also include developing Knowledge Base entities of ISAM which include task frames, command frames, plans, algorithms and resources.

(6) Operator Interfaces

The operator interface work includes integrations of various operator interface technologies, including Java-based tools. This work also includes the analysis of operator interface specification requirements with respect to the ISAM generic shell, World Model, and MSI Guardian. These tools and requirements are being investigated by many SIMA activities and outside researchers, which this project plans to leverage.

5.2 Methodology Development: Generic Shell, Interface Definitions, and Task Execution

The base-class, process template and the experience gained from the first study case provide a solid foundation for the authors to take the step to develop the next generation process templates that provide richer RCS capabilities. However, before launching the development effort, the authors conducted a detailed investigation of the internal execution activity of an RCS control hierarchy to gain insights into how the template-based systems work.

5.2.1 An Internal Execution Model of a RCS Based Manufacturing Workstation

A detailed understanding of the internal execution activity of an RCS-based control hierarchy is desirable before designing such control systems. This analysis reveals design details such as the inter-module interaction requirements and modular synchronization requirements. The authors performed an analysis on an example manufacturing workstation. Time-lines are used to step through the scenarios and reveal the execution details. The result exemplifies an RCS execution model, as the following describes:

The control hierarchy consists of:

- WS (workstation) has two subordinates, R (robot) and M (milling machine). See Figure 35.
- WS contains control node contains an SC (scheduler) and EX (executor) pair for each of WS's subordinates. The other planning supporting functions such as simulation and value judgment functions are assumed.
- M employs a dual-station pallet and is able to switch among tasks.
- R has a single end-effector and performs one task at a time.

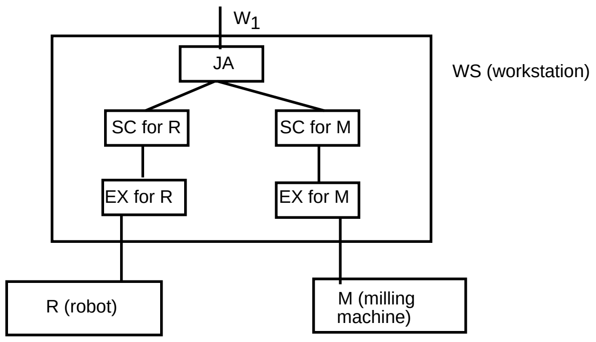


Figure 35: RCS Based Manufacturing Workstation Example

Scenario as described in the control system execution time-line, shown from the top down in Figure 36:

- WS receives a task W1, W1 = make_part_p1 (DL1). DL1 indicates deadline.
- WS-JA assigns scheduling jobs for SC-M and SC-R, the schedulers for the two subordinate equipment units.
- The two SCs coordinate, via communicating each's timing constratiints, to generate robot part fixturing schedules and milling machine machining schedules. The schedulers may retrieve process plans as a priori knowledge and schedule the plan steps. The outcome includes two schedules, schedule-R and schedule-M.
- The two EXs receive the schedules and execute the steps according to the time.
- As WS is executing the first step of W1, (W11), the second task, W2, arrives, W2 = make_part_p2 (DL2). The two schedules are to decompose W2 and the subtasks are interleaved within the original schedules based on the equipment availability. The results are two modified schedules, schedule-R2 and -M2.
- The two EXs continue executing the schedules one step at a time regardless of whether the schedules have been modified.
- If W2 has the highest priority, then an exception procedure may be devised to suspend or abort the original schedules associated with W1 and the scheduling and the execution of W2 takes place immediately.

The first important notion is that only one schedule is maintained for a subordinate controller. The superior controller may handle multiple tasks by interleaving the subtasks for the multiple tasks in one schedule to maximize the subordinate equipment utilization. As the time-line chart illustrates, the two workstation tasks, W1 and W2, may or may not complete in accordance to their arriving order.

The second is that the planning or scheduling process does not wait until the generated plan is completely executed. Rather, the planner (JA and SC) replans concurrently as the generated plan is being executed. The planner appends the generated schedules to the specified planning horizons. At each planning cycle, which may or may not be synchronized with the execution cycle, the planner evaluates the current world state and the future goal state and modifies the schedules when necessary.

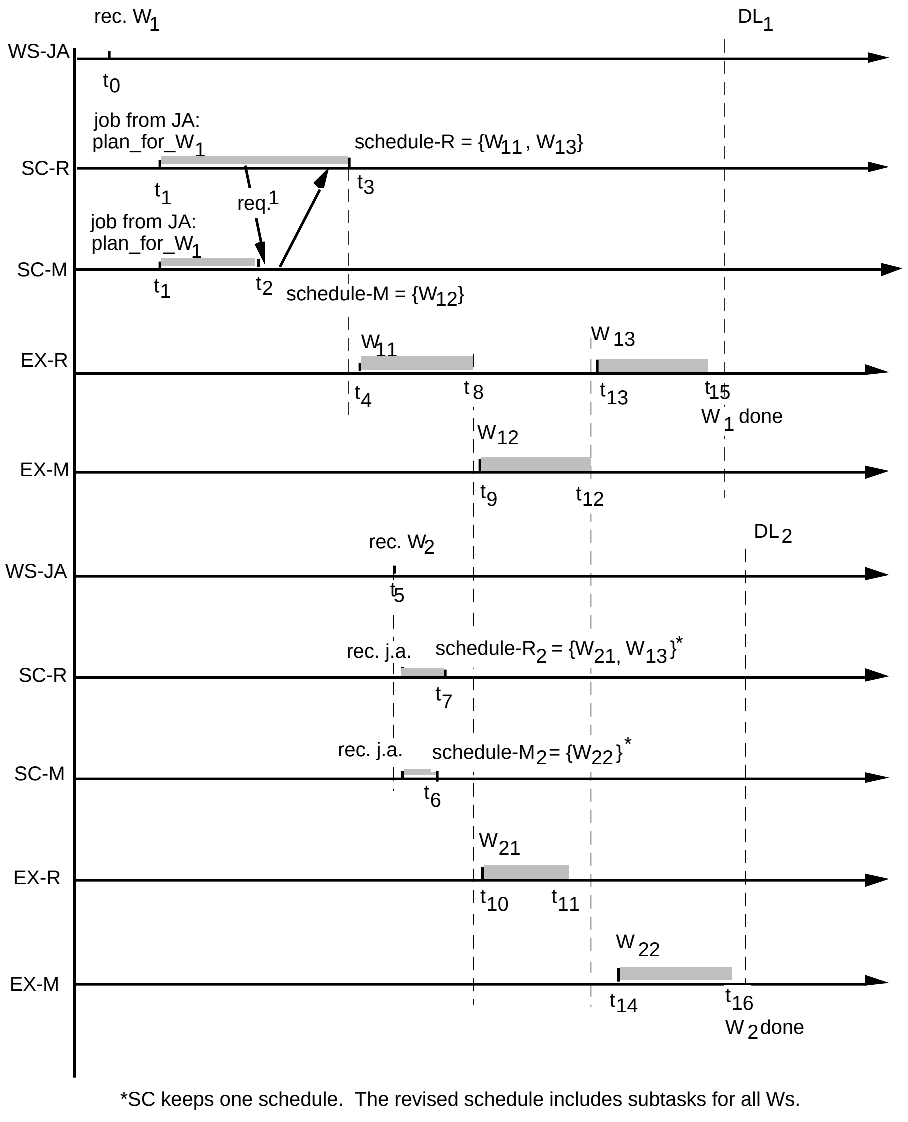


Figure 36: Time-Line Based RCS Hierarchy Internal Execution Model

5.2.2 The Executor Template

The base-class, process template is extended, using the object-oriented inheritance concept, to build the executor template. The EX template is further instantiated in the ISAM implementation as well as in other applications. This is a part of the RCS methodology effort to provide generic implementation tools and to standardize the processing and interfacing models.

Each Executor contains a plan buffer that is loaded via interfacing with the plan selector function in the planning process. The plan consists of two parts: 1) a trajectory of planned actions interleaved with 2) a corresponding trajectory of planned states (or subgoals).

In addition to the plan, each Executor also receives input in the form of estimated (or predicted) state feedback via Sensory Processing and World Modeling modules. Status to and from peer EX modules provide the basis for coordination between the EX modules. EX compares the difference between the estimated (or predicted) state feedback and the current planned subgoal and either performs feedback control to compensate for errors or invokes emergency actions when the errors are too large for the employed control laws.

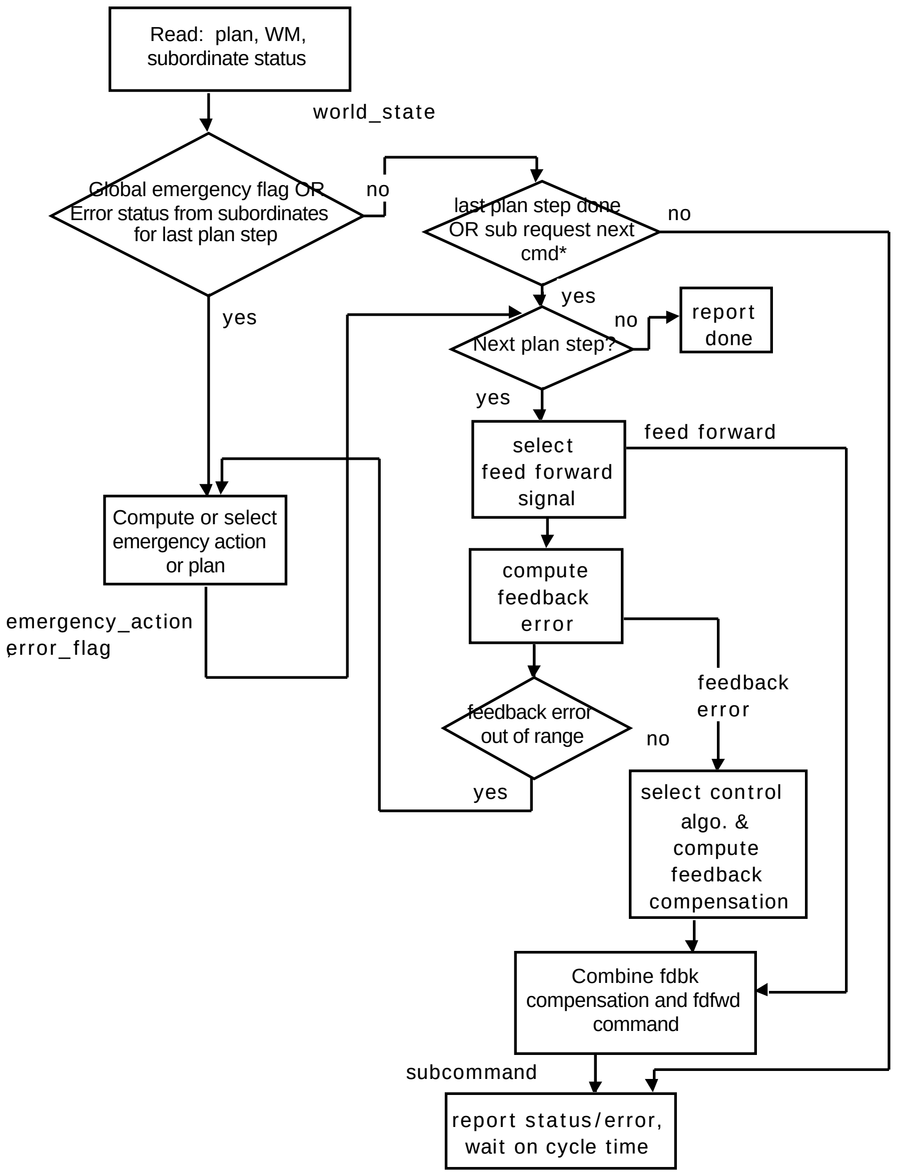

Read: plan, WM, subordinate status
world_state
Global emergency flag OR Error status from subordinates for last plan step
no
last plan step done OR sub request next cmd*
no
yes
yes
Next plan step?
no
report done
yes
select feed forward signal
feed forward
Compute or select emergency action or plan
compute feedback error
emergency_action
error_flag
feedback error out of range
feedback error
no
yes
select control algo. & compute feedback compensation
Combine fdbk compensation and fdfwd command
subcommand
report status/error, wait on cycle time

Figure 37: Internal Functional Flow of Executor Template

5.2.3 The Planner Template

The base-class, process template is extended, using the object-oriented inheritance concept, to build the planner template. Figure 38 provides a simplified view of the planning process as defined in [1]. In this simplification, the job assignment and scheduling functions are combined to reduce communication load. Note that, if any of these functions becomes computationally-intensive, they may be separated into individual computing processes.

Tentative schedules are generated through the node's plan horizon. Simulation and value judgment functions are called to determine the best schedule. Plan selection chooses the best plan and sends it through the plan buffer in the EX module. The EX may report errors caused by executing earlier generated schedules. The planner will correspondingly replan. If the planner has replanned a particular task for too many times (beyond the designed threshold), an error flag will be generated for the superior node.

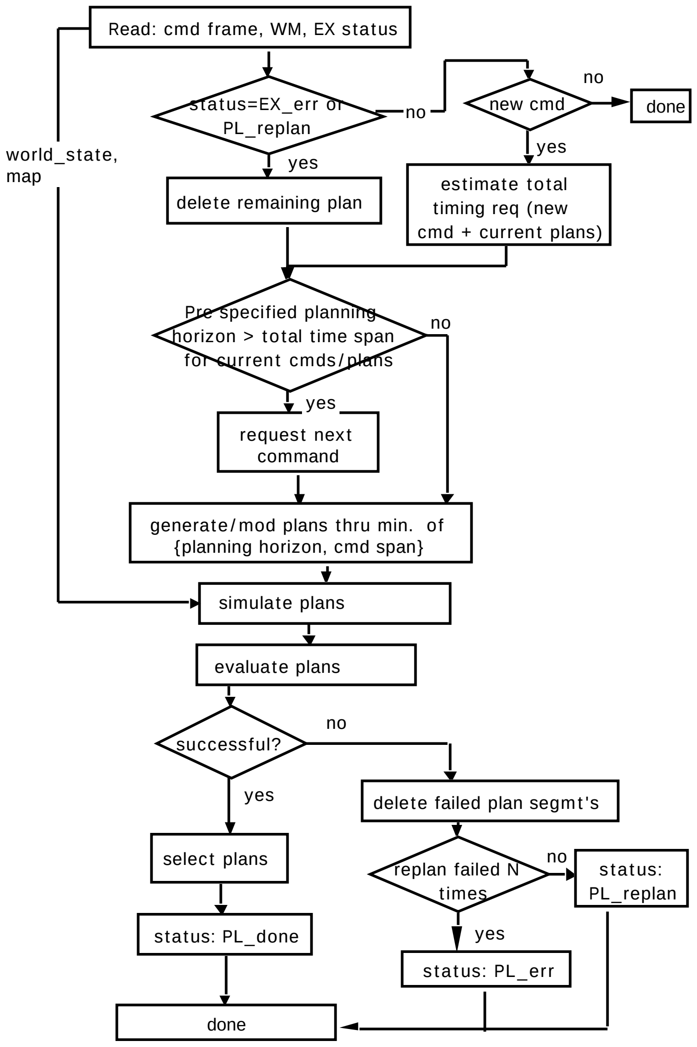

Read: cmd frame, WM, EX status
status=EX_err or PL_replan
no
new cmd
no
done
yes
yes
world_state, map
delete remaining plan
estimate total timing req (new cmd + current plans)
Pre specified planning horizon > total time span for current cmds/plans
no
yes
request next command
generate/mod plans thru min. of {planning horizon, cmd span}
simulate plans
evaluate plans
successful?
no
yes
delete failed plan segmt's
select plans
replan failed N times
no
status: PL_replan
status: PL_done
yes
status: PL_err
done

Figure 38: Internal Functional Flow of Planner Template

5.2.4 Interface Definitions

The developers describe the interfacing method with respect to the templates. The method covers both within a node and between the nodes. ISD has implemented generic command and status base classes as a part of the RCSLIB library [34]. Our interface data structures inherit and expand from these base classes.

(1) The PL/Sim/VJ/PS may send EX an “Execute (schedule)” request (a general sense of command). This request data structure contains a serial number, as inherited from the base class. The schedule can be implemented as pointers to state graphs, inspection plans, or any other manufacturing process schedules. The schedule can also be waypoint arrays of a navigation path for a material handling robot cart. The EX reports, via the information slots defined in the base status class, the serial number echo, the status, and state number. The PL/Sim/VJ/PS uses this report to adjust the planning activity: to abort and replan, continue updating the generated schedule, etc.

(2) The interface between EX and the subordinate PL/Sim/VJ/PS is implemented as a set of task commands and is application specific. The commands include “Goto (goal),” “Load_part,” and “Inspect_part (inspection program).” The data structures for the command frame inherit a serial number and may be appended with a goal, a pointer to a file describing part loading instructions, and methods to retrieve inspection plans from the databases, respectively. The EX may send more than one goal points to meet the subordinate’s planning-ahead requirements.

(3) The interface with SPWM is under investigation in the area of providing CAD data. Currently SPWM supplies the BG templates with state information.

5.3 Initial Application System: Inspection Workstation

5.3.1 Behavior Generation

In the NAMT Next Generation Inspection Workstation testbed [28], a system goal given to the workstation from the highest level, Shop, could be “Inspect_part,” as shown at the top of Figure 39. The Shop may retrieve the instructions from a product database management (PDM) system, which may require WS to interpret the inspection instructions. WS decomposes this command to “Place_part” and “Load_probe” for Measurement (MS) and Fixturing (FX). MS decomposes and issues a subcommand for Tooling to perform the probe loading process. FX and Tooling are performed by human operators at this stage. Therefore, displays are designed to show the instructions for the operators.

The performance of this goal can involve a series of sub-behaviors including a series of “Inspect_feature’s” that the workstation would assign the CMM equipment controller to perform and a series of corresponding “place/fixture_part’s” for the fixturing controller. The two subordinates need to coordinate. The performance of inspecting a specific feature can involve a series of even lower level “Go_to_point” sub-behaviors to be conducted by the inspection probe controller. These sub-behaviors are further decomposed until individual motor behavior is generated. This results in the actuation of the hardware components to achieve the goal within a given tolerance. The inspection data may be recorded, reported, and processed at multiple levels of hierarchy to form different perceptions of the inspection results.

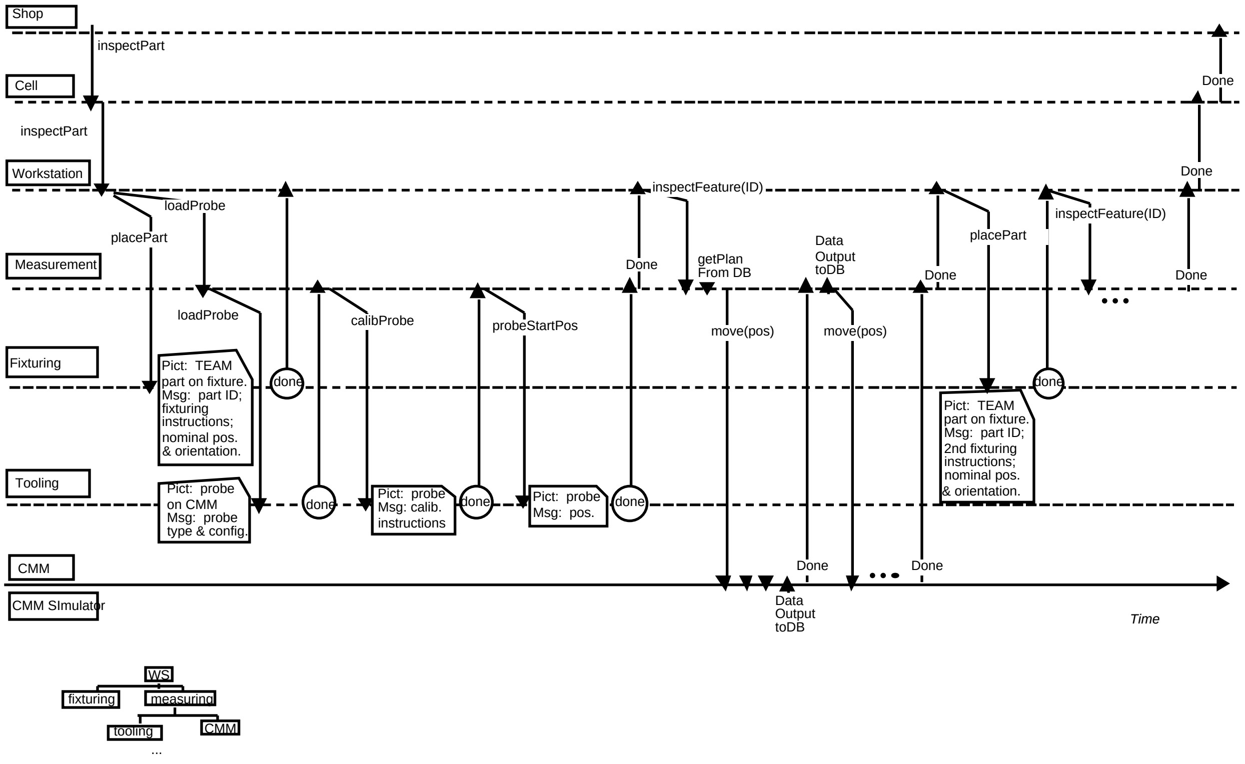


Figure 39: Inspection Behavior Analysis

Figure 40 further maps the behaviors onto the RCS functional modules.

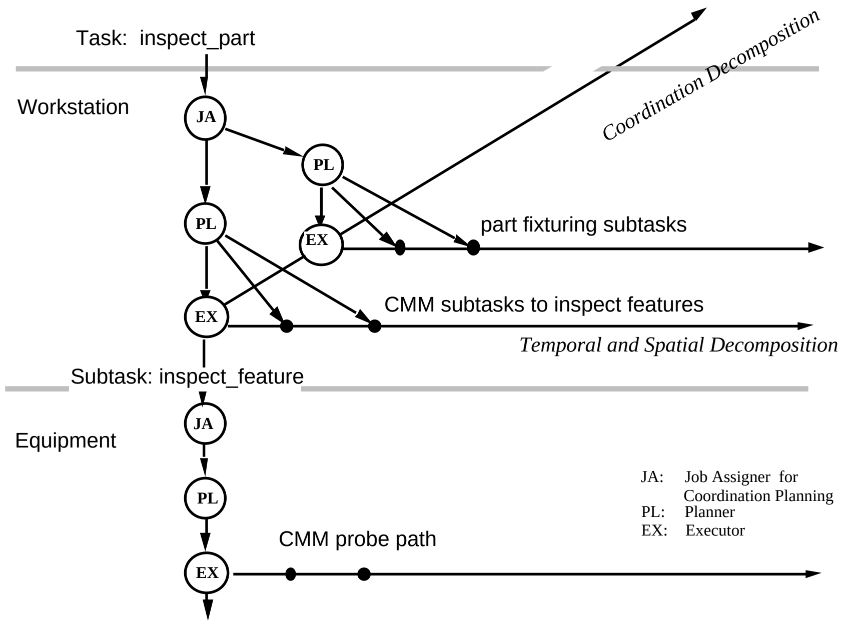


Figure 40: Mapping Behavior on RCS Modules

5.3.2 Inspection Workstation Control System Implementation

The authors applied the methodology to the Inspection Workstation (IWS) of the NAMT. Figure 41 shows the control hierarchy and the command interfaces. Figure 42 shows the internal functionality of some of the control nodes in detail. The Primitive (Prim) and Servo levels handle low level motion control. These levels and the Elemental move (Emove) level were adopted from the existent NGIS implementation [49] and are considered legacy subsystems.

The Emove level also contains a node for tool changing. The tool changing activities are performed by a human using an operator interface (OI) computer screen. The operator reads the commands and instructions on the screen, act on them, and reports the status, via the screen, to the control system to close the control loop.

The Task level contains control nodes to support the measurement subsystem and the fixturing subsystem of the workstation. These nodes each contain a PL and an EX process. PLs employ interpreters to interpret pre-generated plans which are coded in an industrial standard language called DMIS (Dimensional Measuring Interface Standard). This configuration demonstrates ISAM’s capability of adapting industrial practices into intelligent manufacturing. The Fixturing node handles loading, unloading, and refixturing parts. In this implementation, these are executed by a human.

The Workstation level control node accepts jobs from the Shop level and coordinates the activities of the Measurement and Fixturing nodes to carry out the job on a particular part lot. In the recently completed integration, the shop controller has coordinated the PDM to provide certain inspection instructions to the inspection workstation. The PL module for this control node interprets the workstation inspection plan and sends the decomposed fixturing instructions and part feature inspection schedules, as described in the previous behavior analysis section, for the two task level subordinates.

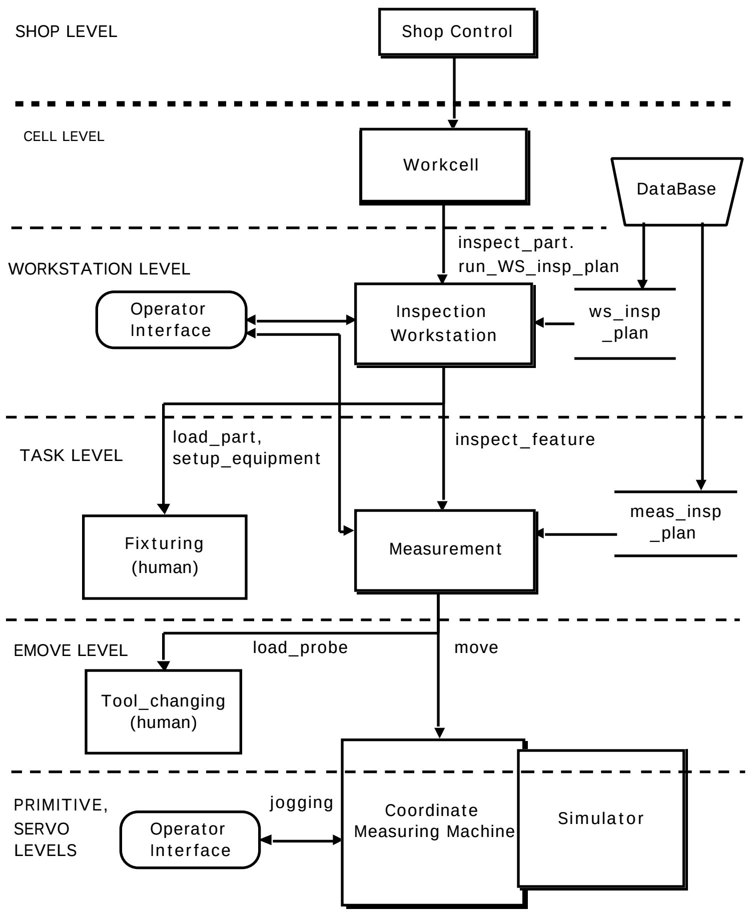


Figure 41: Inspection Workstation

A workcell node resides above the workstation node. This node converts and decomposes a highest level incoming job, implemented via a CORBA protocol, to the IWS native communication protocol, implemented via the EMC NML. This setup demonstrates ISAM's capability to integrate heterogeneous manufacturing subsystems.

The Shop level obtains orders from its operator interface, i.e., the Guardian, and plans the jobs for the subordinates accordingly.

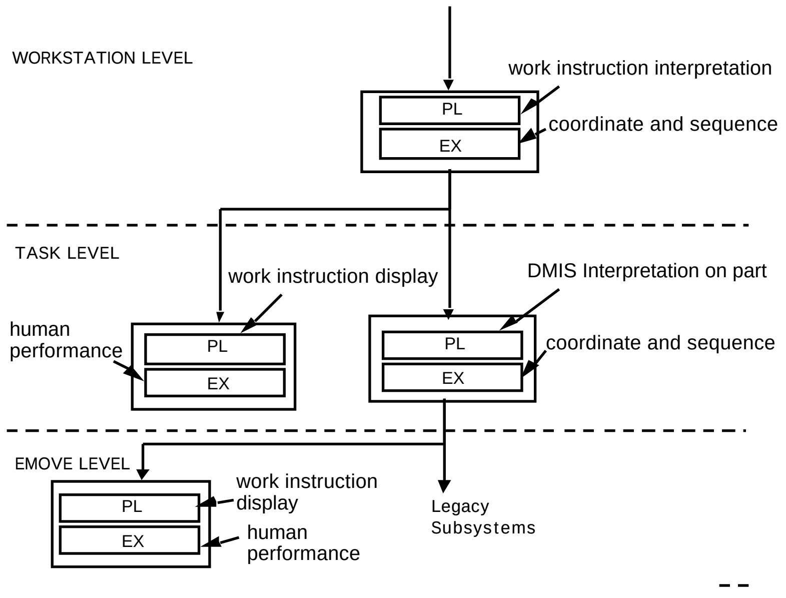


Figure 42: Internal functionality of the Implemented Control Nodes

Figure 43 shows the hardware configuration for the initial testbed implementation. It illustrates the computing hardware and the mapping of the control nodes previously described into the hardware. Also shown is the physical configuration of the communications connections, via ethernet or shared memory mechanisms. (Bit3 is a commercial product used to provide shared memory-like communication between backplanes.) Items indicated in parenthesis are part of the existing NGIS configuration, but not used in the initial implementation described here. They are required for support of the vision system and various inspection probes and their interfaces.

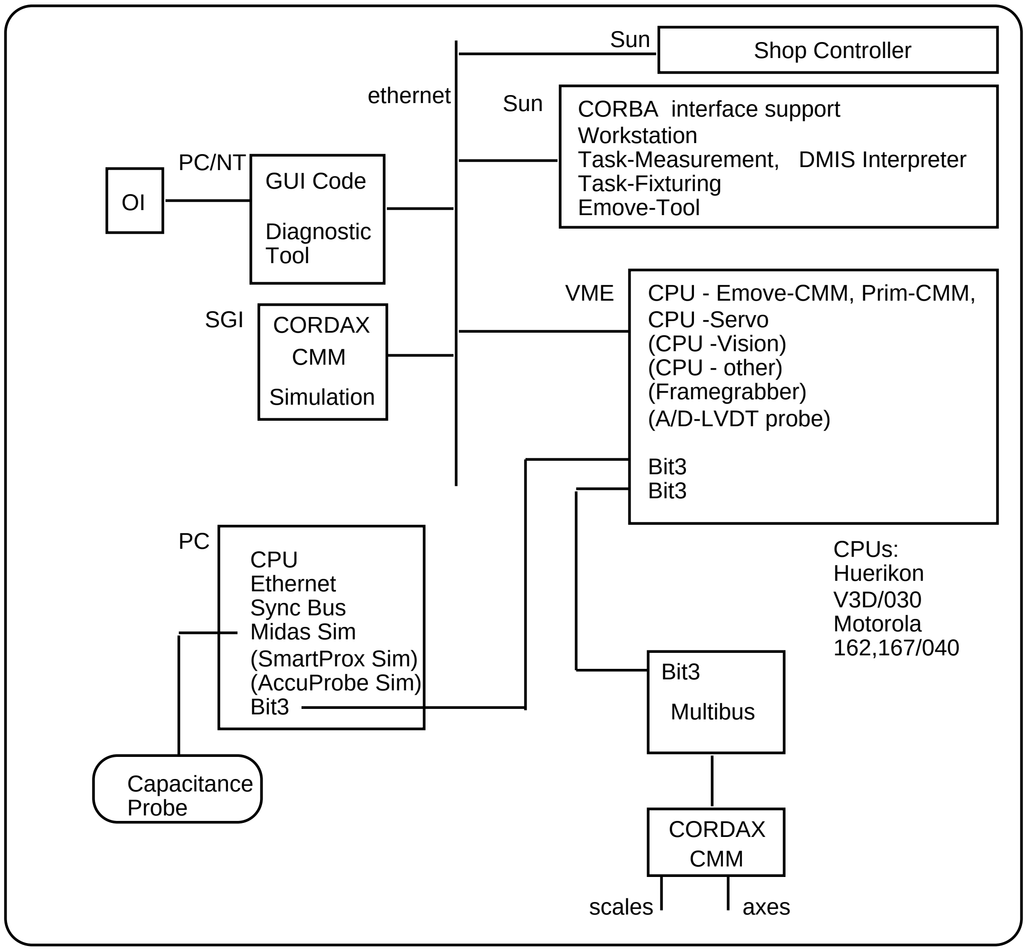


Figure 43: ISAM Testbed Current Hardware Configuration

6. Case Study III, Intelligent Autonomous Vehicles, Exercising Full Architectural Capability

Open system architecture standards are needed for industries that develop intelligent mobile vehicle systems for military security and surveillance and intelligent transportation systems applications. The need is for standard interfaces that facilitate the development and use of commercially available “plug-and-play” components and, in turn, help to reduce cost and development time for system deployment and improvements.

In addition to defining interfaces for subsystems and components, there is also a need for the development and demonstration of intelligent vehicle controllers through the use of advanced, core technology being developed at universities, government labs, and in industry. The United States Army, a major user of the intelligent autonomous vehicle systems which is considering deploying these systems in the battlefield, has identified technology development needs for such systems. These include reliable obstacle detection, tracking, recognition; more mobile off-road performance; night and all weather operations; more autonomous tactical behaviors and single and multiple vehicles.

The emerging industries and the users need measures of performance for evaluating the technology being developed for intelligent mobile systems. Rapid technology evolution, driven by the availability of compact, low cost, high performance computers and new advanced sensors systems, requires the evaluation of autonomous and semi-autonomous control systems with intelligent behaviors. The National Highway Traffic Safety Administration (NHTSA) recognizes the need for evaluating advanced technology for Intelligent Transportation Systems (ITS) and has sponsored work in ISD to develop a real-time measurement and roadway calibration system to evaluate on-vehicle crash avoidance systems for highways. In addition, NIST as part of the Department of Defense (DOD) program to develop technology for next generation unmanned ground vehicles, is being asked to develop measurement systems and conduct tests for evaluating this technology for military applications.

Central to all these issues is the need for an open and standard architecture that allows the integration and exercise of all the advanced component technologies. An instantiated version of RCS, called the 4-D/RCS Reference Model Architecture [29], has been under development as a solution. This development is a jointly sponsored effort that leverages technology from the NIST Intelligent Machines Initiative program and support from the following other agency programs: the Army Research Lab (ARL) Demo III Unmanned Ground Vehicles (UGV) program, the Next Generation Autonomous Vehicle Navigation Control System (AUTONAV) research project between the German Ministry of Defense and the U.S. DOD and the AUTONAV/Department of Transportation (DOT) Crash Avoidance program. This report describes an approach for implementing the architecture and some initial results.

6.1 Scenario Development

As the earlier cases, the scenarios for this project are being developed via interactions with the Army personnel and the other participating government laboratories. An illustration is given:

The scout platoon leader has been told by his commander to expect the enemy beyond Dogwood Hill. The platoon orders its two sections to establish an observation post on Dogwood Hill. Establishing an observation post means finding one or more locations from which the vehicles can survey an area visually, or through the use of other sensors, while being concealed or covered from sight by the enemy. Cover means being hidden from ground vehicles. Concealment means being hidden from air vehicles. Initial planning is done using a map of the area that contains elevation contour lines and features, such as roads and water. A section contains a leader, and one or more soldiers, all traveling in a single vehicle.

Given this mission, the platoon leader divides the relevant map area into two sectors that are assigned to the two sections. The two sectors are parallel tracts along which the sections plan their motion towards their destination. The sections are assigned code names "Alpha" and "Bravo." A rough trajectory for each section vehicle is derived, based on map elevation and features. Consideration is given to cover and concealment. In a "bounding overwatch" movement, one vehicle travels ahead while the other one looks for enemies from a fixed position. Each vehicle travels to locations that provide a view of the forward area from a covered position. When the moving vehicle reaches its next location, it notifies the overwatching vehicle, which then commences travel, after notifying the other vehicle. The platoon leader may receive these communications as well, which occur over the radio. The rear vehicle moves to a location ahead of the other vehicle until it reaches its chosen next overwatch position. This movement continues until both vehicles have attained their final observation positions.

A narrative of the "establish observation post" scenario could evolve as follows:

1) Battalion commander sends instructions to Platoon Commander to establish observation post at Dogwood Hill by 15:00. Some military information may also be conveyed, such as latest estimate of enemy positions and direction of movement. A Named Area of Interest (NAI) is defined beyond Dogwood Hill. A NAI defines a region on a map that is to be given particular attention during a mission.

2) Platoon Commander (PC) acknowledges receipt of instructions and intention to carry those out.

3) PC reviews mission with 2 section leaders, George and Jerry. Together, they look at map of area leading up to Dogwood Hill. The map contains elevation contours and feature information such as tree stands, roads, streams, lakes, and other bodies of water. The resolution of the map is typically fairly low, e.g., 100 meters. Using his knowledge of other battlefield information (e.g., enemy and friendly locations) and expertise in spatial reasoning with two-dimensional maps, PC subdivides the corridor of approach towards the observation area into two roughly parallel ribbons. Terrain features, such as water, and elevation contours guide the division between

the two. He will also mark the map with other military information, such as Line of Departure, Avenue of Approach for the enemy, and Phase Lines.

4) PC establishes coarse-resolution paths for the two sections. He looks for paths that take advantage of cover, such as behind hills or through stands of trees. He avoids wide-open areas and impassable areas. He also strives to place the vehicles at locations from which they can observe as much as possible in front of themselves without revealing themselves. Using criteria such as coverage, concealment, traversability, and visibility, a rough trajectory with possible stopping locations for overwatching is sketched on the map for each corridor. The two trajectories have to be coordinated, since one vehicle must be stopped and overwatching while the other one is moving forward. The moving vehicle must be given a reasonable goal location to stop so that the other vehicle can begin its motion. "Reasonable" is determined from years of experience and training and is based on ensuring that the stationary vehicle does not lose visual contact with the moving vehicle and other criteria.

5) PC gives orders to 2 section leaders: goal of mission, mission parameters, individual corridors, path to travel, and suggested stopping points. The overall spatial scope of the mission covers a distance of about 10 km. The width of the sector being covered by the two sections is about 500m.

6) PC assigns George the Alpha (left) corridor.

7) PC assigns Jerry the Bravo (right) corridor.

8) George and Jerry acknowledge that they understand the mission and parameters.

9) A rehearsal of the mission takes place using the map. The soldiers in both sections watch while George and Jerry place micro-vehicles at the determined locations and simulate the coordinated movements.

10) George and Jerry move the vehicles to the Line of Departure at 15:40 and radio that they are in position.

11) The PC radios that the mission may begin.

12) George (the Alpha Section) radios that he is starting movement.

13) George moves towards the first selected overwatch position for his side. He moves in a manner dictated by the terrain and the enemy location. The navigator uses a map with mission markings on it, along with landmark recognition and a Global Positioning System (GPS) to direct the vehicle along its assigned path. Other constraints for movement are:

a) Move as quickly as possible.

b) Select your path so that you are not visible by the enemy.

c) Keep your sensors pointed so that the targeted area is always covered.

d) Avoid “flanking” the enemy, i.e., do not expose your flank to them.

e) Select your next location by assessing the terrain and features in the near-range (50m - 1000m).

f) Assess upcoming terrain and features by performing “military cresting.” Military cresting is slowly creeping up a hill until only sensor(s) peer over the crest of the hill. One of the soldiers in the vehicle may stand up and look through binoculars and notify the driver when to stop. Using the sensing capability, which relies primarily on human vision, assess your next movement point. Back the vehicle up, and go around the hill at a lower elevation, while avoiding flanking.

14) As George approaches his planned stopping point, he refines the final location based on actual conditions: cover and concealment opportunities that were not captured in the maps his platoon leader used.

15) When George reaches his first stopping point, he radios Jerry that he’s set.

16) Jerry radios George that he’s beginning movement.

17) Jerry moves the Bravo section forward and follows steps 13) and 14) until he’s reached his planned stopping point.

18) While the Bravo section is moving, the Alpha section overwatches to the front. George uses whatever sensors are appropriate to look forward as far as possible for potential enemy locations. Techniques for searching are sensor-dependent.

19) When Jerry is in an overwatch position, about 3 kilometers from Dogwood Hill, possible enemy locations are detected by one of his sensors. Per military doctrine, the following immediately occur:

a) The other sections and the PC are notified of location and situation via a spot report.

b) George and Jerry move their vehicles into overwatch positions. They quickly identify candidate locations in their surroundings that provide cover and concealment. George has to move backwards about 600 m to a more suitable place he’d noted as he was traveling.

c) Any other sections that may be visible from enemy locations(s) move to an overwatch position. These include sections attached to other platoons or battalions.

d) All sections that may be able to confirm sightings try to do so and notify PC and peers of their results.

20) The commander assesses the situation and sends an updated mission to the sections. Instead of proceeding to establish an observation post on Dogwood Hill, they are to remain in overwatch positions and continue sending reports on enemy activities..

6.2 Control Hierarchy and Task Decomposition

Some key aspects of behavior and task decomposition emerge from this high-level scenario. In the Demo III Program, an autonomous vehicle will fulfill the role of a section. As expected in a military hierarchy, commands flow down from superior to subordinates. Status flows back up the command chain. The planning horizons for the platoon level are about an order of magnitude larger than those of the section level (e.g., 10 km versus less than 1 km per section movement). Plans for section paths are planned at the platoon level using coarse data. The sections make extensive use of their sensing systems, especially human vision, for fine-tuning the actual path traveled. Other sensors include GPS and mission-specific ones. Experience, training, and external knowledge bases are essential for making military plans. External knowledge bases include maps and military intelligence. Preprogrammed reactive behaviors are useful in military missions. For example, several behaviors were triggered when potential enemy sightings occurred.

Based on analysis of scenarios like the one above, a control hierarchy emerges. In the control hierarchy under development, Figure 44, active and passive vision are the primary sensors for performing dynamic image perception analysis during navigation. Other sensors, like accelerometers, Inertial Navigation System (INS) and Differential Global Positioning System (DGPS), measure vehicle motion through the environment and provide a basis for precise localization of vehicles, targets, obstacles, and terrain features on a map database. These sensors and their controls form the servo levels of the attention and reconnaissance, surveillance, and target acquisition (RSTA) mission package subsystems.

At the lower levels, including primitive, subsystem, and vehicle, hybrid vision processing involves the fusion of real-time laser range image (LADAR), vision image, and inertial navigation system (INS) data. Obstacles detected in LADAR images are projected into camera images. The algorithms at these levels perform obstacle detection, recognition, and tracking. The locomotion subsystem uses this information to perform obstacle avoidance and vehicle dynamic control. The locomotion servo control involves such actuators as steering wheel, throttle, and brakes.

The higher levels of the control hierarchy involve a platoon controller that coordinates multiple sections of XUVs in the current Army scout mission scenario configuration. In the DOT programs and others, different vehicle grouping will be devised. A section level controller coordinates multiple vehicles. These high level controllers take the low level map data, including terrain, obstacles, and other military information and derive maps at their own resolutions. The map information is then used for mission planning.

The operator interface function is realized via the Operator Control Units (OCU), which follows the hierarchical construct. The high-level mission simulation will be provided via the Modular Semi-Automated Forces (Mod-SAF) system from the U.S. Army Mounted Maneuver Battlefield Laboratory, Ft. Knox, KY.

Task decomposition is illustrated in Figure 44. The platoon receives a command to “establish an observation post.” The planning process generates the responsibility areas and routes for the subordinate sections or squads. The information is embedded in the command “sc_est_obs.” The section/squad planning process generates waypoints that coordinates the overwatch by the two vehicles and sends “goto” commands to the vehicles. The goto command may include responsibility regarding surveying a Named Area of Interest (NAI) or communication directives.   The vehicle controller decomposes this responsibility into subtasks for each of the subsystem controllers. The RSTA mission package will be targeted toward the NAI, while the locomotion subsystem will navigate through optimized routes to support RSTA. These subsystem commands are, in turn, decomposed into the actuator commands for locomotion and sensor pointing/adjusting so that the assigned commands can be accomplished.

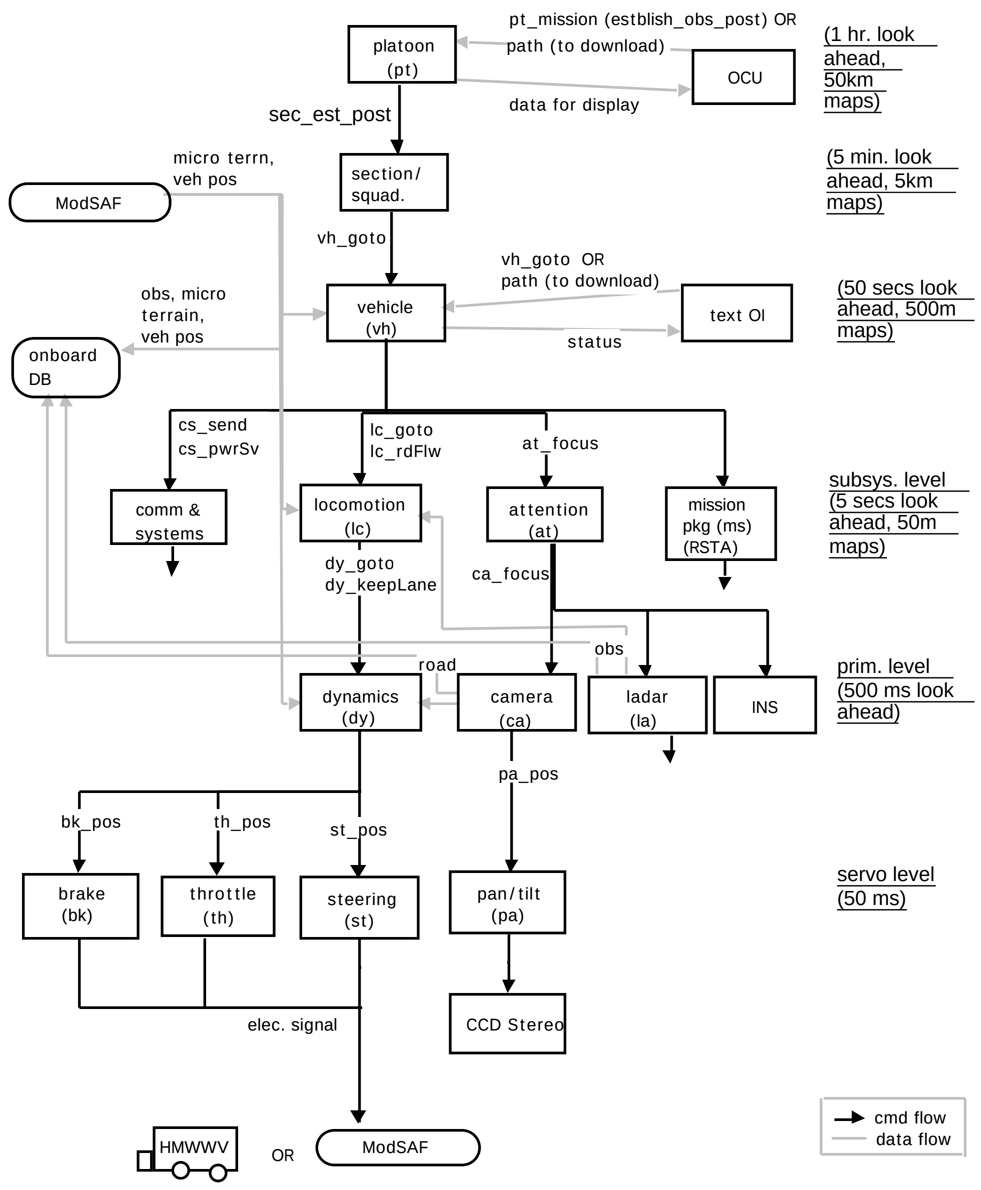


Figure 44: Military Unmanned Vehicle Control Hierarchy

7. Summary

The authors described the hierarchical real-time control systems (RCS) reference model architecture aiming at designing and developing intelligent control for large and complex engineering systems. RCS is based on several general and fundamental principles of engineering systems. The methodology, or the development process to apply RCS to the system control, is described. This process unifies multiple engineering principles, including: multiple spatial and temporal resolutions, behavior-oriented, object-oriented, and functional decomposition.

In the described process, developers arrange controllers in hierarchies according to assigned responsibility and authority. This provides consistent correspondence to both the military chain-of-command and manufacturing system configurations. The developers can easily map current operations, possibly manual, to the automation models. In other words, this feature facilitates incremental upgrade from current manual operations to automatic operations. Operators can easily fit in the RCS hierarchies to perform the designated parts of the system control.

The RCS reference model architecture and methodology provide a simple and systematic mechanism to acquire, describe, and organize domain operational knowledge. The single node concept improves human understanding of the design.

The process template based implementation approach provides a system wide consistent interfacing infrastructure. This frees up the developers from the infrastructural issues and allows them to focus on individual component technology. Developers can apply the templates at an early stage and construct a control hierarchy capable of interfacing and executing individual component technologies that are linked in. This is called the generic shell for the target control system. This shell serves as the backbone for the system operations. This shell based approach is generic and applicable to a broad class of complex commercial and military control systems applications.

The RCS methodology prescribes modular real-time simulation and animation functions. This facilitates early concept visualization and rapid prototyping that, in turn, reduces development cost.

The authors described three ISD case study control systems. The descriptions progressively demonstrate the richness of the RCS architecture, from a generic process template based system to real-time rich sensing and planning systems. The ultimate goal is to implement the latest control systems that capture the full RCS capability. The case studies also described the evolution and the road map of the methodology, from base class models to process and interface definitions and to the plan of fully exercising the methodology.

DISCLAIMER

Certain commercial products or company names are identified in this paper to describe our study adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the products or names identified are necessarily the best available for the purpose.

REFERENCES

1 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.

2 Antsaklis, P.J., et al, eds., *Lecture Notes in Computer Science Series, Vol. 999, Hybrid Systems II,* Springer, New York, New York, 1995.

3 Kowal, J.A., *Behavior Models*, Prentice Hall, Englewood Cliffs, New Jersey, 1992.

4 Astrom, K.J. and Wittenmark, B., *Adaptive Control*, Addison-Wesley, Reading, Massachusetts, 1989

5 Raouf, A. and Ben-Daya eds., *Manufacturing Research and Technology Series, Vol. 23, Flexible Manufacturing Systems: Recent Developments,* Elsevier, New York, New York, 1995.

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

7 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.

8 OSACA, Project Web Page, http://www.isw.uni-stuttgart.de/projekte/osaca/english/osaca.htm

9 Albus, J.S., et al., “NIST Support to the Next Generation Controller Program,” NIST Interagency Report, NISTIR 4888, Gaithersburg, MD, 1991.

10 Lima, P.U., and Saridis, G.N., *Design of Intelligent Control Systems Based on Hierarchical Stochastic Automata*, Series in Intelligent Control and Intelligent Automation, Vol. 2, World Scientific Publishing Co., River Ridge, NJ, 1996.

11 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.

12 Barkmeyer, E.J., et. al., “Background Study, Requisites, Rationale, and Technology Overview for the Systems Integration for Manufacturing Applications (SIMA) program,” NISTIR 5662, Gaithersburg, MD, 1995.

13 Wallace, S., Senehi, M.K., Barkmeyer, E., Luce, M., Ray, S., and Wallace, E., “Control Entity Interface Specification,” NISTIR 5272, September 1993.

14 Donmez, M. A., ed., “Progress Report of the Quality in Automation Project for FY90,” NISTIR 4536, March 1991.

15 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.

16 Proctor, F., et al., "Validation of Standard Interfaces for Machine Control," Intelligent Automation and Soft Computing: Trends in Research, Development, and Applications, Vol. 2, TSI Press, Albuquerque, NM, 1996.

17 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.

18 H. Huang, “An Architecture and Methodology for Intelligent Control,” IEEE Expert, April 1996.

19 A. J. Barbera, J. S. Albus, M.L. Fitzgerald, and L.S. Haynes, "RCS: The NBS Real-Time Control System," Robots 8 Conference and Exposition, Detroit, MI, June 1984.

20 Albus, J.S., "System Description and Design Architecture for Multiple Autonomous Undersea Vehicles", NIST Technical Note 1251, National Institute of Standards and Technology, U. S. Department of Commerce, September 1988.

21 R. Quintero, and Barbera, A., “A Software Template Approach to Building Complex Large-Scale Intelligent Control Systems,” in Proceedings of the 8th IEEE International Symposium of Intelligent Control, Chicago, Illinois, 1993.

22 G. Booch, Object-Oriented Analysis and Design, The Benjamin/Cummins Publishing Company, Inc., 1994.

23 Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., Object-Oriented Modeling and Design, Prentice-Hall, 1991.

24 P. Coad, and Yourdan, E., Object-Oriented Analysis, Yourdan Press, 1991.

25 Shlaer, S. and Mellor, Object Lifecycles: Modeling the World in States, Prentice-Hall, 1992.

26 Albus, J.S., Intelligent System Architecture for Manufacturing, Draft, to be published as NISTIR, 1998.

27 Huang, H., “Intelligent Manufacturing System Control: Reference Model and Initial Implementation,” Proceedings of the 35th IEEE Conference on Decision and Control (CDC), Kobe, Japan, December 1996.

28 http://www.mel.nist.gov/nsmt/

29 Albus, J. S., 4-D/RCS, A Reference Model Architecture for Demo III, NISTIR 5994, Gaithersburg, MD, 1997.

30 J. Albus, “Outline for a Theory of Intelligence.” IEEE Transactions on Systems, Man, and Cybernetics, Vol. 21, No. 3, May/June 1991.

31 J. S. Albus and A. Meystel, “Behavior Generation: The Architectural Issues.” NISTIR, 1997.

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

33 Horst, J.A. and Scott, H., Scenario for Demonstrating the Intelligent System Architecture for Manufacturing (ISAM), NIST ISD Internal Document, April, 1998.

34 http://isd.cme.nist.gov/proj/rcs_lib/

35 Brown D., Marvin, J., and Scherer W., “A Survey of Intelligent Scheduling Systems,” *Intelligent Scheduling Systems*, 1995.

36 Proctor, F., et al., "Validation of Standard Interfaces for Machine Control," Intelligent Automation and Soft Computing: Trends in Research, Development, and Applications, Vol. 2, TSI Press, Albuquerque, NM, 1996.

37 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.

38 H. Huang, “An Architecture and Methodology for Intelligent Control,” IEEE Expert, April 1996.

39 A. J. Barbera, J. S. Albus, M.L. Fitzgerald, and L.S. Haynes, "RCS: The NBS Real-Time Control System," Robots 8 Conference and Exposition, Detroit, MI, June 1984.

40 Albus, J.S., "System Description and Design Architecture for Multiple Autonomous Undersea Vehicles", NIST Technical Note 1251, National Institute of Standards and Technology, U. S. Department of Commerce, September 1988.

41 Huang, H., “Operator Interface And Situation Perception in Hierarchical Intelligent Control: A Case Study,” Combined IEEE International Symposium on Intelligent Control (ISIC), International Conference on Control Applications (CCA) and International Symposium on Computer-Aided Control System Design (CACSD), Dearborn, Michigan, September 16-19, 1996.

42 Albus, J.S., “From Cerebellar Model Articulation Controller to a Reference Model for Intelligent Control,” from the Proceedings of the 15th Annual International Conference IEEE Engineering in Medicine & Biology Society, San Diego, CA, October 28-21, 1993.

43 Lauwerier, H., *Fractals*, Princeton University Press, Princeton, NJ, 1991.

44 Mayhew, D. J., *Principles and Guidelines in Software User Interface Design*, Prentice Hall, Englewood Cliffs, NJ, 1992.

45 Bloom, H. M. and Christopher, N., “A Framework for Distributed and Virtual Discrete Part Manufacturing,” from the proceedings of CALS EXPO 96, October 1996, Long Beach, CA.

46 Nashman, M., Hong, T., Rippey, W. G., and Herman, M., "An Integrated Vision Touch-Probe System for Inspection Tasks,” Machine Vision Applications, Cleveland, Ohio, June 1996.

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

48 http://isd.cme.nist.gov/info/team, web location for Technologies Enabling Agile Manufacturing (TEAM) Application Programming Interfaces.

49 http://isd.cme.nist.gov/brochure/NGIS.html