RCS: A Reference Model Architecture for Intelligent Machine Systems

James S. Albus

Robot Systems Division
Manufacturing Engineering Laboratory
National Institute of Standards and Technology

Abstract

A reference model architecture based on the NIST Real-time Control System (RCS) has been developed for real-time intelligent control systems. RCS partitions the control problem into four basic elements: behavior generation (or task decomposition), world modeling, sensory processing, and value judgment. It clusters these elements into computational nodes that have responsibility for specific subsystems, and arranges these nodes in hierarchical layers such that each layer has characteristic functionality and timing. The RCS reference model architecture has a systematic regularity, and recursive structure that suggests a canonical form. Control systems based on the RCS have been built for a number of applications. Examples of RCS applications include a Machine Tool Controller, a Deburring and Chamfering Robot, an Advanced Manufacturing Research Facility, a Space Station Flight Telerobotic Servicer, Multiple Autonomous Undersea Vehicles, Unmanned Ground Vehicles, and Intelligent Vehicles and Highway Systems.

RCS: A Reference Model Architecture for Intelligent Machine Systems

James S. Albus

Robot Systems Division
Manufacturing Engineering Laboratory
National Institute of Standards and Technology

Introduction

The Real-time Control System (RCS) is a reference model architecture for intelligent real-time control systems. It partitions the control problem into four basic elements: task decomposition, world modeling, sensory processing, and value judgment. It clusters these elements into computational nodes that have responsibility for specific subsystems, and arranges these nodes in hierarchical layers such that each layer has characteristic functionality and timing. The RCS architecture has a systematic regularity, and recursive structure that suggests a cannonical form.

An RCS design paradigm begins with analysis of tasks and goals, then develops control algorithms, data structures, and sensory information necessary to perform the tasks and accomplish the goals. Attempts are underway to develop this approach into a formal methodology for engineering real-time intelligent control systems and analyzing their performance.

An outline for a theory of intelligence based on RCS [1] defines intelligence as the ability to act appropriately in an uncertain environment, where appropriate action is that which increases the probability of success, and success is the achievement of behavioral goals. The intelligent system acts so as to maximize probability of success and minimize probability of failure. Both goals and success criteria are generated in the environment external to the intelligent system. At a minimum, intelligence requires the abilities to sense the environment, make decisions, and control action. Higher levels of intelligence require the abilities to recognize objects and events, store and use knowledge about the world, and to reason about and plan for the future. Advanced forms of intelligence have the abilities to perceive and analyze, to plot and scheme, to choose wisely and plan successfully in a complex, competitive, hostile world. The amount of intelligence is determined by the computational power of the computing engine, the sophistication and elegance of algorithms, the amount and quality of information and values, and the efficiency and reliability of the system architecture. The amount of intelligence can grow through programming, learning, and evolution. Intelligence is the product of natural selection, wherein more successful behavior is passed on to succeeding generations of intelligent systems, and less successful behavior dies out. Natural selection is driven by competition between individuals within a group, and groups within the world.

The above theory of intelligence is expressed in terms of a reference model architecture for real-time intelligent control systems based on the RCS (Real-time Control System) [2]. RCS partitions the control problem into four basic elements: behavior generation (or task decomposition), world modeling, sensory processing, and value judgment. It clusters these elements into computational nodes that have responsibility for specific subsystems, and arranges these nodes in hierarchical layers such that each layer has characteristic functionality and timing. The RCS reference model architecture has a systematic regularity, and recursive structure that suggests a canonical form.

This paper is divided into seven sections. Section 1 describes the evolution of the RCS system through its various versions. Section 2 gives an example of RCS applied to a machining workstation application. Section 3 describes the timing of real-time task decomposition (planning and execution) and sensory processing(sampling and integration) at the various layers in the RCS hierarchy. Sections 4, 5, 6, and 7 define the functionality and contents of the Task Decomposition, World Modeling, Sensory Processing, and Value Judgment modules respectively.

Section 1. Evolution of RCS

RCS has evolved through variety of versions over a number of years as understanding of the complexity and sophistication of intelligent behavior has increased. The first implementation was designed for sensory-interactive robotics by Barbera in the mid 1970's [3]. In RCS-1, the emphasis was on combining commands with sensory feedback so as to compute the proper response to every combination of goals and states. The application was to control a robot arm with a structured light vision system in visual pursuit tasks. RCS-1 was heavily influenced by biological models such as the Marr-Albus model of the cerebellum [4], and the Cerebellar Model Arithmetic Computer (CMAC) [5]. CMAC can implement a function of the form

$$\text{Command}_{t+1}(i-1) = H_i(\text{Command}_t(i), \text{Feedback}_t(i))$$

where H_i is a single valued function of many variables
i is an index indicating the hierarchical level
t is an index indicating time

CMAC thus became the reference model building block of RCS-1, as shown in Figure 1(a). A hierarchy of these building blocks was used to implement a hierarchy of behaviors such as observed by Tinbergen [6] and others. CMAC becomes a state-machine when some of its outputs are fed directly back to the input, so RCS-1 was implemented as a set of state-machines arranged in a hierarchy of control levels. At each level, the input command effectively selects a behavior that is driven by feedback in stimulus-response fashion. RCS-1 is thus similar in many respects to Brooks subsumption architecture [7], except that RCS selects behaviors before the fact through goals expressed in commands, rather than after the fact through subsumption.

The next generation, RCS-2, was developed by Barbera, Fitzgerald, Kent, and others for manufacturing control in the NIST Automated Manufacturing Research Facility (AMRF) during the early 1980's [8,9,10]. The basic building block of RCS-2 is shown in Figure 1(b). The H function remained a finite state machine state-table executor. The new feature of RCS-2 was the inclusion of the G function consisting of a number of sensory processing algorithms including structured light and blob analysis algorithms. RCS-2 was used to define an eight-level hierarchy consisting of Servo, Coordinate Transform, E-Move, Task, Workstation, Cell, Shop, and Facility levels of control. Only the first six levels were actually built. Two of the AMRF workstations fully implemented five levels of RCS-2. The control system for the Army Field Material Handling Robot (FMR)[11] was also implemented in RCS-2, as were the Army TMAP and TEAM semi-autonomous land vehicle projects.

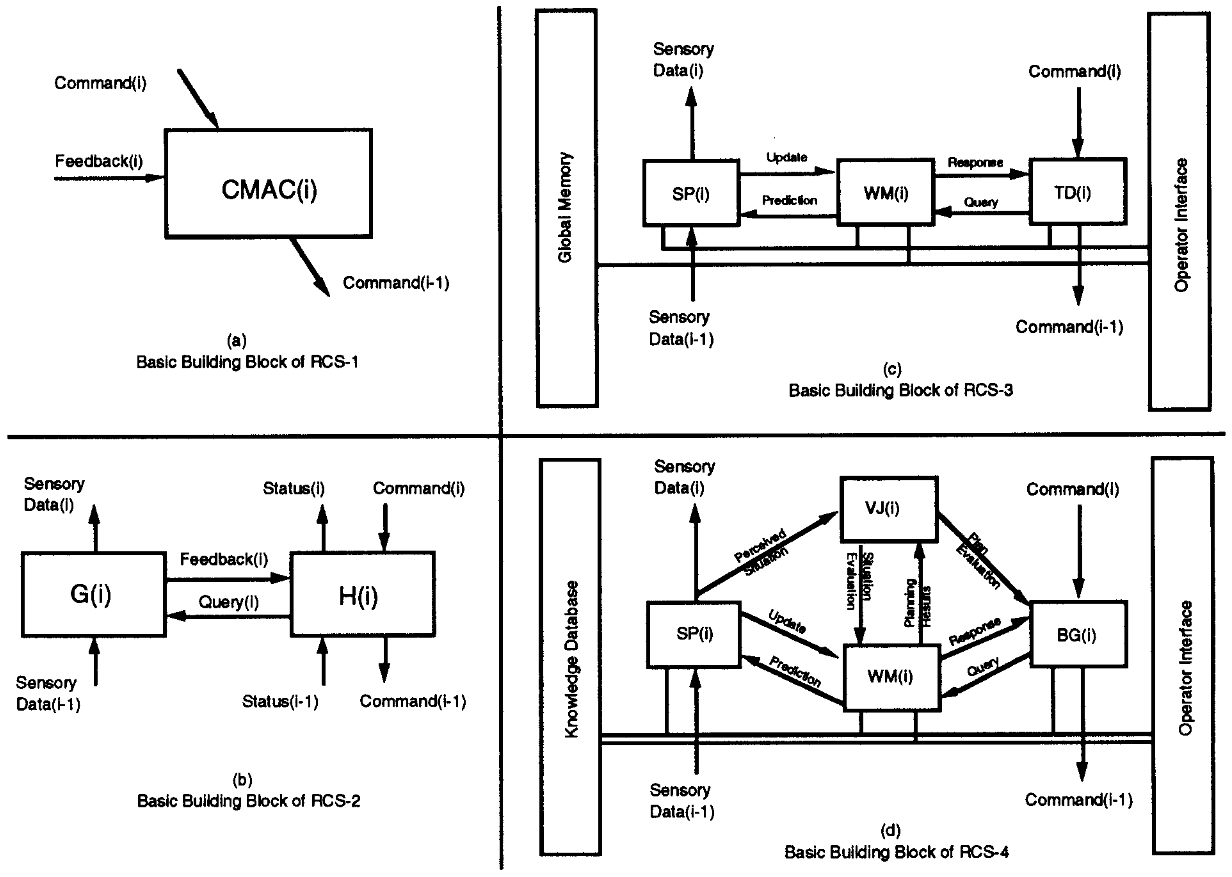

Figure 1. The basic building blocks of the RCS control paradigm.

RCS-3 was designed for the NBS/DARPA Multiple Autonomous Undersea Vehicle (MAUV) project [12] and was adapted for the NASA/NBS Standard Reference Model Telerobot Control System Architecture (NASREM) being used on the space station Flight Telerobotic Servicer [13]. The basic building block of RCS-3 is shown in Figure 1(c). The block diagram of NASREM is shown in Figure 2. The principle new features introduced in RCS-3 are the World Model and the operator interface. The inclusion of the World Model provides the basis for task planning and for model-based sensory processing. This led to refinement of the task decomposition (TD) modules so that each have a job assigner, and planner and executor for each of the subsystems assigned a job. This corresponds roughly to Saridis' three-level control hierarchy [14].

NASREM: NASA/NBS STANDARD REFERENCE MODEL

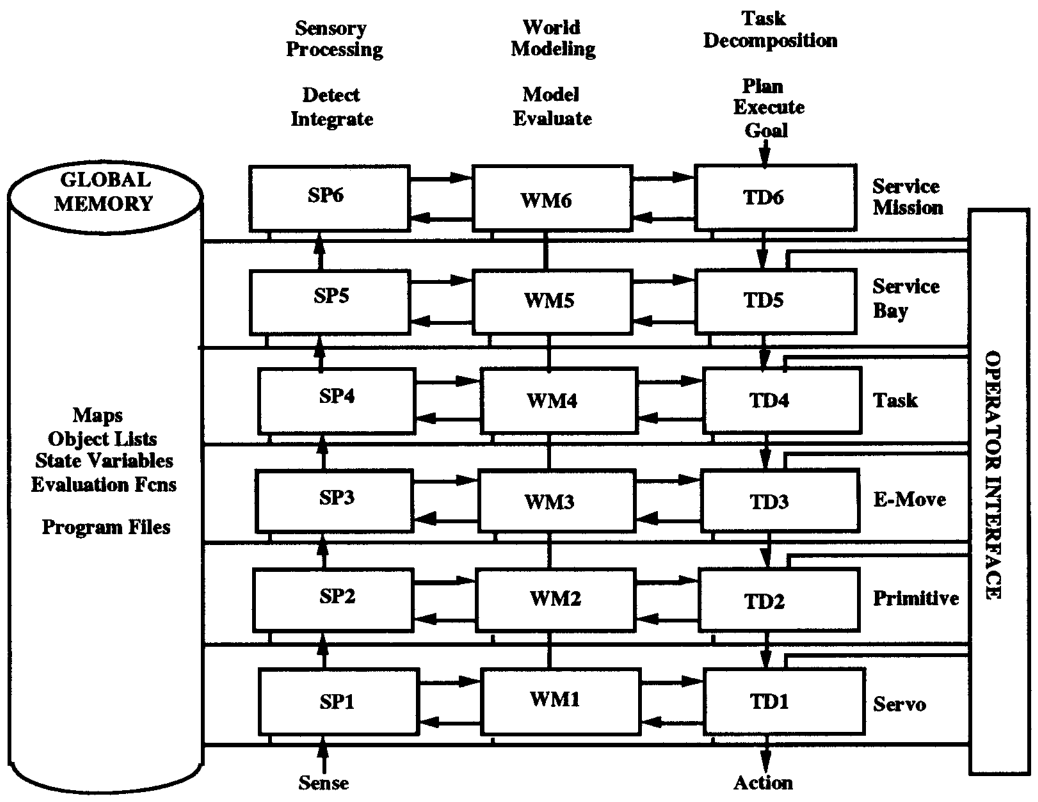

Figure 2. The NASREM (RCS-3) control system architecture.

RCS-4 is under current development by the NIST Robot Systems Division. The basic building block is shown in Figure 1(d). The principle new feature in RCS-4 is the explicit representation of the Value Judgment (VJ) system. VJ modules provide to the RCS-4 control system the type of functions provided to the biological brain by the limbic system. The VJ modules contain processes that compute cost, benefit, and risk of planned actions; and that place value on objects, materials, territory, situations, events, and outcomes. Value state-variables define what goals are important, and what objects or regions should be attended-to, attacked, defended, assisted, or otherwise acted upon. Value judgments, or evaluation functions, are an essential part of any form of planning or learning. However, the evaluation functions are usually not made explicit, or assigned to a separate module as in RCS-4. The application of value judgments to intelligent control systems has been addressed by George Pugh [15]. The structure and function of VJ modules are developed more completely developed in [1].

RCS-4 also uses the term behavior generation (BG) in place of the RCS-3 term task decomposition (TD). The purpose of this change is to emphasize the degree of autonomous decision making. RCS-4 is designed to address highly autonomous applications in unstructured environments where high-bandwidth communications are impossible, such as unmanned vehicles operating on the battlefield, deep undersea, or on distant planets. These applications require autonomous value judgments and sophisticated real-time perceptual capabilities. RCS-3 will continue to be used for less demanding applications such as manufacturing, construction, or telerobotics for near-space, or shallow undersea operations, where environments are more structured and communication bandwidth to a human interface is less restricted.

Section 2. A Machining Workstation Example

Figure 3 illustrates how the RCS-3 system architecture can be applied to a specific machining workstation consisting of a machine tool, a robot, an inspection machine, and a part buffer. RCS-3 produces a layered graph of processing nodes, each of which contains a Task Decomposition (TD), World Modeling (WM), and Sensory Processing (SP) module. These modules are richly interconnected to each other by a communications system. At the lowest level, communications are typically implemented through common memory or message passing between processes on a single computer board. At middle levels, communications are typically implemented between multiple processors on a backplane bus. At high levels, communications can be implemented through bus gateways and local area networks. The global memory, or knowledge database, and operator interface are not shown in Figure 3.

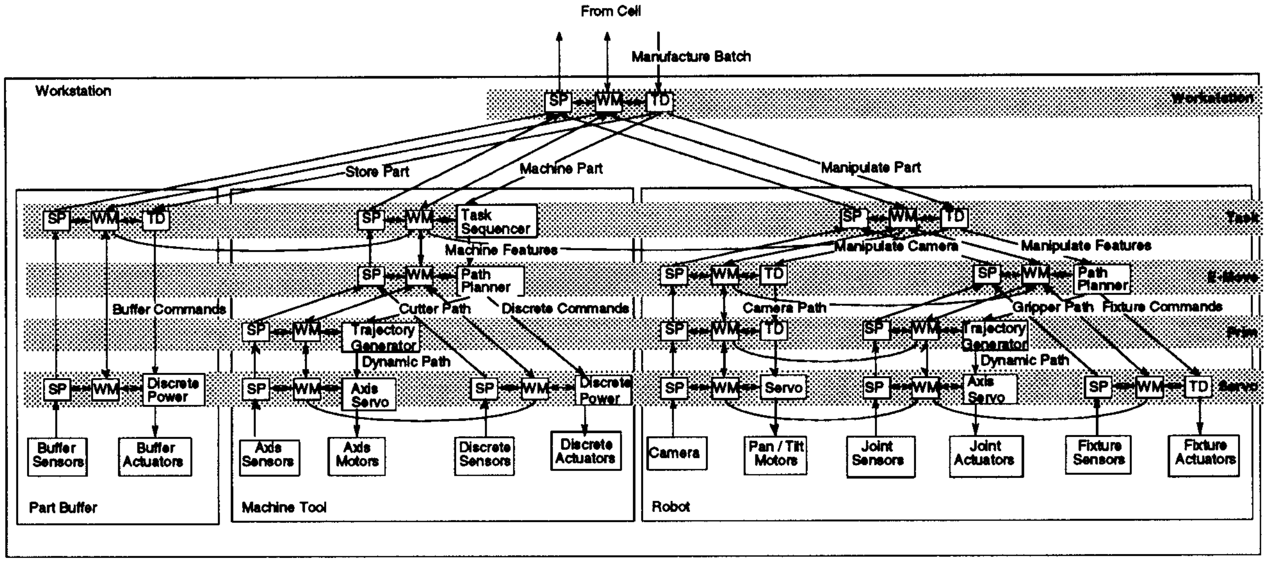

Figure 3. A RCS-3 application of a machining workstation containing a machine tool, part buffer, and robot with vision system.

Figure 3 illustrates the ability of RCS-3 to integrate discrete sensors such as microswitches with more complex sensors such as cameras and resolvers. Discrete commands can be issued to valves and fixtures, while continuous signals are provided to servoed actuators. Notice that in some branches of the control tree, nodes at some levels may be absent. For example, in the case of the part buffer, discrete commands at the Task level can be directly executed by the Servo level. In the case of the part fixture, discrete commands issued from the robot E-Move level can be executed by the Servo level. In these cases, the missing modules can be thought of as degenerate computing nodes that produce unity gain pass-through of inputs to outputs.

The branching of the control tree (for example, between the camera and manipulator subsystems of the robot), may depend on the particular algorithm chosen for decomposing a particular task. The specifications for branching reside in the task frame (defined later) of the current task being executed. Similarly, the specification for sharing information between WM modules at a level also are task dependent. In Figure 3, the horizontal curved lines represent the sharing of state information between subtrees in order to synchronize related tasks. The information that must be shared is also dependent on the specific choice of task decomposition algorithms defined in the task frame.

The functionality of each level in the control hierarchy can be derived from the characteristic timing of that level, and vice versa. For example, in a manufacturing environment, the following hierarchical levels are becoming more or less standard.

Level 7—Shop

The shop level schedules and controls the activities of one or more manufacturing cells for an extended period on the order of 24 hours. (The specific timing numbers given in this example are representative only, and may vary from application to application.) At the shop level, orders are sorted into batches and commands are issued to the cell level to develop a production schedule for each batch. At the shop level, the world model symbolic database contains names and attributes of orders and the inventory of tools and materials required to fill them. Maps describe the location of, and routing between, manufacturing cells.

Level 6—Cell

The cell level schedules and controls the activities of several workstations for about a one hour look ahead. Batches of parts and tools are scheduled into workstations, and commands are issued to workstations to perform machining, inspection, or material handling operations on batches or trays of parts. The world model symbolic database contains names and attributes of batches of parts and the tools and materials necessary to manufacture them. Maps describe the location of, and routing between, workstations. (The Shop and Cell levels are above the levels shown in the Figure 3 example.) The output from the cell level provides input to the workstation level.

Level 5—Workstation

The workstation level schedules tasks and controls the activities within each workstation with about a five minute planning horizon. A workstation may consist of a group of machines, such as one or more closely coupled machine tools, robots, inspection machines, materials transport devices, and part and tool buffers. Plans are developed and commands are issued to equipment to operate on material, tools, and fixtures in order to produce parts. The world model symbolic database contains names and attributes of parts, tools, and buffer trays in the workstation. Maps describe the location of parts, tools, and buffer trays.

Level 4—Equipment task

The equipment level schedules tasks and controls the activities of each machine within a workstation with about a 30 second planning horizon. (Tasks that take much longer may be broken into several 30 second segments at the workstation level.) Level 4 decomposes each equipment task into elemental moves for the subsystems that make up each piece of equipment. Plans are developed that sequence elemental movements of tools and grippers, and commands are issued to move tools and grippers so as to approach, grasp, move, insert, cut, drill, mill, or measure parts. The world model symbolic database contains names and attributes of parts, such as their size and shape (dimensions and tolerances) and material characteristics (mass, color, hardness, etc.). Maps consist of drawings that illustrate part shape and the relative positions of part features.

Level 3—Elemental move (E-move)

The E-move level schedules and controls simple machine motions requiring a few seconds, such GO-ALONG-PATH, MOVE-TO-POINT, MILL-FACE, DRILL-HOLE, MEASURE-SURFACE, etc. (Motions that require significantly more time may be broken up at the task level into several elemental moves.) Plans are developed and commands are issued that define safe path waypoints for tools, manipulators, and inspection probes so as to avoid collisions and singularities, and assure part quality and process safety. The world model symbolic database contains names and attributes of part features such as surfaces, holes, pockets, grooves, threads, chamfers, burrs, etc. Maps consist of drawings that illustrate feature shape and the relative positions of feature boundaries.

Level 2—Primitive

The primitive level plans paths for tools, manipulators, and inspection probes so as to minimize time and optimize performance. It computes tool or gripper acceleration and deceleration profiles taking into consideration dynamical interaction between mass, stiffness, force, and time. Planning horizons are on the order of 300 milliseconds. The world model symbolic database contains names and attributes of linear features such as lines, trajectory segments, and vertices. Maps (when they exist) consist of perspective projections of linear features such as edges, lines or of tool or end-effector trajectories.

Level 1—Servo level

The servo level transforms commands from tool path to joint actuator coordinates. Planners interpolate between primitive trajectory points with a 30 millisecond look ahead. Executors servo individual actuators and motors to the interpolated trajectories. Position, velocity, or force servoing may be implemented, and in various combinations. Commands that define actuator torque or power are output every 3 milliseconds (or whatever rate is dictated by the machine dynamics and servo performance requirements). The servo level also controls the output drive signals to discrete actuators such as relays and solenoids. The world model symbolic database contains values of state variables such as joint positions, velocities, and forces, proximity sensor readings, position of discrete switches, condition of touch probes, as well as image attributes associated with camera pixels. Maps consist of camera images and displays of sensor readings.

At the Servo and Primitive levels, the command output rate is perfectly regular. At the E-Move level and above, the command output rates typically are irregular because they are event driven.

Section 3. Organization and Timing in the RCS Hierarchy

Figure 4 summarizes the relationship in an RCS-4 system between the organizational hierarchy that is defined by the command tree, the computational hierarchy that is defined along each chain of command, and the behavioral hierarchy that is produced in state-space as a function of time. The behavioral hierarchy consists of state/time trajectories that are produced by computational modules executing tasks in real-time.

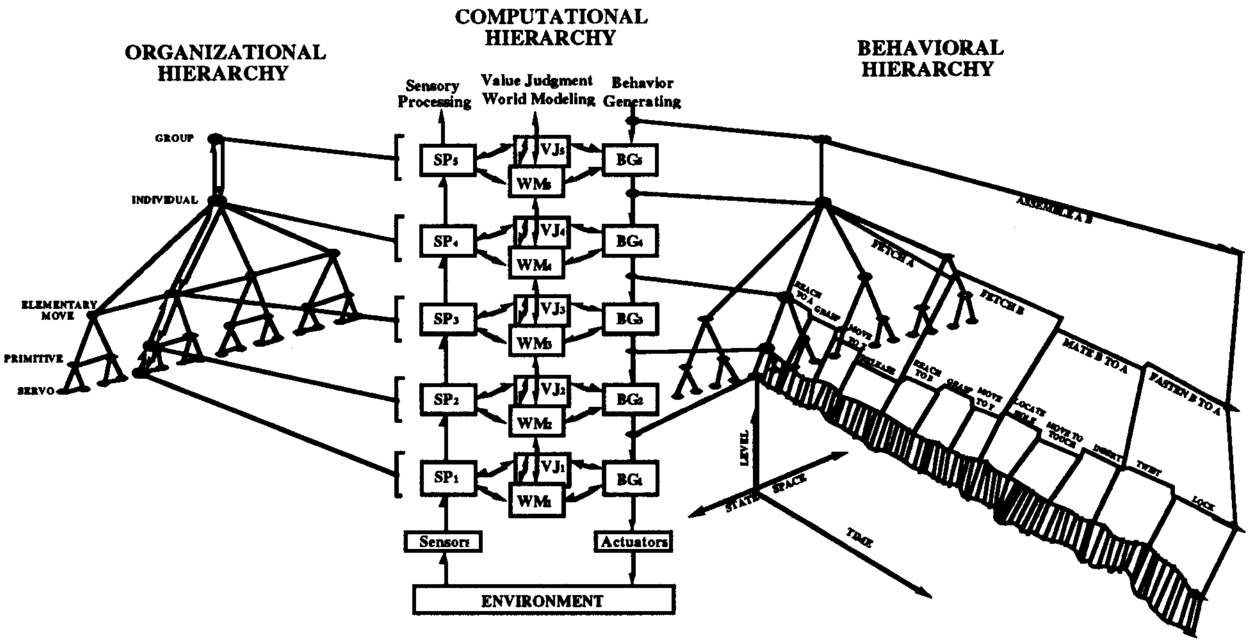

Figure 4. The relationship in RCS-4 between the organizational hierarchy of subsystems, the computational hierarchy of computing modules, and behavioral hierarchy of state trajectories that result as the computing modules execute tasks.

Levels in the RCS command hierarchy are defined by temporal and spatial decomposition of goals and tasks into levels of resolution, as well as by spatial and temporal integration of sensory data into levels of aggregation. Temporal resolution is manifested in terms of loop bandwidth, sampling rate, and state-change intervals. Temporal span is measured in length of historical traces and planning horizons. Spatial resolution is manifested in the resolution of maps and grouping of elements in subsystems. Spatial span is measured in range of maps and the span of control.

Figure 5 is a timing diagram that illustrates the temporal relationships in a RCS hierarchy containing seven levels of task decomposition and sensory processing. The sampling rate, the command update rate, the rate of subtask completion, and the rate of subgoal events, increases at the lower levels of the hierarchy, and decreases at upper levels of the hierarchy. The particular numbers shown in Figure 5 simply illustrate the relative timing between levels. Specific timing requirements are dependent on specific machines and applications.

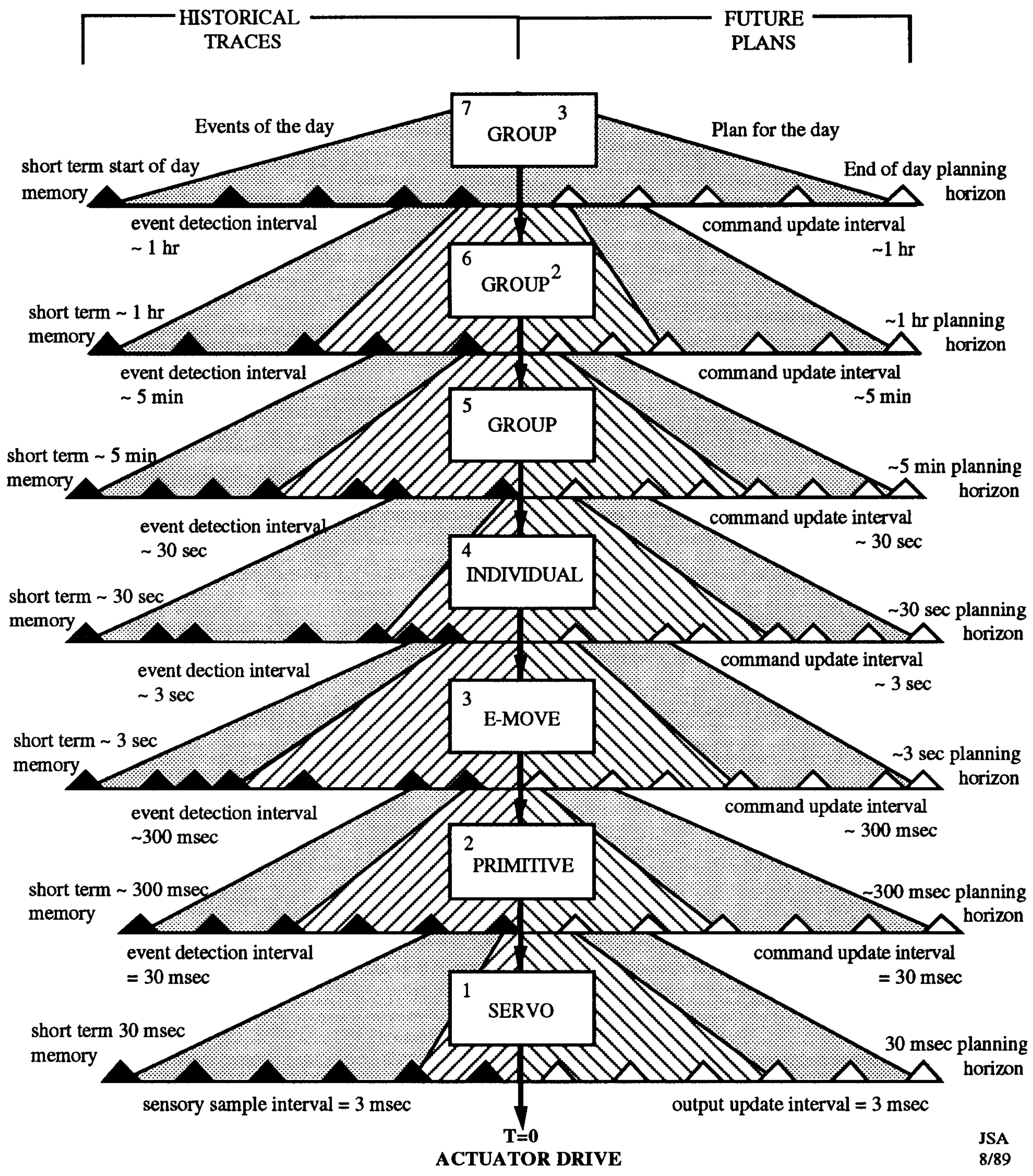
HISTORICAL TRACES
FUTURE PLANS
7 GROUP 3
Events of the day
Plan for the day
short term start of day memory
End of day planning horizon
event detection interval ~ 1 hr
command update interval ~1 hr
6 GROUP 2
short term ~ 1 hr memory
~1 hr planning horizon
event detection interval ~ 5 min
command update interval ~5 min
5 GROUP
short term ~ 5 min memory
~5 min planning horizon
event detection interval ~ 30 sec
command update interval ~ 30 sec
4 INDIVIDUAL
short term ~ 30 sec memory
~30 sec planning horizon
event dection interval ~ 3 sec
command update interval ~ 3 sec
3 E-MOVE
short term ~ 3 sec memory
~3 sec planning horizon
event detection interval ~300 msec
command update interval ~ 300 msec
2 PRIMITIVE
short term ~ 300 msec memory
~300 msec planning horizon
event detection interval = 30 msec
command update interval = 30 msec
1 SERVO
short term 30 msec memory
30 msec planning horizon
sensory sample interval = 3 msec
output update interval = 3 msec
T=0
ACTUATOR DRIVE
JSA
8/89

Figure 5. A timing diagram illustrating the temporal flow of activity in the task decomposition and sensory processing systems.

A fundamental principle of RCS-3 and 4 is that planning goes on simultaneously and continuously at all levels. At each level, planners periodically generate plans containing, on average, about ten steps. At each successive level, the first one or two steps in the current plan from the level above are further decomposed into subplans with an order of magnitude more resolution, and an order of magnitude less span in time and space. As a result, the average period between task commands decreases by about an order of magnitude at each lower level.

Replanning is done either at cyclic intervals, or whenever emergency conditions arise. The cyclic replanning interval may be as short as the average command update interval at each level, or about about ten percent of the planning horizon at each level. This allows the real-time planner to generate a new plan about as often as the executor generates a new output command. Less frequent replanning will reduce the computational speed requirements for real-time planning, but may also reduce control system effectiveness. Emergency replanning begins immediately upon the detection of an emergency condition.

Task execution also goes on simultaneously and continuously at all levels. At each level, executors periodically sample feedback, compare it with the current plan, and issue output commands to correct for deviations between plans and observations. At each sampling period feedback is tested for emergency conditions, and preplanned emergency recovery routines are invoked immediately upon detection of any emergency condition. This allows the system to react to emergencies within a single control cycle with preplanned recovery routines that bridge the temporal gap between when an emergency is detected and when the appropriate planners have new plans ready for execution.

As can be seen in Figure 5, there exists a duality between the task decomposition and the sensory processing hierarchies. At each hierarchical level a task can be decomposed into a set of subtasks at the next lower level. At each level a sensory event can be composed from a sequence of events at the next lower level, a recognized entity can be composed from a group of subentities at the next lower level. At each level, the sensory processing modules look back into the past about as far the planner modules look forward into the future. At each level, future plans have about the same detail as historical traces.

Section 4. Task Decomposition

Each behavior generation (BG), or task decomposition (TD), module at each level consists of three sublevels: Job Assignment, Planning, and Execution.

The Job Assignment sublevel—JA submodule

The JA submodule is responsible for spatial task decomposition. It partitions the input task command into N spatially distinct jobs to be performed by N physically distinct subsystems, where N is the number of subsystems currently assigned to the TD module.

The JA submodule is also responsible for assigning tools and allocating physical resources (such as arms, hands, sensors, tools, and materials) to each of its subordinate subsystems for their use in performing their assigned jobs.

The Planner sublevel—PL(j) submodules, j = 1, 2, . . , N

For each of the N subsystems, there exists a planner submodule PL(j). Each planner submodule is responsible for decomposing the job assigned to its subsystem into a temporal sequence of planned subtasks. Each planner submodule is also responsible for resolving conflicts and mutual constraints between hypothesized plans of submodules.

The Executor sublevel—EX(j) submodules

There is an executor EX(j) for each planner PL(j). The executor submodules are responsible for successfully executing the plan state-graphs generated by their respective planners. When an executor is informed by the world model that a subtask in its current plan is successfully completed, the executor steps to the next subtask in that plan. When all the subtasks in the current plan are successfully executed, the executor steps to the first subtask in the next plan. If the feedback indicates the failure of a planned subtask, the executor branches immediately to a preplanned emergency subtask. Its planner simultaneously begins work selecting or generating an error recovery sequence which can be substituted for the former plan which failed. Output subcommands produced by executors at level i become input commands to job assignment submodules in TD modules at level i-1.

Planners PL(j) constantly operate in the future, each generating a plan to the end of its planning horizon. The executors EX(j) always operate in the present, at time t=0, constantly monitoring the current state of the world reported by feedback from the world model.

At each level, each executor submodule closes a reflex arc, or servo loop, and the executor submodules at the various hierarchical levels form a set of nested servo loops. The executor loop bandwidth decreases about an order of magnitude at each higher level. Each level has a dominant frequency of execution. The actual frequency is dictated by the physical equipment and the application.

Task Knowledge

Fundamental to task decomposition is the representation and use of task knowledge. A task is a piece of work to be done, or an activity to be performed. For any TD or BG module, there exists a set of tasks that that module knows how to do. Each task in this set can be assigned a name. The task vocabulary is the set of task names assigned to the set of tasks each TD or BG module is capable of performing.

Knowledge of how to perform a task may be represented in a frame data structure. An example task frame is:

TASKNAME	Name of the task
Goal	Event or condition that successfully terminates the task
Object	Identification of thing to be acted upon
Parameters	Priority Status (e.g., active, waiting, inactive) Timing (e.g., speed, completion time) Coordinate system in which task is expressed Stiffness matrices Tolerances
Agents	Identification of subsystems that will perform the task
Requirements	Feedback information required from the world model during the task Tools, time, resources, and materials needed to perform the task Enabling conditions that must be satisfied to begin or continue the task Disabling conditions that will interrupt or abort the task activity
Procedures	Pre-computed plans or scripts for executing the task Planning algorithms Functions that may be called Emergency procedures for each disabling condition

Figure 6. A typical task frame

The name of the task is a string that identifies the type of activity to be performed. The goal may be a vector that defines an attractor value, set point, or desired state to be achieved by the task. The goal may also be a map, graph, or geometric data structure that defines a desired "to-be" condition of an object, or arrangement of components.

The requirements section includes information required during the task. This may consist of a list of state variables, maps, and/or geometrical data structures that convey actual, or "as-is" conditions that currently exist in the world. Requirements may also include resources, tools, materials, time, and conditions needed for performing the task.

The procedure section contains either a set of pre-computed plans or scripts for decomposing the task, or one or more planning algorithms for generating a plan, or both. For example, the procedure section may contain a set of IF/THEN rules that select a plan appropriate to the "as-is" conditions reported by the world model. Alternatively, the procedure section may contain a planning algorithm that computes the difference between "to-be" and "as-is" conditions. This difference may then be treated as an error that the task planner attempts to reduce, or null through "Means/Ends Analysis" or A* search. Each subsystem planner would then develop a sequence of subtasks designed to minimize its subsystem error over an interval from the present to its planning horizon. In either case, each executor would act as a feedback controller, attempting to servo its respective subsystem to follow its plan. The procedure section also contains emergency procedures that can be executed immediately upon the detection of a disabling condition.

In plans involving concurrent job activity by different subsystems, there may be mutual constraints. For example, a start-event for a subtask activity in one subsystem may depend on the goal-event for a subtask activity in another subsystem. Some tasks may require concurrent and cooperative action by several subsystems. This requires that both planning and execution of subsystem plans be coordinated. (The reader should not infer from this discussion or others throughout this paper, that all these difficult problems have been solved, at least not for the general case. Much remains unknown that will require extensive further research. The RCS architecture simply provides a framework wherein each of these problems can be explicitly represented and input/output interfaces can be defined. In several RCS designs, human operators are an integral part of some computational nodes, so that tasks that cannot yet be done automatically are done interactively by humans. In the NIST Automated Deburring and Chamfering System workstation, for example, the tasks of the fixturing subsystem are performed by a human operator.)

The library of task frames that reside in each TD module define the capability of the TD module. The names of the task frames in the library define the set of task commands that TD module will accept. There, of course, may be several alternative ways that a task can be accomplished. Alternative task or job decompositions can be represented by an AND/OR graph in the procedure section of the task frame.

The agents, requirements, and procedures in the task frame specify for the TD module "how to do" commanded tasks. This information is a-priori resident in the task frame library of the TD module. The goal, object, and parameters specify “what to do”, “on what object”, “when”, “how fast”, etc. Figure 7 illustrates the instantiation of a task frame residing in the TD module library by a task command. When a TD module inputs a task command, it searches its library of task frames to find a task name that matches the command name. Once a match is found, the goal, object, and parameter attributes from the command are transferred into the task frame. This activates the task frame, and as soon as the requirements listed in the task frame are met, the TD module can begin executing the task plan that carries out the job of task decomposition.

Task knowledge is typically difficult to discover, but once known, can be readily used and duplicated. For example, the proper way to mill a pocket, drill a hole, or fixture a part may be difficult to derive from first principles. However, once such knowledge is known and represented in a task frame, it is relatively easy to transform into executable code.

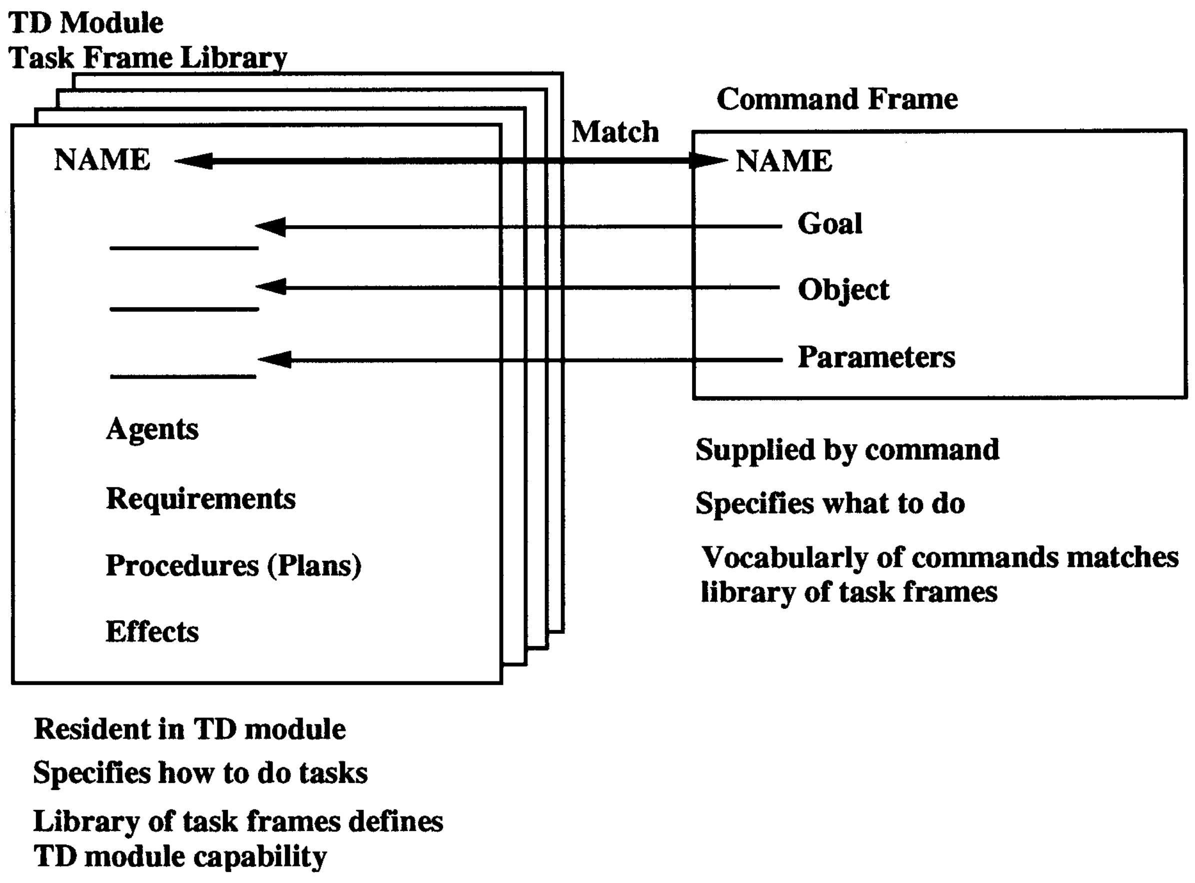

Figure 7. The relationship between commands and task frames.

Section 5. World Modeling

The world model is an intelligent system's internal representation of the external world. It is the system's best estimate of objective reality. It acts as a buffer between sensory processing and task decomposition. This enables the task decomposition system to act on information that may not be immediately or directly observable by sensors, and enables the sensory processing system to do recursive estimation based on a number of past sensory measurements. The world model is hierarchically structured and distributed such that there is a world model (WM) module with Knowledge Database (KD) in each node at every level of the RCS control hierarchy. The KD forms a passive, hierarchically structured, distributed data store. The WM provides the operations that act upon the KD. At each level, the WM and value judgment (VJ) modules perform the functions illustrated in Figure 8 and described below.

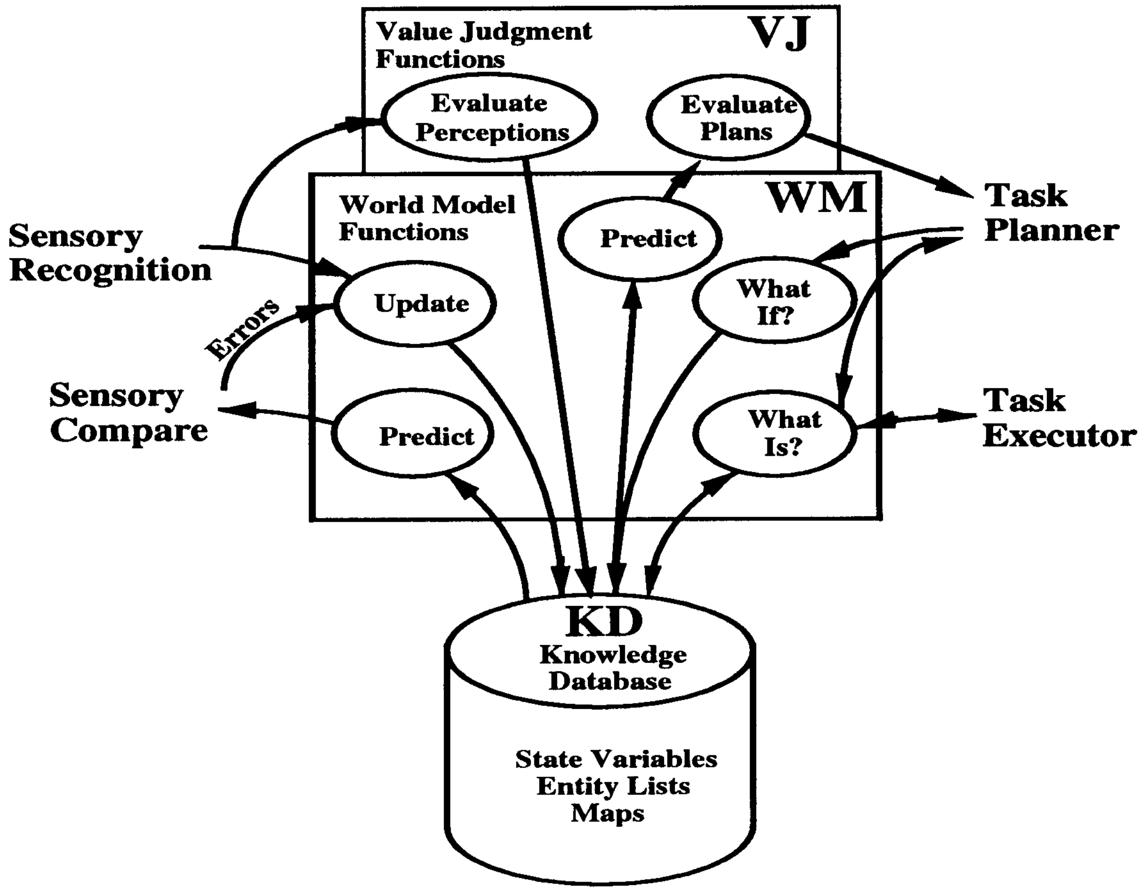

Figure 8. Functions performed by the WM module. 1) Update the knowledge database with recognized entities. 2) Predict sensory data. 3) Answer "What is?" queries from task executor and return current state of world. 4) Answer"What if?" queries from task planner and predict results for evaluation.

1) WM modules maintain the KD knowledge database, keeping it current and consistent. In this role, the WM modules perform the functions of a database management system. They update KD state estimates based on correlations and differences between world model predictions and sensory observations at each hierarchical level. The WM modules enter newly recognized entities, states, and events into the KD database, and delete entities and states determined by the sensory processing modules to no longer exist in the external world. The WM modules also enter estimates, generated by the VJ modules, of the reliability of KD state variables. Believability or confidence factors are assigned to many types of state variables.

2) WM modules generate predictions of expected sensory input for use by the appropriate sensory processing SP modules. In this role, a WM module performs the functions of a graphics engine, or state predictor, generating predictions that enable the sensory processing system to perform correlation and predictive filtering. WM predictions are based on the state of the task and estimated states of the external world. For example in vision, a WM module may use the information in an object frame to generate predicted images which can be compared pixel by pixel, or entity by entity, with observed images.

3) WM modules answer "What is?" questions asked by the planners and executors in corresponding BG modules. In this role, the WM modules perform the function of database query processors, question answering systems, or data servers. World model estimates of the current state of the world are used by BG or TD module planners as a starting point for planning. Current state estimates are also used by BG or TD module executors for servoing and branching on conditions.

4) WM modules answer "What if?" questions asked by the planners in the corresponding level BG or TD modules. In this role, the WM modules perform the function of simulation by generating expected status resulting from actions hypothesized by the planners. Results predicted by WM simulations are sent to value judgment VJ modules for evaluation. For each BG or TD hypothesized action, a WM prediction is generated, and a VJ evaluation is returned to the BG or TD planner. This BG-WM-VJ loop enables BG planners to select the sequence of hypothesized actions producing the best evaluation as the plan to be executed.

The Knowledge Database

The world model knowledge database (KD) includes both a-priori information which is available to the intelligent system before action begins, and a-posterior knowledge which is gained from sensing the environment as action proceeds. The KD represents information about space, time, entities, events, states of the world, and laws of nature. Knowledge about space is represented in maps. Knowledge about entities, events, and states is represented in lists and frames. Knowledge about the laws of physics, chemistry, optics, and the rules of logic and mathematics is represented in the WM functions that generate predictions and simulate results of hypothetical actions. Laws of nature may be represented as formulae, or as IF/THEN rules of what happens under certain situations, such as when things are pushed, thrown, or dropped.

The world model also includes knowledge about the intelligent system itself, such as the values assigned to goal priorities, attribute values assigned to objects, and events; parameters defining kinematic and dynamic models of robot arms or machine tool stages; state variables describing internal pressure, temperature, clocks, fuel levels, body fluid chemistry; the state of all the currently executing processes in each of the BG, SP, WM, and VJ modules; etc.

The correctness and consistency of world model knowledge is verified by sensors and sensory processing SP mechanisms that measure differences between world model predictions and sensory observations. These differences may be used by recursive estimation algorithms to keep the world model state variables the best estimates of the state of the world. Attention algorithms may be used to limit the number of state variables that must be kept up-to-date and any one time.

Information in the world model knowledge database may be organized as state variables, system parameters, maps, and entity frames.

State variables

State variables define the current value of entity and system attributes. A state variable may define the state of a clock, the position, orientation, and velocity of a gripper, the position, velocity, and torque of an actuator, or the state of a computing module.

System parameters

System parameters define the kinematic and dynamic characteristics of the system being controlled, e.g., the inertia of objects, machines, and tools. System parameters may also define coordinate transforms necessary to transform commands and sensory information from one working coordinate system to another.

Entity Frames

An entity frame is a symbolic list structure, in which the ENTITY NAME is the list head, and in which knowledge about the entity is stored as attribute-value pairs (or attribute-list pairs). The world model contains a list of all the entities that the intelligent system knows about. A subset of this list is the set of current entities known to be present in any given situation. A subset of the list of current entities is the set of entities of attention. These are the entities that the system is currently acting upon, or is planning to act upon momentarily.

There are two types of entities: generic and specific. A generic entity is an example of a class of entities. A generic entity frame contains the attributes of its class. A specific entity is a particular instance of an entity. A specific entity frame inherits the attributes of its class.

Different levels of entities exist at different levels of the hierarchy. At level 1, entity frames describe points; at level 2, entity frames describe lines and vertices; at level 3, they describe surfaces; at level four, objects; and at level 5, groups; at level 6 and above, entity frames describe higher order groups.

An example of an entity frame is shown in Figure 9.

ENTITY NAME	-- part id#, lot#, etc.
kind	-- model#
type	-- generic or specific
level	-- point, line, surface, object, group
position	-- map location of center of mass (time, uncertainty)
orientation	-- coordinate axes directions (time, uncertainty)
coordinates	-- coordinate system of map
dynamics	-- velocity,acceleration (time, uncertainty)
trajectory	-- sequence of positions (time, uncertainty)
geometry	-- center of gravity (uncertainty)
	axis of symmetry (uncertainty)
	size (uncertainty)
	boundaries (uncertainty)
subentities	-- pointers to lower level entities that make up named entity
parent entity	-- pointer to higher level entity of which named entity is part
properties	-- mass, color, hardness, smoothness, etc.
value	-- sunk cost, value at completion
due date	-- date required

Figure 9. An example entity frame

Maps

Maps describe the distribution of entities in space. Each point, or pixel, on a map may have a pointer that points to the name of the entity that projects to that point on the map. A pixel may also have one or more attribute-value pairs. For example, a map pixel may have a brightness, or color (as in an image), or an altitude (as in a topographic map). A map may also be represented by a graph that indicates routes between locations, for example, the routes available to robot carts moving between workstations.

Any specific map is defined in a particular coordinate frame. There are three general types of map coordinate frames that are important: world coordinates, object coordinates, and egospheres. An egosphere is a spherical coordinate system with the intelligent system at the origin and properties of the world are projected onto the surface of the sphere. World, object, and egosphere coordinate frames are discussed more extensively in [1].

Map-Entity Relationships

Map and entity representations may be cross referenced in the world model as shown in Figure 10. For example, each entity frame may contain a set of geometrical and state parameters that enables the world model to project that entity onto a map. The world model can thus compute the set of egosphere or world map pixels covered by an entity. By this means, entity parameters can be inherited by map pixels, and hence entity attributes can be overlaid on maps.

Conversely, each pixel on a map may have pointers to entities covered by that pixel. For example, a pixel may have a pointer to a point entity whose frame contains the projected distance or range to the point covered by that pixel. Each pixel may also have a pointer to a line entity frame indicating the position and orientation of an edge, line, or vertex covered by the pixel. Each pixel may also have a pointer to a surface entity indicating position and orientation of a surface covered by the pixel. Each pixel may have a pointer to an object entity indicating the name of the object covered, and a pointer to a group entity indicating the name of the group covered.

The world model thus provides cross referencing between pixel maps and entity frames. Each level of world modeling can thus predict what objects will look like to sensors, and each level of sensory processing can compare sensory observations with world model predictions. (The reader should not infer from this discussion that such a cross-coupled systems have been fully implemented, or that it is even well understood how to implement such systems for real-time operations. What is described is the kind of cross-coupled system that will be necessary in order to achieve truly intelligent robotic systems. It seems likely that special purpose hardware and firmware will need to be developed. Clearly, much additional research remains to be done before these problems are solved. The RCS reference model architecture simply defines how such processes should be organized and what interfaces need to be defined.

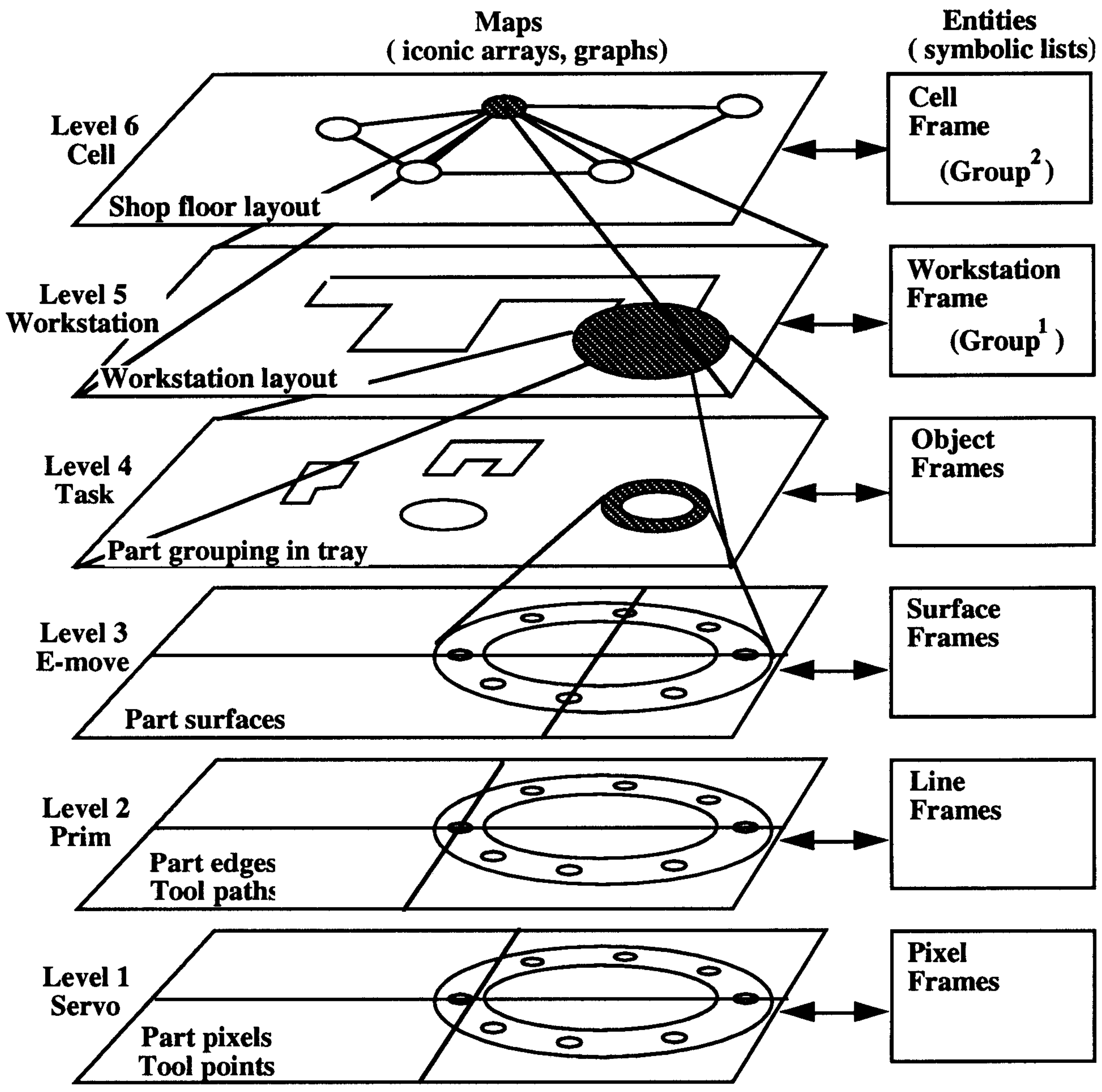

Figure 10. Map-entity relationships in a world model for manufacturing. The world model provides processes by which symbolic entity frames can be transformed into maps, and vice versa.

An example of a knowledge database for an intelligent robot in a manufacturing workstation such as shown in Figure 3 might be the following:

Level 1

State Variables

State clock and sync signals
State vector that defines the best estimate of the position, velocity, and force of each joint actuator.
Estimated time or distance to nearest point entity contact

System parameters

Joint limit margin for each actuator
Gravity compensation matrix
Inertia matrix
Forward and inverse kinematic transform from end-effector to joint coordinates
Forward and inverse transform from sensor egosphere to end-effector egosphere

Entity frames

Point entity frames for entities of attention
Pixel frames for sensor egosphere map pixels

Maps

Sensor egosphere map overlaid with projections of point entities of attention

Level 2

State variables

State clock and sync signals
State vector defining the best estimate of load or tool pose, velocity, force, etc.
Estimated time or distance to nearest linear entity contact
Singularity clearance margins

System parameters

Forward and inverse force and velocity coordinate transform from equipment to end-effector coordinates
Forward and inverse transform from end-effector egosphere to equipment centered inertial egosphere coordinates
Manipulator dynamic model
Load dynamic model
Limits of travel in coordinate system of command
Positions of singularities
Position, velocity, and force limits

Entity frames

Linear entity frames (edges, lines, trajectories, vertices, etc.) for entities of attention

Maps

Tool or end-effector egosphere map overlaid with projection of linear entities of attention and their trajectories (observed and planned)

Level 3

State variables

State clock and sync signals
Best fit trajectories for observed poses, velocities, and forces of load or tool
Estimated time or distance to nearest surface entity contact
Estimated minimum clearance for singularities and obstacle surfaces

System parameters

Forward and inverse transform from equipment centered inertial egosphere coordinates to part coordinates
Manipulator geometry and dynamic model
Load geometry and dynamic model

Limits of travel in coordinate system of command
Position, velocity, and force limits
Cost/benefit function parameters for analysis of hypothesized path plans

Entity frames

Surface entity frames for entities of attention

Maps

Equipment centered inertial egosphere map overlaid with projection of surface entities of attention and their swept volumes (observed and planned)

Level 4

State variables

State clock and sync signals from other equipment
Best estimate observed degree of task completion
State of task enabling and disabling conditions
Predicted results of plan

System parameters

Task enabling and disabling conditions
Forward and inverse transform from equipment centered inertial egosphere to equipment centered world coordinates
Equipment geometry and dynamic model
Part and tool geometry and dynamic model
Limits of travel in coordinate system of command
Cost/benefit function for evaluation of plan results

Entity frames

Object entity frames for objects of attention (trays, fixtures, tool racks, free space, parts, grippers, tools, fasteners, etc.)

Maps

Equipment centered world map overlaid with projections of objects of attention; also overlaid with projections of planned sequence of E-moves

Level 5

State variables

State clock and sync signals from other equipment
Observed degree of task completion
State of task enabling and disabling conditions
Predicted results of hypothesized plan

System parameters

Forward and inverse transforms from workstation to equipment centered world coordinates
Task enabling and disabling conditions
Workstation task timing model
Cost/benefit evaluation function for hypothesized plan results

Entity frames

Workstation equipment entity frames
Group entity frames (objects grouped in trays, buffers, tool racks, fixtures, etc.) for groups of attention

Maps

Workstation centered world map overlaid with projections of equipment entities and group entities of attention; also overlaid with planned sequence of robot task

Additional discussion of RCS world models for robots can be found in [16].

Section 6. Sensory Processing

Sensory processing is the mechanism of perception. Perception is the establishment and maintenance of correspondence between the internal world model and the external real world. The function of sensory processing is to extract information about entities, events, states, and relationships in the external world, so as keep the world model accurate and up to date.

Map Updates

World model maps may be updated by sensory measurement of points, edges, and surfaces. Such information may be derived from vision, touch, sonar, radar, or laser sensors. The most direct method of measuring points, edges, and surfaces is through touch. In the manufacturing environment, touch probes are used by coordinate measuring machines to measure the 3-D position of points. Touch probes on machine tools and tactile sensors on robots can be used for making similar measurements. From such data, sensory processing algorithms can compute the orientation and position of surfaces, the shape of holes, the distance between surfaces, and the dimensions of parts.

Other methods for measuring points, edges, and surfaces include stereo vision, photogrammetry, laser ranging, structured light, image flow, acoustic ranging, and focus-based optical probes. Additional information about surface position and orientation may also be computed from shading, shadows, and texture gradients. Each of these various methods produce different accuracies and have differing computational and operational requirements.

Recognition and Detection

Recognition is the establishment of a one-to-one match, or correspondence, between a real world entity and a world model entity . The process of recognition may proceed top-down, or bottom-up, or both simultaneously. For each entity in the world model, there exists a frame filled with information that can be used to predict attributes of corresponding entities observed in the world. The top-down process of recognition begins by hypothesizing a world model entity and comparing its predicted attributes with those of the observed entity. When the similarities and differences between predictions from the world model and observations from sensory processing are integrated over a space-time window that covers an entity, a matching, or cross-correlation value is computed between the entity and the model. If the correlation value rises above a selected threshold, the entity is said to be recognized. If not, the hypothesized entity is rejected and another tried.

The bottom-up process of recognition consists of applying filters and masks to incoming sensory data, and computing image properties and attributes. These may then be stored in the world model, or compared with the properties and attributes of entities already in the world model. Both top-down and bottom-up processes proceed until a match is found, or the list of world model entities is exhausted. Many perceptual matching processes may operate in parallel at multiple hierarchical levels simultaneously.

If a SP module recognizes a specific entity, the WM at that level updates the attributes in the frame of that specific WM entity with information from the sensory system.

If the SP module fails to recognize a specific entity, but instead achieves a match between the sensory input and a generic world model entity, a new specific WM entity will be created with a frame that initially inherits the features of the generic entity. Slots in the specific entity frame can then be updated with information from the sensory input.

If the SP module fails to recognize either a specific or a generic entity, the WM may create an "unidentified" entity with an empty frame. This may then be filled with information gathered from the sensory input.

When an unidentified entity occurs in the world model, the behavior generation system may (depending on other priorities) select a new goal to <identify the unidentified entity>. This may initiate an exploration task that positions and focuses the sensor systems on the unidentified entity, and possibly even probes and manipulates it, until a world model frame is constructed that adequately describes the entity. The sophistication and complexity of the exploration task depends on task knowledge about exploring things. Such knowledge may be very advanced and include sophisticated tools and procedures, or very primitive. Entities may, of course, simply remain labeled as "unidentified," or “unexplained.”

Detection of events is analogous to recognition of entities. Observed states of the real world are compared with states predicted by the world model. Similarities and differences are integrated over an event space-time window, and a matching, or cross-correlation value is computed between the observed event and the model event. When the cross-correlation value rises above a given threshold, the event is detected.

Sensory Processing Modules

The Sensory Processing (SP) modules are responsible for gathering data from sensors, filtering and integrating this data, and interpreting it. Noise rejection techniques such as Kalman filtering are implemented here, as well as feature extraction, pattern recognition, and image understanding. At the upper levels of the hierarchy, more abstract interpretations of sensory data are performed and inputs from a variety of sensor systems are integrated over space and time into a single interpretation of the the external world.

Each SP module at each level consists of five types of operations: 1) Coordinate transformation, 2) Comparison, 3) Temporal Integration, 4) Spatial Integration, and 5) Recognition/Detection, as shown in Figure 11.

1) Coordinate transformation

Before a world model estimated variable can be compared with an observed sensory variable, the estimated and observed variables must be registered with each other in the same coordinate system. Sensory variables are most often represented in sensor egosphere coordinates. World model estimates may be represented in a variety of coordinate frames, each of which have certain computational advantages. Among these are head egosphere, inertial egosphere, part or tool egosphere, or world coordinates with origins at convenient points.

2) Comparison

Each comparator matches an observed sensory variable (such as an image pixel attribute) with a world model prediction of that variable. The predicted variable may be subtracted from the observed, or multiplied by it, to obtain a measure of similarity (correlation) and difference (variance) between the observed variable and the predicted variable. Similarities indicate the degree to which the WM predictions are correct, and hence are a measure of the correspondence between the world model and reality. Differences indicate a lack of correspondence between world model predictions and sensory observations. Differences imply that either the sensor data or world model is incorrect. Difference images from the comparator go three places:

a) They are returned directly to the WM for real-time local pixel attribute updates. This produces a tight feedback loop whereby the world model predicted image becomes an array of recursive state-estimations. Difference images are error signals used to make each pixel attribute in the predicted image a "best estimate" of the corresponding pixel attribute in the current sensory input.

b) Difference images are also transmitted upward to spatial and temporal integrators where they are integrated over time and space in order to recognize and detect entity attributes. This integration constitutes a summation, correlation, or "chunking", of sensory data into entities. At each level, lower order entities are "chunked" into higher order entities, i.e. points are chunked into lines, lines into surfaces, surfaces into objects, objects into groups, etc.

c) Difference images also are transmitted to the VJ module at the same level where statistical parameters are computed so as to assign uncertainty and believability factors to estimates of pixel entity attributes.

3) Temporal integration

Temporal integration accumulates similarities and differences between predictions and observations over intervals of time. Temporal integration operating just on sensory data can produce a summary, such as a total, or average, of sensory information over the given time window. Temporal integrators operating on the similarity and difference values computed by comparators may produce temporal cross-correlation and covariance functions between the model and the observed data. These correlation and covariance functions are measures of how well the dynamic properties of the world model entity match those of the real world entity. The boundaries of the temporal integration window may be derived from world model prediction of event durations, or from behavior generation parameters such as sensor fixation periods.

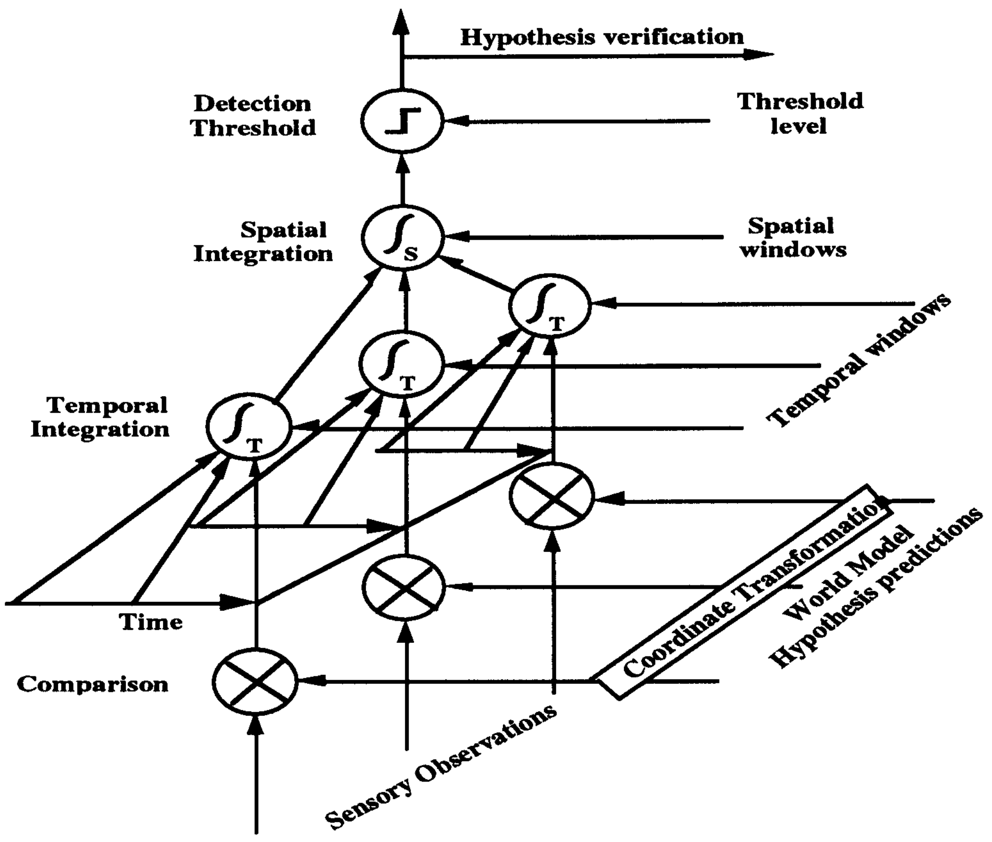

Figure 11. Each sensory processing SP module consists of: 1) a set of coordinate transformers, 2) a set of comparators that compare sensory observations with world model predictions, 3) a set of temporal integrators that integrate similarities and differences, 4) a set of spatial integrators that fuse information from different sensory data streams, and 5) a set of threshold detectors that recognize entities and detect events

4) Spatial integration

Spatial integration accumulates similarities and differences between predictions and observations over regions of space. This produces spatial cross-correlation or convolution functions between the model and the observed data. Spatial integration summarizes sensory information from multiple sources at a single point in time. It determines whether the geometric properties of a world model entity match those of a real world entity. For example, the product of an edge operator and an input image may be integrated over the area of the operator to obtain the correlation between the image and the edge operator over the region covered by the filter. The limits of the spatial integration window may be determined by world model predictions of entity size and shape. In some cases, the order of temporal and spatial integration may be reversed, or interleaved.

5) Recognition/Detection threshold

When the spatio-temporal correlation function exceeds some threshold, object recognition (or event detection) occurs. For example, if the spatio-temporal summation over the area of an edge operator exceeds threshold, an edge is said to be recognized at the center of the area.

The Sensory Processing Hierarchy

It has long been known that sensory processing occurs in a hierarchy of processing modules, and that perception proceeds by "chunking," i.e., by recognizing patterns, groups, strings, or clusters of points at one level as a single feature, or point in a higher level, more abstract space. It also has been observed that this chunking process proceeds by about an order of magnitude per level, both spatially and temporally [17,18]. Thus, at each level in the proposed architecture, SP modules integrate, or chunk, information over space and time by about an order of magnitude.

In order to facilitate the comparison of predicted and observed variables, there must be a correspondence between the form of the sensory data being processed at each level of the sensory processing hierarchy and the form of the information stored in the world model knowledge base at that level. This is illustrated in Figure 10 where:

Level 1 of the sensory processing hierarchy compares point measurements with projected point entities from the world model. Differences are used to correct the estimated point entities. Similarities between observed and predicted point entities are integrated into line entities.

Level 2 of the sensory processing hierarchy compares observed line entities with projected line entities from the world model. Differences are used to correct the estimated line entities, and similarities are integrated into surface entities.

Level 3 compares observed surfaces with predicted surfaces. Differences are used to correct the predictions, and similarities are integrated to recognize objects.

Level 4 compares observed objects with predicted objects. Differences are used to correct the world model, and similarities are integrated into groups.

Level 5 compares observed and predicted group characteristics, updates the world model, and integrates information into larger groups. And so on.

Further discussion of the sensory processing hierarchy appears in [1].

Section 7. Value Judgments

Value judgments provide the criteria for making intelligent choices. Value judgments evaluate the costs, risks, and benefits of plans and actions, and the desirability, attractiveness, and uncertainty of objects and events. Value judgment modules produce evaluations that can be represented as value state-variables. These can be assigned to the attribute lists in entity frames of objects, persons, events, situations, and regions of space. They can be assigned to map pixels and can become map overlays. Value state-variables can thus label objects, situations, or places as good or bad, friendly or hostile, attractive or repulsive.

Value judgment algorithms may evaluate risk and compute the level of uncertainty in the recognition of entities and the detection of events. Knowledge in the world model can thus be labeled as reliable or uncertain.

Value judgments can evaluate events as important or trivial. The utilization of memory can be optimized by storing only those events and situations evaluated as "important". Those events labeled as "trivial" can safely be forgotten and cleared from memory. Intelligent machines that include capabilities for learning require value judgment functions to indicate what to "reward" and what to "punish."

Cost/benefit value judgment algorithms can be used to evaluate plans or steer task execution so as to minimize cost and maximize benefits. Value state-variables can be assigned to the attribute lists of plans and actions in task frames to label tasks and plans as good or bad, costly or inexpensive, risky or safe, with high or low expected payoff. Priorities are value state-variables that provide estimates of importance. Priorities can be assigned to task frames so that planners and executors can decide what to do first, how much effort to spend, how much risk is prudent, and how much cost is acceptable, for each task. Value state-variables can thus be used by the behavior generation modules both for planning and executing actions. They provide criteria for making decisions about which coarse of action to take. Priorities may also determine the degree of alertness, or tactics chosen for planning tasks in a hostile environment. A priority on safety may produce conservative tactics. A priority on aggression may produce an attack on an enemy. Such priorities are typically input from a human commander.

Intelligent systems designed for military purposes typically require value judgments labeling objects in the world model as "friend" or "foe." If an object is labeled "friend" it should be defended or assisted. If it is labeled "foe" it should be attacked or avoided. The computation of "friend or foe" may be accomplished by signal analysis or pattern recognition. This computation may be very difficult, or trivially simple. For example, in current battlefield situations, friendly systems typically carry transponders that actively emit signals, or modulate reflected energy in a distinctive way that makes recognition of "friend" easy, even in smoke or fog or at night. Often, anything else is assumed to be "foe".

VJ Modules

Value state-variables may be assigned by a human programmer or operator, or they may be computed by value judgment functions residing in VJ modules. Inputs to VJ modules describe entities, events, situations, and states. Inputs may also include sensor signal-to-noise ratio, variance between sensory observations and world model predictions, degree of success in goal achievement, and reward or punishment signals from a variety of sources. VJ value judgment functions may compute measures of cost, risk, and benefit. VJ outputs are value state-variables that may be assigned to objects, events, or plans.

The VJ value judgment mechanism can typically be defined as a mathematical or logical function of the form

$$\mathbf{E} = \mathbf{V}(\mathbf{S})$$

where **E** is an output vector of value state-variables

V is a value judgment function that computes **E** given **S**

S is an input state vector defining conditions in the world model, including the self.

The components of **S** are entity attributes describing states of tasks, objects, events, or regions of space. These may be derived either from processed sensory information, or from the world model. The value judgment function **V** in the VJ module computes a numerical scaler value (i.e. an evaluation) for each component of **E** as a function of the input state vector **S**. The components of **E** may be assigned to attributes in the world model frame of various entities, events, or states. **E** is a time dependent vector.

Further discussion of value judgments can be found in [1].

Conclusion

RCS is a reference model architecture that defines the types of functions that are required in a real-time intelligent control system, and how these functions are related to each other. RCS is not a system design, nor is it a specification of how to implement specific systems. It has, nevertheless, been found useful by many researchers and engineers for designing intelligent machine systems.

Systems based on the RCS architecture have been designed and more or less implemented for a wide variety of applications that include loading and unloading of parts and tools in machine tools, controlling machining workstations, performing robotic deburring and chamfering, and controlling space station telerobots, multiple autonomous undersea vehicles, unmanned land vehicles, coal mining automation systems, postal service mail handling systems, and submarine operational automation systems. Software developers accustomed to using RCS for building control systems have found it provides a structured, even recursive, approach to design that makes it possible to reuse a great deal of software, not only within a single system, but between systems designed for significantly different applications.

Critics of RCS have objected to its rigidity, claiming "it forces a system to be built in a certain structure, even if it can be built more effectively in a different architecture". This, however, misses the point of a reference model architecture. A reference model architecture is a canonical form, not a system design specification. Reference models can be implemented in a variety of ways. For example, RCS could be implemented with neural nets (at least in principle). Many of the lower level features can be implemented by finite-state-machines and motion controllers. Higher-level functions can be implemented by expert systems, lisp machines, transputers, connection machines, or special purpose processing engines. The RCS reference model architecture combines real-time motion planning and control with high level task planning, problem solving, world modeling, recursive state estimation, tactile and visual image processing, and acoustic signature analysis. In fact, the evolution of the RCS concept has been driven by an effort to include the best properties and capabilities of most, if not all, the intelligent control systems currently known in the literature, from subsumption to SOAR [19], from blackboards [20] to object-oriented programming [21].

However, the apparent completeness of the canonical form of RCS should not be taken to mean that all of the problems are solved in each of these areas, and all the issues are resolved. Far from it. The enormous complexity of the RCS architecture should lead one to appreciate how difficult the task of creating intelligent systems is going to be. Not so long ago, respected people were predicting that intelligent robots would be able to duplicate human performance before the end of this century. A correct understanding of the RCS reference model should lead one to realize that the creation of intelligence may be more complex and scientifically challenging than controlling fusion or analyzing the genome, and possibly more rewarding as well.

Current research efforts at NIST are directed towards understanding how to use the RCS reference model as a design tool. An RCS design paradigm begins with analysis of tasks and goals, then develops control algorithms, data structures, and sensory information necessary to perform the tasks and accomplish the goals. Attempts are underway to develop this approach into a formal methodology for engineering real-time intelligent control systems and analyzing their performance.

REFERENCES

[1] J.S. Albus, “Outline for a Theory of Intelligence,” IEEE Trans. on Systems, Man, and Cybernetics, Vol.21, No.3, May/June 1991.

[2] 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.

[3] Barbera, A.J., J.S. Albus, M.L. Fitzgerald, "Hierarchical Control of Robots Using Microcomputers," Proceedings of the 9th International Symposium on Industrial Robots, Washington, DC, March 1979.

[4] J.S. Albus, “A Theory of Cerebellar Function,” Mathematical Biosciences, Vol. 10, pgs. 25-61, 1971.

[5] J.S. Albus, “A New Approach to Manipulator Control : The Cerebellar Model Articulation Controller (CMAC),” *Transactions ASME,* September 1975.

[6] N. Tinbergen, *The Study of Instinct,* Clarendon, Oxford, 1951.

[7] R. Brooks, “A Robust Layered Control System for a Mobile Robot,” *IEEE Journal of Robotics and Automation*, vol. RA-2, 1, March, 1986.

[8] J.A. Simpson, R.J. Hocken, and J.S. Albus, "The Automated Manufacturing Research Facility of the National Bureau of Standards," *Journal of Manufacturing Systems*, Vol. 1, No. 1, 1983.

[9] J.S. Albus, C. McLean, A.J. Barbera, and M.L. Fitzgerald, "An Architecture for Real-Time Sensory-Interactive Control of Robots in a Manufacturing Environment," *4th IFAC/IFIP Symposium on Information Control Problems in Manufacturing Technology*, Gaithersburg, MD, October 1982.

[10] E. W. Kent and J.S. Albus, “Servoed World Models as Interfaces Between Robot Control Systems and Sensory Data,” Robotica, Vol. 2, No.1, January 1984

[11] H.G. McCain, R.D. Kilmer, S. Szabo, A. Abrishamian, "A Hierarchically Controlled Autonomous Robot for Heavy Payload Military Field Applications," Proceedings of the International Conference on Intelligent Autonomous Systems, Amsterdam, The Netherlands, December 8-11, 1986.

[12] J.S. Albus, "System Description and Design Architecture for Multiple Autonomous Undersea Vehicles," National Institute of Standards and Technology, Technical Report 1251, Gaithersburg, MD, September 1988.

[13] J.S. Albus, H.G. McCain, and R. Lumia, "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)," National Institute of Standards and Technology, Technical Report 1235, Gaithersburg, MD , April 1989.

[14] G.N. Saridis, “Foundations of the Theory of Intelligent Controls,” *IEEE Workshop on Intelligent Control,* 1985.

[15] G.E. Pugh and G.L. Lucas, “Applications of Value-Driven Decision Theory to the Control and Coordination of Advanced Tactical Air Control Systems,” Decision-Science Applications, Inc., Report No. 218, April 1980.

[16] L.R. Kelmar, R. Lumia, "World Modeling for Sensory Interactive Trajectory Generation," Proceedings of the Third International Symposium on Robotics and Manufacturing, Vancouver, BC, Canada, July 18-20, 1990.

[17] G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *The Psychological Review*, 63, pp.71-97, 1956.

[18] A. Meystel, "Theoretical Foundations of Planning and Navigation for Autonomous Robots," *International Journal of Intelligent Systems*, 2, 73-128, 1987.

[19] J.E. Laird, A. Newell, and P. Rosenbloom, "SOAR: An Architecture for General Intelligence," *Artificial Intelligence*, vol. 33, 1987, pp. 1-64.

[20] B. Hayes-Roth, "A Blackboard Architecture for Control," *Artificial Intelligence*, pgs. 252-321, 1985.

[21] P. Coad and E. Yourdon, *Object-Oriented Analysis*, Yourdon Press, Englewood Cliffs, New Jersey, 1991.