An Intelligent Systems Architecture for Manufacturing (ISAM)
A Reference Model Architecture for Intelligent Manufacturing Systems

Abstract

The Intelligent Systems Architecture for Manufacturing (ISAM) addresses the application of intelligent systems to the manufacturing enterprise at three degrees of abstraction: 1) a conceptual framework for developing metrics, standards, and performance measures, 2) a reference model architecture for conceptual design of intelligent systems, and 3) a set of engineering guidelines for the implementation of manufacturing applications.

ISAM consists of a hierarchically layered set of intelligent processing nodes organized as a nested series of control loops. In each node, tasks are decomposed, plans are generated, world models are maintained, feedback from sensors is processed, and control loops are closed. In each layer, nodes have a characteristic span of control, with a characteristic planning horizon, and corresponding level of detail in space and time. Nodes at the higher levels deal with corporate and production management, while nodes at lower levels deal with machine coordination and process control. ISAM integrates and distributes deliberative planning and reactive control functions throughout the entire hierarchical architecture, at all levels, with different spatial and temporal scales at each level.

Contents

Introduction

The ISAM model addresses the application of intelligent systems to the manufacturing enterprise at three degrees of abstraction:

1. At a high degree of abstraction, ISAM provides a conceptual framework for the entire manufacturing enterprise, including machines, processes, tools, facilities, computers, software, and human beings operating over time on materials to produce products.

2. At a middle degree of abstraction, ISAM provides a reference model architecture for the design of manufacturing systems and software. ISAM addresses the integration of perception, cognition, knowledge representation, simulation, reasoning, and planning with reactive control. ISAM applies to any modular structure with well defined interfaces.

3. At a low degree of abstraction, ISAM provides engineering guidelines for implementing specific instances of manufacturing systems, including machine tools, robots, inspection machines, and material handling systems organized into workstations, cells, shops, and factories.

At all levels of abstraction, ISAM provides a framework for developing metrics, information exchange standards, and performance measures.

I. ISAM as a Conceptual Framework

The ISAM conceptual framework spans the entire range of manufacturing operations, from those that take place over time periods of microseconds and distances of microns to those that take place over time periods of years and distances of many kilometers. The ISAM model is intended to allow for the representation of activities that range from detailed dynamic analysis of a single actuator in a single machine to the combined activity of thousands of machines and human beings in hundreds of plants comprising the operations of a multinational corporation.

In order to span this wide range of activities, ISAM adopts a hierarchical layering, with different range and resolution in time and space at each level. It defines functional units at each level within the enterprise such that each unit can view its particular responsibilities and priorities at a level of spatial and temporal resolution that is understandable and manageable to itself. At any level within the hierarchy, functional units receive goals and priorities from above, and observe situations in the environment below. In each functional unit at each level, there are decisions to be made, plans to be formulated, and actions to be taken that affect peers at the same level and subordinates at a level below. In each unit, information must be processed, situations analyzed, and status reported to peers at the same level and a supervisor above. Each functional unit has access to a model of the world that enables situation analysis, decision making, and planning to be carried out despite the uncertainties and noise that exists in the real world.

At each level, there are values (sometimes not explicitly stated) that set priorities and guide decision making.

For example, every manufacturing enterprise has a top level where there is typically a board of directors. This is where corporate policy is set and strategic goals are established. The board decides what kind of business the corporation is to engage in, what types of products will be produced, what kind of business and labor policies will be pursued, and what are the values that determine priorities and shape corporate decisions. These values typically include criteria of success, which may be defined in terms of profits, market share, stock prices, etc. In successful corporations, there typically are well defined corporate goals with a time table and a business plan for achieving them. The CEO provides the highest level executor function.

At the top of the manufacturing hierarchy, categorical imperatives such as <make a profit> are decomposed into prioritized tasks and goals that determine behavior throughout the enterprise. At intermediate levels, tasks with goals and priorities are received from the level above, and subtasks with subgoals and priorities are output to the level below.

Typically, the corporation is organized into management units such that each unit of management consists of a group of intelligent agents (humans or machines), each of which possesses a particular combination of knowledge, skills, and abilities, and each of which has a job description that defines duties and responsibilities. Each management unit accepts tasks from higher level management units, and issues subtasks to subordinate management units. Within each management unit, agents are given job assignments and allocated resources with which to carry out their assignments. Within each management unit, intelligent agents schedule their activities so as to achieve the goals of the jobs assigned to them. Each agent is expected to make local executive decisions in order to keep operations on schedule by solving problems and compensating for minor unexpected events. Major problems that cannot be handled locally are referred up through the hierarchy to higher levels of management.

Typically, each unit of management has a model of the environment in which it functions. This world model is a representation of the state of the environment, the entities that exist in the environment, the events that take place in the environment, the attributes and behavior of entities and events, and the relationships among them. The world model also typically includes a set of rules that describes how the environment is expected to behave under various conditions. Each unit of management also has access to sources of information that keep its world model current and accurate. Finally, each management unit has a set of values, priorities, or cost functions, that it uses to analyze the state of the world and evaluate plans for future actions.

Every enterprise also has a bottom level, where physical actions take place and sensors measure phenomena in the environment. Typically the bottom level actuators and sensors are grouped into operational units or subsystems, and those subsystems are controlled by subsystem controllers. Subsystems are typically grouped into machines,

machines into workstations, workstations into cells, cells into shops, shops into factories, etc. At each level of grouping, a controller may be assigned to coordinate and control group activities. This controller typically has the responsibility of decomposing tasks into job assignments for each subsystem, assigning resources to each subsystem, and developing a schedule of subtasks for each subsystem to accomplish its job. In many cases, especially at higher levels in the enterprise, coordination and control functions are performed by organizational units consisting of one or more humans that are schooled in the knowledge and trained in the skills required to perform the required control activity. Increasingly, computers can be programmed to perform these functions. Typically, computer controllers have operator interfaces whereby human operators can access the system to gather information, make decisions, and take action. Operators usually can select modes (such as manual, single-step, or automatic) and often can override automatic operations (for example by changing feed-rate, or commanding halt, or resume.)

Every corporation has some sort of organizational chart that describes the functional responsibilities of each management unit and defines the flow of command and control. Of course, no corporate management chart ever shows all the communication pathways by which information flows throughout the enterprise. In any organization, much information flows horizontally between peer agents, within management units and between units, through both formal and informal channels. Furthermore, the flow of command and control is not always strictly hierarchical. For example, a product design developed in one division of a manufacturing organization may be used to develop process plans in an entirely different part of the corporation, or even in a different company. Process plans developed in one organizational unit may be used to develop routing plans for material flow in a different unit. NC machine code developed in one corporate entity may be used by machinists in an entirely different facility for making parts. A programmer writing NC code may not even belong to the same company as the user who runs the code.

The ISAM model is designed to accommodate all of these possibilities. The fundamental property of the ISAM model is that it consists of modular nodes that correspond to organizational units. Each node contains one or more agents that have clearly defined functional responsibilities, knowledge, skills, and abilities. Each node also has clearly defined interfaces and communication channels that can be established with other nodes in the system. Thus, although most examples of the ISAM model are presented here in terms of a strictly hierarchical organization, ISAM can be adapted to fit any modular organization, so long as there are clearly defined roles and responsibilities for each of the modules and well defined interfaces between modules.

ISAM vs. Current Practice

ISAM can be used to describe current industry practice, but it is designed to anticipate and address the future needs of U.S. industry. For the most part, current industry practice assumes that manufacturing consists of highly predictable processes that can safely proceed without the benefit of, or need for, on-line measurement and real-time feedback control. Most on-line adjustments to manufacturing processes are made by

human operators who often use intuition and experience to tune parameters. In control parlance, most production processes operate "open-loop," with little or no consideration given to the need for real-time planning or replanning, automatic error recovery, on-line optimization, or adaptability to changing conditions. On-line schedule changes and process modifications are handled mostly by manual ad-hoc methods.

Current data exchange standards such as the Initial Graphics Exchange Specification (IGES) [1] and the Standard for Exchange of Product model data (STEP) [2, 3] are limited to static data. Most communication protocol standards such as the Manufacturing Message Specification (MMS) [4, 5, 6, 7] and TCP/IP do not address the semantics or logic of the processes being controlled. A few steps toward standards for interfaces that convey meaning such as the Common Object Request Broker (CORBA) [8] and Component Object Model (COM) [9] have recently appeared, but these do not support control systems that require hard real-time response. Open architecture interface standards such as Open Modular Architecture Controls (OMAC) [10] that are designed to both specify functionality and operate in real-time are still under development. The Metrology Automation Association [11] has supported early development efforts in interface standardization for Coordinate Measuring Machines (CMMs). A recent effort, IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems, focuses on how to describe such systems, and what should be included in the descriptions [12].

ISAM is intended to provide a theoretical foundation for the next generation of physical and informational measurements and standards. ISAM is designed to address future manufacturing practices that will depend heavily on in-process measurements and highly sophisticated computer-based control techniques. The assumption is that the future will be shaped by the demands of agile manufacturing, including rapid response to changing customer requirements, concurrent design and engineering, low-cost small-volume production, out-sourcing of supply, distributed manufacturing, just in-time delivery, real-time planning and scheduling, increased demands for precision and quality, reduced tolerance for error, in-process measurement, and feedback control. These demands generate requirements for adaptability and on-line decision making that cannot be met with current practice.

ISAM incorporates intelligent control concepts that in many cases are being developed outside of the field of industrial engineering. Intelligent control includes concepts from artificial intelligence, operations research, game theory, pattern recognition, neural nets, fuzzy logic, and control theory. It borrows heavily from cognitive psychology, semiotics, neuroscience, and computer science. Intelligent control closes the loop between sensing and acting through perception, world modeling, planning, and control. Intelligent control addresses problems of large complex systems with many sensors and actuators that are designed to pursue and achieve sophisticated goals in uncertain, competitive, and sometimes hostile environments. Intelligent control deals with systems designed to analyze the past, perceive the present, and plan for the future. Intelligent control enables systems to assess the cost, risk, and benefit of past events and future plans, and to make intelligent real-time dynamic choices between alternative courses of action in the face of

uncertainty. The ISAM conceptual framework brings intelligent control concepts to bear on the domain of manufacturing so as to enable the full range of agile manufacturing concepts.

II. ISAM as a Reference Model Architecture

At a middle degree of abstraction, ISAM is intended to provide a reference model architecture for the design of intelligent systems and software and for the development of future standards and performance measures. The goal of the ISAM reference model architecture is to provide a sound theoretical foundation for:

1. metrics, measures, and procedures that can quantitatively measure and evaluate the performance of intelligent control systems.

2. interface standards that can support dynamic real-time interactions between production goals and sensed conditions in the industrial environment.

3. open architecture standards that can enable manufacturing system software from a variety of vendors to work together without extensive effort required for software integration.

To illustrate the types of issues that are addressed by the ISAM reference model, an example of a seven level ISAM hierarchy for a machine shop is given below. *The particular functional assignments and numerical values given in the example are for illustration purposes only.* A methodology for applying the ISAM architecture to specific manufacturing applications will be addressed in a later section.

Level 7 -- Shop (≈5 h planning horizon)

The shop level plans activities and allocates resources for one or more manufacturing cells for a period of five to eight hours. At the shop level, orders for products are sorted into batches and a production schedule is generated for the cells to process the batches. At the shop level, the world model maintains a knowledge database containing names, contents, and attributes of batches and the inventory of tools and materials required to manufacture them. Maps may describe the location of, and routing between, manufacturing cells within the factory. Sensory processing algorithms compute information about the flow of parts, the level of inventory, and the operational status of all the cells in the shop. Value judgment computes the cost and benefit of various batching and routing options and evaluates statistical quality control parameters. An operator interface allows human operators to visualize the status of orders and inventory, the flow of work, and the overall situation within the entire shop facility. Operators can intervene to change priorities and redirect the flow of materials and tools. Executors monitor how well plans are being followed, and modify parameters as necessary to keep production on schedule. Output commands from shop level nodes consist of work flow assignments for specific cells.

Level 6—Cell (≈30 minute planning horizon)

The cell level plans activities and allocates resources for one or more workstations for a period of about 30 minutes into the future. Batches of parts and tools are scheduled into particular workstations. 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 within the cell. Sensory processing determines the location and status of trays of parts and tools. Value judgment evaluates routing options for moving batches of parts and tools. An operator interface allows human operators to visualize the status of batches and the flow of work through and within the cell. Operators can intervene to change priorities and reorder the plan of operations. Executors monitor how well plans are being followed, and modify parameters as necessary to keep work flow on schedule. The output commands from the cell level nodes consists of tasks assigned to specific workstation controllers to perform specific machining, inspection, or material handling operations on specific batches or trays of parts.

Level 5—Workstation (≈3 minute planning horizon)

The workstation level schedules tasks and controls activities within each workstation. A workstation may consist of a group of machines, such as one or more closely coupled machine tools, robots, inspection machines, material 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 machines, robots, and part trays within the workstation. Sensory processing computes the position and orientation of parts and tools in trays and buffers. Value judgment evaluates plans for sequencing machining and parts handling operations within the workstation. An operator interface allows human operators to visualize the status of parts and tools within the workstation, or to intervene to change priorities and reorder the sequence of operations within the workstation. Executors keep track of how well plans are being followed, and modify parameters as necessary to keep on plan. Output commands are issued to controllers for particular machine tools, robots, and tray buffers to perform specific operations on specific parts, tools, and fixtures.

Level 4—Equipment task (≈20 s planning horizon)

The equipment level schedules tasks and controls the activities of each machine within a workstation. (Tasks that take much longer than 20 seconds may be broken into several ≈20 s segments at the workstation level.) Level 4 decomposes each equipment task into elemental moves for the subsystems. Plans are developed that sequence elemental movements of tools and grippers, tool changers, and pallet shuttle systems. Commands are formulated to move tools and grippers so as to approach, grasp, move, fixture, 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 plan drawings that illustrate part shape and the relative positions of features on parts, or parts in assemblies. Sensory processing computes estimates of part shape and dimensions, and estimates differences between desired and measured properties. Value judgment

evaluates part quality and supports planning for part handling and fixturing sequences. An operator interface allows human operators to visualize the status of operations of the machine, or to intervene to change priorities or modify the sequence of operations. Executors monitor how well plans are being followed, and modify parameters as necessary to keep on plan. Output commands consist of elemental movement commands (such as <GoAlongPath>, <MoveToPoint>, <DrillHole>, <MillFace>, <MeasureSurface>) that are issued to machine subsystem controllers for controlling elementary movements or behaviors for machining, manipulating, and inspecting part features. Output motion commands are referenced to a coordinate frame defined in the object or part being operated upon.

Level 3—Elemental move (≈2 s planning horizon)

The Elemental move (E-move) level decomposes elemental movement commands into a series of trajectory segments defined in machine coordinates. (Complex movements that require significantly more than one second are broken up at the task level into several E-moves.) Plans are developed and commands are issued defining safe path way points or reference trajectories 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. Maps consist of drawings that illustrate feature shape and dimension and the relative positions of feature boundaries. Sensory processing computes estimated attributes of part features such as position, shape, dimensions, and surface properties. Value judgment supports planning of machine motions and evaluates feature quality. An operator interface allows a human operator to visualize the state of the machine, or to intervene to change mode or interrupt the sequence of operations. Executors monitor how well plans are being followed, and modify parameters as necessary to keep on plan. Output consists of commands to motion sequencers to move along trajectory segments between waypoints defined in machine coordinates.

Level 2—Primitive (200 ms planning horizon)

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. 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 tool or end-effector trajectories. Sensory processing applied to machine axis data computes estimated motions of tools and grippers. Sensory processing applied to touch probes and cameras may compute part feature attributes such as position, shape, size, and orientation. Value judgment supports trajectory optimization. An operator interface allows a human operator to visualize the state of the machine, the position of the tool, or to intervene to change mode or override the feed rate. Executors monitor how well plans are being followed, and modify parameters as necessary to keep part dimensions within tolerance. Output consists of commands to coordinated axis controllers to move tools or

grippers at desired velocities and accelerations in tool tip coordinates, or to exert desired forces, or maintain desired stiffness parameters.

Level 1—Servo level (20 ms planning horizon)

The servo level transforms commands from tool tip to joint actuator coordinates. Planners interpolate between primitive trajectory points for each actuator. The world model symbolic database contains values of state variables such as joint positions, velocities, and forces, proximity sensor readings, position of discrete switches, state of touch probes, as well as image attributes associated with camera pixels. Maps consist of camera images and displays of sensor readings. Sensory processing scales and filters data from sensors that measure actuator positions, velocities, forces, torques, and touch. This information is provided as estimated state feedback to actuator controllers. Sensory processing applied to data from touch probes may measure the position of points on part features. Sensory processing applied to camera data may compute attributes of pixels in an image. An operator interface allows a human operator to visualize the position and velocity of individual axes or to intervene to change mode or jog individual axes. Executors servo individual actuators and motors to follow interpolated trajectories. Position, velocity, or force servoing may be implemented, and in various combinations. Output consists of commands to power amplifiers that specify desired actuator torque or power. Outputs are produced every 2 ms (or whatever rate is dictated by the machine dynamics and servo performance requirements). The servo level also commands switch closures that control discrete actuators such as relays and solenoids.

At the Servo and Primitive levels, the output command rate is typically clock driven on a regular cycle. At the E-Move level and above, the output command rate becomes irregular because it is event driven.

The above example shows how the complexity inherent in a large manufacturing enterprise can be managed through hierarchical layering. Hierarchical decomposition of tasks is a well developed method for organizing complex systems. It has been used in many different types of organizations throughout history for effectiveness and efficiency of command and control. In a hierarchical organization, higher level nodes have broader scope and longer time horizons, with less concern for detail. Lower levels have narrower scope and shorter time horizons, with more focus on detail. Similarly, ISAM nodes at the upper levels in the hierarchy are responsible for long range plans consisting of major milestones, while at lower levels, ISAM nodes successively refine the long range plans into short term tasks with detailed activity goals. At lower levels, information from sensors is computed over local neighborhoods and short time intervals, while at higher levels, sensory information is integrated over large spatial regions and long time. At low levels, knowledge is short term and fine grained, while at the higher levels knowledge is broad in scope and generalized. At every level, feedback loops are closed to provide reactive behavior. Lower levels have high-bandwidth loops with fast-response reactions, whereas higher levels integrate feedback over longer time intervals and react more deliberately to slower trends.

The ISAM Node

ISAM defines a computational node as shown in Figure 1 consisting of five basic processes: sensory processing (SP), world modeling (WM), behavior generation (BG), value judgment (VJ), and a knowledge database process (KD). All these processes may have input and output connections to an Operator Interface. The arrows between processes indicate the flow of information within the ISAM node among SP, WM, BG, VJ processes, the KD, and the operator interface. The pathway from sensory processing through world modeling to behavior generation closes a reactive feedback control loop between observed input and commanded actions. The pathway from behavior generation through world modeling and value judgment back to behavior generation enables deliberative planning and reasoning about future actions. The pathway from sensory processing through value judgment to world modeling enables situation evaluation and learning. The looping interconnection between sensory processing and world modeling enables recursive estimation and knowledge acquisition. Thus, ISAM integrates real-time planning and execution of behavior with dynamic world modeling, knowledge representation, and sensory perception within each node.

Input from the Operator Interface enables a human supervisor to provide commands, to override or modify system behavior, to perform various types of teleoperation, to switch control modes (e.g., automatic, teleoperation, single step, pause). Output to the Operator Interface enables a human to observe the values of state variables, images, maps, and entity attributes. The Operator Interface can also be used for maintenance, programming, and debugging.

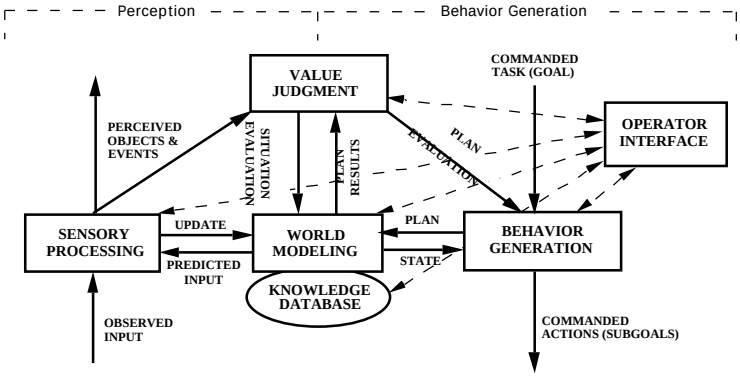


Figure 1. A node in the ISAM reference model architecture. The elemental processes of an intelligent system are behavior generation (planning and control), sensory processing (filtering, detection, recognition, and interpretation), world modeling (store and retrieve knowledge and predict future states), and value judgment (compute cost, benefit, importance, and uncertainty). These are supported by a knowledge database and a system architecture that interconnects the elemental processes and the knowledge database. This collection of processes and their interconnections make up a generic node in the ISAM reference model architecture. Each process in the node may have an operator interface.

Figure 2 shows a more detailed view of the ISAM node. In Figure 2, a BG unit accepts task command input from an Executor in a higher-level BG unit. Within the BG unit there is a task decomposition planner that decomposes each commanded task into set of a tentative plans for subordinate BG units. These tentative plans are submitted to a WM simulator/predictor which generates expected results. Value Judgment computes the cost and benefit of each tentative plan and its expected results, and reports this evaluation back to the task decomposition planner. The planner then selects the best set of tentative plans to give to Executors for execution. For each subordinate BG unit, there is an Executor that compares each step in its plan with feedback from the Knowledge Database. Each Executor then outputs subtask commands to a subordinate BG unit and monitors its behavior. The KD is kept up-to-date by processed sensory information. The SP, WM, and VJ processes that supply the KD with the information needed by the BG unit for planning and control are also shown in Figure 2.

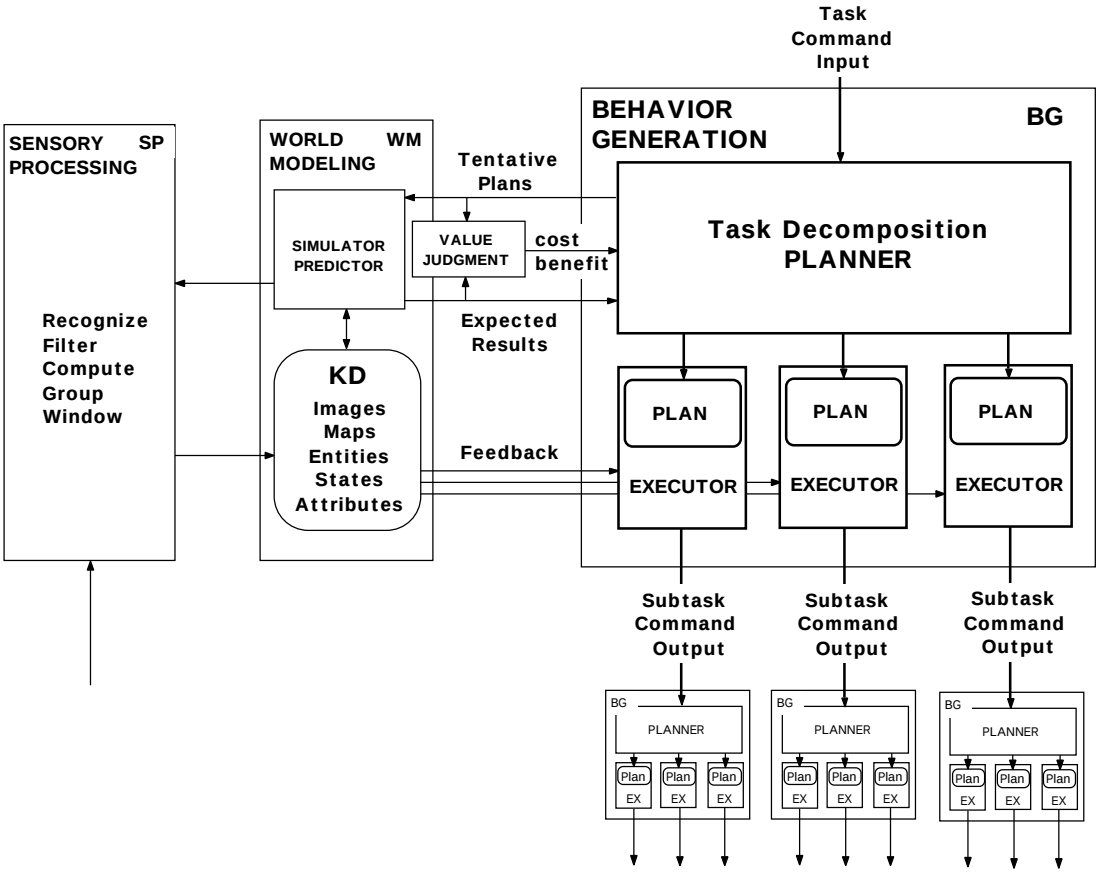


Figure 2. Interfaces between BG, WM, SP, and VJ. The Task Decomposition Planner within the Behavior Generation BG process selects or generates tentative plans. These are sent to the WM simulator/predictor to generate expected results. The Value Judgment process computes cost and benefits of tentative plans and expected results. The planner selects the tentative plan with the best cost/benefit

ratio and places it in plan buffers within the Executors. Each Executor cycles through its plan, compares feedback with planned results, and issues subtask command output to lower level BG modules.

The ISAM generic node illustrated in Figures 1 and 2 can be used to construct a hierarchical reference model architecture such as shown in Figure 3.

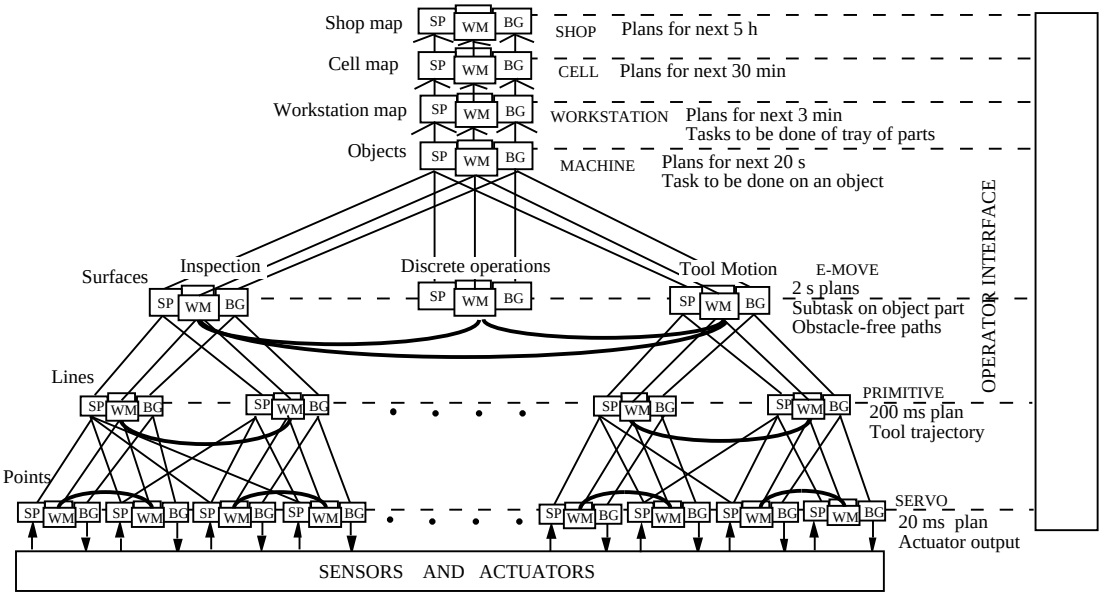


Figure 3. An ISAM reference model architecture for a machining center. Processing nodes are organized such that the BG processes form a command tree. On the right, are examples of the functional characteristics and planning horizon of the BG processes at each level. On the left, are examples of the type of entities recognized by the SP processes and stored by the WM in the KD knowledge database at each level. (KD processes are not shown in this figure.) Sensory feedback data flowing up the hierarchy typically form a graph, not a tree. VJ processes are hidden behind WM processes. An operator interface may provide input to, and output from BG, WM, SP, and VJ processes in every node.

Each node in the ISAM architecture shown in Figure 3 corresponds to an organizational unit in a manufacturing enterprise. At the top of Figure 3 is the Shop Control node responsible for generating plans for one or more Cell Control nodes out to a planning horizon of about 5 h. At the Cell level, Cell Control nodes are responsible for generating plans for one or more Workstation Control nodes out to a planning horizon of about 30 minutes. At the Workstation level, Workstation Control nodes are responsible for generating plans for one or more Machine Control nodes out to a planning horizon of about 3 minutes. At the Machine level, each Machine Control node is responsible for generating plans for each of its E-move nodes out to a planning horizon of about 20 s. The Machine node in Figure 3 supervises three E-move nodes. One is for an Inspection

subsystem that computes trajectories for touch probes and pan and tilt motors for cameras. A second is for a Tool Motion subsystem that computes motion trajectories for several axes of continuous tool path and spindle control. A third is for a Discrete operations subsystem that controls a conveyor, buffer, pallet load and unload, tool change, and coolant on/off. Coordination between E-move nodes is accomplished through peer-to-peer communication between agents in the Machine node and between world model processes at the E-move level. The horizontal curved lines between WM processes represent the sharing of state information between nodes within subsystems. Each E-move Control node is responsible for generating plans for its Primitive nodes out to a planning horizon of about 2 s. The Tool Motion E-move node has one or more Primitive Control nodes that compute dynamic tool trajectories out to a planning horizon of about 200 ms. The Inspection subsystem E-move node may have one or more Primitive Control nodes that compute camera pointing trajectories, or touch probe trajectories out to a planning horizon of 200 ms. The Discrete Operations E-move node typically has no Primitive Control node but connects directly to one or more Servo nodes. This is because discrete operations typically consist of simple switch closures. Finally, at the Servo level, tool motions are decomposed into signals for each actuator. For each actuator there is an Executor that sends it commands. Each sensor sends signals to a sensory processing process that provides updates to the world model knowledge database.

At all levels, ISAM nodes close reactive loops. At the Servo level, ISAM nodes route feedback from observed sensory input through world modeling to behavior generation in a tight feedback control loop. The Servo level knowledge database contains estimated and predicted state variables. Servo level sensory processing and world modeling interact to provide Kalman filtering. The Servo feedback loop for a machine tool typically operates at a rate of 10,000 Hz. For a robot, the Servo loop may operate at 1,000 Hz or even slower. At higher levels, reactive loops are slower because sensory information is processed, filtered, and clustered into higher level abstractions as it ascends the sensory processing hierarchy.

Each ISAM node acts as an intelligent controller. Depending on where the ISAM node resides in the architecture, it might serve as an intelligent controller for a set of actuators, a subsystem, a machine, a workstation, a manufacturing cell, a shop, factory, or upper management organizational unit. The functionality of any ISAM node (or a set of processes within a node) can be implemented as a computer controller, or as a human management unit, at any level in the manufacturing enterprise.

Within each ISAM node, World Modeling, Sensory Processing, and Value Judgment processes provide Behavior Generation processes with the information needed for decision making and control. Within each node, Behavior Generation processes decompose tasks into subtasks for subordinate Behavior Generation processes. At each level World Model knowledge is shared between Knowledge Databases at the same level, and relational pointers are established between Knowledge Data structures at both higher and lower levels.

At each level, Sensory Processing processes accept sensory observations from Sensory Processing processes at lower levels, and output processed and clustered sensory observations to Sensory Processing processes in higher level nodes. At all levels, SP processes compare and correlate predictions from the WM with observations from lower level SP processes. Differences are used to update the estimated state of the World stored in the world model Knowledge Database. Correlation is used to recognize correspondence between what is stored in the internal Knowledge Database and what is observed in the external real World.

At each level, knowledge is represented in a form, and with a spatial and temporal resolution, that meets the processing requirements of the nodes at that level. At each level, state variables, entities, events, and maps are maintained to the resolution in space and time that is appropriate to that level. At each successively lower level in the hierarchy, detail geometrically increases while range geometrically decreases. Temporal resolution increases while the span of interest decreases. This produces a computational load within each node that remains relative constant throughout the hierarchy. As a result, behavior generating functions make plans of roughly the same length in time at each level. For many this will seem strange, because in current practice machine tool paths may consist of thousands of steps, or only a single step. ISAM handles this by making planners at each level look ahead to relatively constant time horizons. Thus, a tool path that is 1000 commands long may be partitioned into 100 paths of roughly 10 steps, each of which take about the same period of time.

Sensory perception functions compute entities that contain roughly the same number of sub entities. At higher levels, plans, perceived entities, and world modeling simulations are more complex, but there are less of them and there is more time available between replanning intervals for processes to run. Thus, hierarchical layering keeps the amount of computing resources needed in each node within reasonable limits and relatively constant.

At each level, there is a characteristic loop bandwidth, a characteristic planning horizon, a characteristic type of task decomposition, a characteristic range of temporal integration of sensory data, and a characteristic window of spatial integration. At each level, information is extracted from the sensory data stream in order to keep the world model knowledge database accurate and up to date.

At each level, sensory data is processed, entities are recognized, world model representations are maintained, and tasks are deliberatively decomposed into parallel and sequential subtasks, to be performed by cooperating sets of agents within the BG processes. At each level, feedback from sensors reactively closes a control loop allowing each agent to respond and react to unexpected events. At each level, tasks are decomposed into subtasks and subgoals, and agent behavior is planned and controlled. The result is a system that combines and distributes deliberative and reactive control information throughout the entire hierarchical architecture, with both planned and reactive capabilities tightly integrated at all levels of space and time resolution.

At each level, global goals are refined and focused onto more narrow and higher resolution sub goals. At each level, attention is focused into a more narrow and higher resolution view of the world. The effect of each hierarchical level is thus to geometrically refine the detail of the task and the view of the world, while only linearly increasing the computational power required of the overall system.

At the bottom of the hierarchy and external to the control system, are actuators that act on the world environment, and sensors that transform events in the world into information signals for the control system. The external world environment contains a variety of real objects, such as materials, tools, machines, and fixtures as well as other intelligent agents, and forces of nature, all of which have states and may cause events and situations to occur.

Some definitions

To provide a solid foundation for interface standards and performance measures, the language used to define the reference model must be precise, so that there is no ambiguity and the reference model precisely reflects physical design and software architecture within the intelligent system. Therefore, we provide the following set of definitions for terminology used in describing ISAM.

Df: framework
a description of the functional elements, the representation of knowledge, and the flow of information within the system

Df: architecture
the assignment of functions to subsystems and the specification of the interfaces between subsystems

Df: reference model architecture
an architecture in which the entire collection of entities, relationships, and information units involved in interactions between and within subsystems are defined and modeled

To be considered as a reference model for the study and design of intelligent manufacturing systems, an architecture should have the following properties:

1. It should define the functional elements, subsystems, interfaces, entities, relationships, and information units involved in intelligent manufacturing systems.

2. It should support the selection of goals, the generation of plans, the decomposition of tasks, the scheduling of sub tasks, and provide for feedback to be incorporated into control processes so that both deliberative and reactive behaviors can be combined into a single integrated system.

3. It should support the processing of signals from sensors into knowledge of situations and relationships, and the storage of knowledge in representational forms that can support reasoning, decision making, and intelligent control.

4. It should provide both static (long term) and dynamic (short term) means for representing the richness and abundance of knowledge necessary to describe the manufacturing environment.

5. It should support the transformation of information from sensor signals to symbolic representations of objects, events, and situations; and from iconic (pictorial) to descriptive (symbolic) forms, and vice versa.

6. It should support the acquisition (or learning) of new information, and the integration and consolidation of newly acquired data into long term memory.

7. It should provide for the representation of values, the computation of costs and benefits, assessment of uncertainty and risk, the evaluation of plans and behavioral results, and the optimization of performance.

Df: intelligent system
a system with the ability to act appropriately in an uncertain environment

Df: appropriate action
that which maximizes the probability of success

Df: success
the achievement or maintenance of behavioral goals

Df: behavioral goal
a desired state of the environment that a behavior is designed to achieve or maintain

Df: functional elements
the fundamental computational processes from which the system is composed

Axiom: The functional elements of an intelligent system are sensory processing, world modeling, value judgment, and behavior generation.

Df: process
a series of actions or operations that accept input and produce output

Df: **sensory processing**
sensory processing is a set of processes by which sensory data interacts with prior knowledge to detect or recognize useful information about the world

Sensory processing accepts signals from sensors that measure properties of the external world or conditions internal to the system itself. Sensory processing scales, windows, and filters data, computes observed features and attributes, and compares them with predictions from internal models. Correlation between sensed observations and internally generated expectations are used to detect events and recognize entities and

situations. Variance between sensed observations and internally generated predictions are used to update internal models. Sensory processing also computes attributes of entities and events, and it clusters, or groups, recognized entities and detected events into higher-order entities and events.

In general, sensors do not directly measure the state of the world. Sensors typically only measure phenomena that depend on the state of the world. Signals generated by sensors may be affected by control actions that cause the sensors to move through the world. Sensor output signals are also corrupted by noise. The set of equations that describe how sensory output depends on the state of the world, the control action, and sensor noise is called a measurement model.

A measurement model is typically of the form

$$\mathbf{y = H(x, u, \eta)}$$

where
y = signals from sensors
x = state of the world
u = control action
η = sensor noise
H = a function that relates sensor output to world state, control action , and noise

A linearized form of the measurement model is typically of the form

$$\mathbf{y = Cx + Du + \eta}$$

where
C is a matrix that defines how sensor signals depend on the world state
D is a matrix that defines how sensor signals depend on the control action

Df: **world modeling**
world modeling is a process that constructs and maintains a world model that can be used for feedback control as well as for the generation of predictions of sensory signals and simulation of plans

World modeling performs four principal functions:

1. It generates and maintains a best estimate of the state of the world that can be used for controlling current actions and planning future behavior. This best estimate resides in a knowledge database describing the state and attributes of objects, events, classes, agents, situations, and relationships. This knowledge database has both iconic and symbolic structures and both short-term and long-term components.

2. It predicts (possibly with several hypotheses) sensory observations based on the estimated state of the world. Predicted signals can be used by sensory processing to configure filters, masks, windows, and schema for correlation, model matching, recursive estimation, and focusing attention.

3. It acts as a database server in response to queries for information stored in the knowledge database.

4. It simulates results of possible future plans based on the estimated state of the world and planned actions. Simulated results are evaluated by the value judgment system in order to select the "best" plan for execution.

Df: **world model**
an internal representation of the world

The world model is the intelligent system's best estimate of the world. The world model may include models of portions of the environment, as well as models of objects and agents. It also includes a system model that represents the internal state of the intelligent system itself. The world model is stored in a dynamic distributed knowledge database that is maintained by world modeling processes. Knowledge stored in the world model is distributed among computational nodes in the ISAM reference architecture. The world model in each node contains knowledge of the world with range and resolution that is appropriate for control functions in the behavior generation process in that node. The ISAM concept of a world model is closely related to the control theory of a system model.

Df: system model
a set of differential equations (for a continuous system) or difference equations (for a discrete system) that predict how a system will respond to a given input
A system model is typically of the form

$$\mathbf{dx/dt} = \mathbf{F}(\mathbf{x}, \mathbf{u}, \zeta)$$

where
x = state of the system
dx/dt = rate of change in the system state
u = control action
F = function that defines how the system state changes over time in response to control actions
ζ = error in the system model

A linearized form of the above system model is of the form

$$\mathbf{dx/dt} = \mathbf{A}\,\mathbf{x} + \mathbf{B}\,\mathbf{u} + \zeta$$

where

A is a matrix that defines how the system state evolves over time without control action
B is a matrix that defines how the control action affects the system state

Learning may enable the world model to acquire a system model. This type of learning is often called "system identification." Learning of world model parameters may be implemented by neural nets, adaptive filtering, or system identification techniques.

Df: knowledge database
the data structures and the static and dynamic information that collectively form the world model

The knowledge database is a store of information about the world in the form of structural and dynamic models, state variables, attributes and values, entities and events, rules and equations, task knowledge, images, and maps.

The knowledge database has three parts:

a) An internal representation of immediate experience consisting of sensor signals and current values of observed, estimated, and predicted images, entities, events, attributes, and relationships.

b) A short-term memory containing iconic and symbolic representations of entities, events, and relationships that are the subject of current attention. The list of items that are the subject of current attention is defined top-down by task commands, requirements, and expectations. The attention list is defined bottom-up by items that are surprising or dangerous.

c) A long-term memory containing symbolic representations of all the generic and specific objects, events, and rules that are known to the intelligent system. In some systems, long term memory may also contain iconic representations.

Df: value judgment
value judgment is a process that:

a) computes cost, risk, and benefit of actions and plans,
b) estimates the importance and value of objects, events, and situations,
c) assesses the reliability of information,
d) calculates the rewarding or punishing effects of perceived states and events.

In ISAM, success in achieving goals produces rewarding effects. Failure produces punishing effects. Knowing the expected cost, risk, and benefit of plans and actions is crucial to making good behavioral choices. Value judgment evaluates perceived and planned actions, events, objects, relationships, and situations, thereby enabling behavior generation to select goals and set priorities. Tentative plans evaluated as more beneficial, less risky, or less costly will be selected for execution over those evaluated as less beneficial, more risky, or more costly. Objects evaluated as attractive or valuable will be

pursued or defended. Objects evaluated as repulsive or feared will be avoided or attacked.

Value judgment computes what is important. Entities and events evaluated as important become the focus of attention. This enables cameras and other sensors to be directed toward objects and places in the world judged to be important. Evaluation of what is important is crucial for deciding what entities, events, and situations to store in memory. What is evaluated as important can be remembered by transfer from short-term to long-term memory. What is unimportant can be safely ignored and forgotten.

Value judgment assesses the reliability of information and judges whether current sensory experience or stored information in the world model is more believable and a more reliable guide for behavior. Information judged reliable will be trusted, and confidence assigned to predictions based on that information. Confidence and believability factors can be attached to various interpretations of entities and events.

Value judgment also computes what is rewarding and punishing. This can be used directly for feedback control of action as well as for learning. An intelligent system capable of learning will learn to repeat actions leading to events and situations evaluated as good or rewarding, and will learn to avoid those actions resulting in punishment.

Df: **behavior generation**
behavior generation is the planning and control of actions designed to achieve behavioral goals

Behavior generation accepts task commands with goals and priorities, formulates and/or selects plans, and controls action. Behavior generation develops or selects plans by using a priori task knowledge and value judgment functions combined with real-time information provided by world modeling to find the best assignment of tools and resources to agents, and to find the best schedule of actions (i.e., the most efficient plan to get from an anticipated starting state to a goal state). Behavior generation controls action by both feedforward actions and by feedback error compensation

Df: **command**
a name, a commanded action, and a command goal. Both commanded action and command goal may include parameters.

The command name is an identifier. The commanded action may include parameters that specify how, where, when, how much, how fast, and on what. The command goal is a desired state that may include parameters that specify tolerance in space and time on various state variables. Goal parameters may also specify the value of the goal.

Df: **control law**
a set of equations that computes control action given predicted state, desired state, and feedforward action

A control law is typically of the form

u = E(uff, xd, x*)

where
u = control action
uff = feedforward control action (from a plan)
xd = desired world state (from a plan)
x* = predicted world state (from the world model)
E = a function that defines how the output depends on goals, plans, and feedback

A linearized form of a control law is

u = uff + G(xd – x*)

where
G = a matrix that defines the feedback compensation applied to the difference between the desired and predicted state of the world

III. ISAM Engineering Guidelines

At the lowest level of abstraction, ISAM is intended to provide engineering guidelines for implementing specific instances of manufacturing systems such as machining and inspection systems. In this section we will describe in some detail how the elemental processes can be implemented using current software engineering tools, languages, and practices.

We will start with a detailed analysis of the internal structure of the elemental processes that make up the ISAM node described in the previous section. We will then suggest several methodologies and software tools and libraries that can be used to implement these nodes within the ISAM architecture.

Behavior Generation

Df: The Behavior Generation (BG) process
a functional process within an ISAM node responsible for decomposing tasks into jobs for a set of agents within the node, for generating coordinated plans for those agents, and for executing those plans so as to produce subtasks for subordinate nodes.

A BG process is the part of an ISAM node that plans, coordinates, and executes tasks. The set of intelligent agents in the BG process may correspond to a team of persons within a management unit, or a set of coordinated processes in an intelligent control system. For any ISAM node, there exists a set of skills for a collection of tasks that the BG process is capable of performing. The knowledge required to perform this collection of tasks is stored in a set of task frames. If the BG process is given a command to do a

task that it is capable of performing, it uses the knowledge in the corresponding task frame to accomplish the task. If it is given a command to do a task that it is incapable of performing, (i.e. for which there is no task knowledge) it responds to its supervisor with an error message = <can't do>.

A typical BG process contains four types of subprocesses:
1. A Job Assignor (JA)
2. A set of SChedulers (SC)
3. A Plan Selector (PS)
4. A set of EXecutors (EX)

Within the BG process, the JA, SC, PS, and EX subprocesses are arranged in the configuration shown in Figure 4. Together, JA and SC processes generate tentative plans that the PS submits to the WM and VJ for evaluation.

Df: Job Assignor (JA)
a subprocess of BG that decomposes input tasks into job assignments for subordinate BG units

The Job Assignor (JA) performs four functions:

1. JA accepts input task commands from an Executor in a higher level BG unit.
2. JA decomposes each commanded input task into a set of jobs which will be assigned to schedulers for subordinate BG units. This is a spatial decomposition.
3. JA transforms each job assignment into a coordinate frame of reference which is appropriate for the performing subordinate BG unit.
4. JA allocates resources to the subordinate BG units to enable them to accomplish their assigned jobs[1].

Df: job
an activity and goal assigned by the Job Assignor to a Scheduler within a BG process

A job is evoked by a command from a Job Assignor to a Scheduler subprocess. A job defines the work to be done by an agent. A job consists of a job activity and a job goal. The difference between a job and a task is that a task is an input to a JA subprocess and a job is an output from a JA subprocess. A task is an assignment to be performed by a BG unit at the j-th level, while a job is decomposed into a sequence of subtasks to be performed by a subordinate BG process at the (j-1)-th level.

Df: Scheduler (SC)
a subprocess within BG that accepts a job assignment and computes a schedule for its subordinate BG unit

[1] At lower levels, the assignment of resources to subordinate BG units may be fixed by the system design.

There is a SC subprocess for each subordinate BG unit. Each SC accepts its job assignment from the JA and computes a sequence of planned actions and resulting states, from a job starting state to a job goal state. This is a temporal decomposition. The SC subprocesses within the BG unit typically communicate with each other, and may negotiate with each other to resolve conflicts and coordinate their respective job plans. SC processes may define plan synchronization flags that EX processes can use to synchronize activities between subordinate BG units.

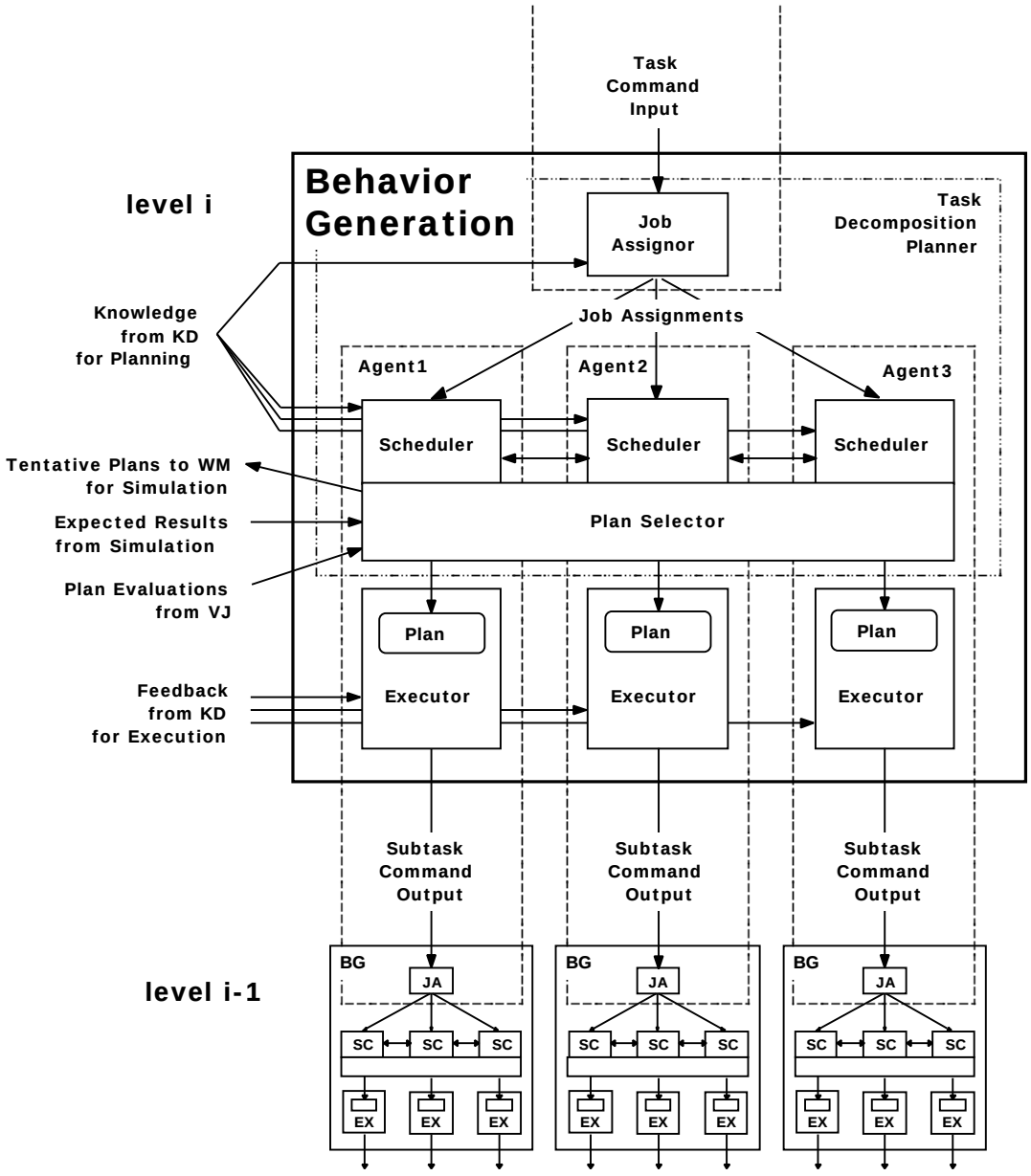


Figure 4. Internal structure of Behavior Generation (BG) units. The task decomposition planner contains a Job Assignor (JA), and a Scheduler (SC) for each of the subordinate BG units. The JA and SC processes generate tentative plans that are

submitted to WM for simulation, and to VJ for evaluation. The Plan Selector (PS) selects the best of the tentative plans to be sent to the appropriate Executor. The three agents are peers at level i and supervisors at level i-1. The JA subprocess at level i is part of a supervisor agent from level i+1.

Df: Plan Selector (PS)
a subprocess within BG that works with a WM plan simulator and VJ plan evaluator to select the best overall plan (i.e. job assignment, resource allocation, and coordinated schedule) for its subordinate BG units

The Plan Selector subprocess submits tentative plans generated by the JA and SC subprocesses to the World Model for simulation. The World Model simulator generates a prediction of what would likely result if the tentative plan were executed. The results of simulation are then sent to Value Judgment processes for evaluation. The Value Judgment evaluates the cost, benefit, and risk of the plan and its predicted result, and returns its evaluation to the Plan Selector. The Plan Selector then selects the best of the tentative plans to be placed in plan buffers for execution. This is a decision process. Many tentative plans may need to be evaluated before a "best" plan can be selected.

The JA and SC subprocesses within a BG unit comprise a management team. The JA subprocess acts as the supervisor of the BG unit. The SC subprocesses act as planners for the peer agents within the BG organizational unit. This management team can implement different management styles by different distribution of duties and responsibilities between the JA and SC subprocesses. For example, an autocratic, or micro-management style can be achieved if the JA subprocess assumes most or all of the responsibility for both job assignment and scheduling, leaving very little discretion to the SC subprocesses. On the other hand, a collegial management style can be achieved if the SC subprocesses negotiate with the JA for job assignments and assume complete responsibility for scheduling their respective jobs. In either extreme, or some combination of the two styles, the JA and SC subprocesses work together to create tentative plans that allocate resources and responsibilities to subordinate BG units to accomplish the commanded task.

It is the responsibility of the Plan Selector to assure that the plan buffer always contains an acceptable plan. This guarantees that execution is never starved for input. Whenever a new tentative plan is evaluated as better than the current plan in the buffer, the Plan Selector replaces the plan in the buffer with the better plan. This assures that the plan buffer always contains the best plan found to date.

The Plan Selector decouples planning from execution. Plan execution can respond immediately to sensory feedback, while planning proceeds asynchronously at its own pace. All that is required is that the planner (i.e. the combined activity of JA, SC, WM, VJ, and PS) generates new plans at least as fast as the executors carry them out.

For applications where the world environment is very dynamic, the planner can continuously be computing a new plan while the Executor is executing the current plan.

As soon as a newer and better plan is ready, the PS can replace the old plan in the plan buffer, and the Executor can continue without interruption.

Df: Executor (EX)
a subprocess within BG that executes its portion of the selected plan by generating subtasks for its subordinate BG process

The EX subprocess coordinates actions between itself and other EX subprocesses. It handles errors between planned results and the evolution of the world state reported by the world model. There is an EX subprocess for each subordinate BG unit. Each EX subprocess closes a feedback control loop within its own BG unit. This is a reactive process.

For continuous control systems, each EX subprocess:

1. computes feedforward commands based on planned actions
2. detects errors between the current planned subgoal (i.e., desired state) and the observed (or predicted) state of the world
3. computes feedback error compensation designed to null the errors
4. outputs commands consisting of feedforward and feedback compensation to its subordinate BG unit.
5. reacts to emergency conditions

For discrete event systems, each EX subprocess:

1. computes feedforward task commands based on planned actions
2. increments its controller from action to action along the plan as goals are achieved.
3. modifies output command parameters as necessary to correct errors between observed (or predicted) states and planned states.
4. reacts to emergency conditions

For both continuous and discrete systems, EX subprocesses produce strings of output subtask commands that become input task commands to JA subprocesses in BG units at the next lower level.

Organizational Units vs. Agents

In Figure 4, note the distinction between the BG organizational unit and the agents that belong to the units. The ISAM BG process is defined from the perspective of organizational units. Examples include military units such as division, battalion, company, platoon, squad, and vehicle. Agent architectures such as are popular in the current literature are typically organized from the perspective of agents. Examples include personnel with rank such as major, captain, lieutenant, sergeant, private [13, 14, 15]. The ISAM architecture can be viewed either as a hierarchy of agents or a hierarchy of organizational units. In both cases, it is important to maintain a clear distinction

between organizational units and the agents that belong to them. If this distinction is not carefully observed, the system design (and discussions between designers) can become very confusing.

For example, it should be noted that each organizational unit consists of agents with two different ranks:

1) a commander agent of higher rank, and
2) subordinate agents of lower rank.

Conversely, each intelligent agent belongs to two organizational units at two different levels:

1) a lower level unit in which the agent is a commander; and
2) a higher level unit in which the agent is a peer with other agents of similar rank that are subordinate to a commander of higher rank.

ISAM makes this distinction by defining tasks as assignments to operational units, and jobs as assignments to agents that belong to those units.

In Figure 4, a BG unit at level(i) consists of a JA subprocess that performs the role of unit supervisor, and one or more SC and EX subprocesses that fill the role of subordinate staff in the unit. Within the dotted lines, the SC, EX, and JA subprocesses are arranged as agents with SCs and EXs acting as peer subordinates in the BG unit at level(i) and the corresponding JAs acting as supervisors of BG units at level(i-1).

Df: agent
an entity that plans and executes jobs (possibly in coordination with other agents)

Each agent[2] contains three subprocesses as shown in Figure 4:

1. A Scheduler (SC) at level i
2. An Executor (EX) at level i
3. A Job Assignor (JA) at level i-1

Within each BG process, the JA subprocess of a supervisor agent works with the SC subprocess of peer agents to develop a plan. The EX subprocess of each agent works with the JA subprocess at the next lower level to command the subordinate BG unit. The EX thus acts as a server to the lower level BG unit. It interfaces between upper and lower level planners and provides synchronization by establishing start times of subtasks. Thus, each BG unit can plan and coordinate the actions of subordinate BG units. The set of agents within each BG unit can work as a team to accomplish tasks assigned to the BG unit of which they are a part. Coupling between Job Assignment and Scheduling

[2] The RCS convention defines agents as separate from the SP, WM, VJ, and KD processes that support them within a node. In most of the artificial intelligence literature, the definition of an agent includes the SP, WM, VJ, and KD functions within the agent itself. The ISAM convention is preferred here because RCS treats the SP, WM, and VJ functions, and the KD data structures separate from the BG functions.

functions produce teamwork between supervisor and subordinate agents in planning and execution of BG tasks.

Each agent within a BG process may compete or cooperate with other agents within the same BG process. Within a BG process, a SC subprocess may negotiate with the JA subprocess of the supervisor agent, and compete against peer agent SC subprocesses for job assignments and resource allocations. The SC subprocesses of an agent may also work together with other SC subprocesses to plan cooperative behavior. SC subprocesses may work with their supervisor agent to formulate tentative job assignments and schedules for plans. The Executor subprocess of each agent then executes its part of the selected plan by outputting task commands to the JA subprocess in a BG unit at the next lower level.

Whether ISAM is viewed as a hierarchy of organizational units, or a hierarchy of agents, it must be understood that this hierarchy is a nested multiresolutional structure in which communication flows both vertically between supervisors and subordinates and horizontally between peers. Mathematically ISAM is not a tree. It is a layered multiresolutional lattice that is richly interconnected by horizontal communications within each layer.

The structure shown in Figure 4 makes clear a number of different ways that dividing lines between levels in the ISAM hierarchy could be established.

1) The boundary between BG units at different levels establishes a dividing line between organizational units. Information flows across this boundary in the form of task commands.
2) The boundary between JA and SC subprocesses establishes a dividing line between agents. Information flows across this boundary in the form of job assignments.
3) The boundary between the PS and EX subprocesses establishes a dividing line between tasks and subtasks. Information flows across this boundary in the form of plans.

The overall result is that tasks are input to a BG unit at one rate, and subtasks are output to lower level BG units at about an order of magnitude higher rate. Similarly, jobs are assigned to agents in a BG unit at one rate, and to agents in BG units at the next lower level at an order of magnitude higher rate.

Tasks and Plans

Df: task
an activity to be performed by a set of agents in a BG process to achieve or maintain a task goal state

A task to achieve a goal, such as <to machine a part>, <to paint a house>, <to assemble an engine>, or <to shop for groceries>, is “done” when the task is accomplished. At that time, the system is “ready” to accept another task. A task to maintain a goal, such as

<to maintain a temperature>, <maintain velocity>, or <maintain altitude> is typically continued until a command is received to perform a different task.

Df: task goal
a desired state that a task is designed to achieve or maintain

A task implies action. In natural language, verbs are used to connote action. In the above definition of task, the infinitive form of the verb is used (e.g. task = <to machine a part>). The application of the verb <do> to the <task> transforms the task verb from its infinitive to its active form. It puts the task verb into the form of a command. For example, <do_<to paint the house>> becomes a command to <paint the house>, where <paint> is the action verb, and <the house> is the object upon which the action is done.

Df: do_task
a command to perform a task

For any task, there can be assigned an action verb that is a command to perform the task. The set of tasks that a BG process is capable of performing therefore defines a vocabulary of action verbs, or commands, that the BG process is capable of accepting.

A task such as <to paint a house> may be performed by a number of agents (painters) on many objects (walls, windows, doors, shutters, etc.) A task such as assemble an engine may be performed by a number of agents (assembly line workers) on a number of objects (pistons, bearings, gaskets, head, oil pan, etc.) At the input of the BG process, a task command such as <paint a house> or <assemble an engine> is encoded as single command to do a single activity to achieve a single goal. The JA process at input of the BG process performs the job assignment function of decomposing the task into a set of jobs for a set of agents (a painting crew, or an assembly line team). For each of the agents, a Scheduling process schedules a sequence of subtasks, which are executed by the EX process within each agent to accomplish the commanded task.

Df: task command
a command to a BG process to do a task, or to modify a previous task command

A task command can be represented as an imperative sentence in a message language that directs a BG process to do a task, and specifies the object(s), the task goal, and the task parameters. A task command has a specific interface data structure defined by a task command frame.

Df: task command frame
a data structure containing all the information necessary for commanding a BG process to do a task

A task command frame may include:
1. *task name* (from the vocabulary of tasks the receiving BG process can perform)
2. *task identifier* (unique for each commanded task)

3. *task goal* (a desired state to be achieved or maintained by the task)
4. *task goal time* (time at which the goal should be achieved, or until which it should be maintained)
5. *task object(s)* (on which the task is to be performed)
6. *task parameters* (such as speed, force, priority, constraints, tolerance on goal position, tolerance on goal time, tolerance on path deviations, coordination requirements, and level of aggressiveness.)
7. *next task name, goal, goal time, objects, and parameters*.

The task command presented to the BG process at level i is derived from the first planned subtasks in the current plan at level i+1. Item #7 in the task command frame enables the BG planning process to always plan to its planning horizon even when its planning horizon extends beyond the current task goal.

The interface between BG processes at level i+1 and level i consists of the task command from level i+1 to level i and the status response which returns from level i to level i+1 either directly, or through the world model.

Task Knowledge

Task knowledge is what enables a BG process to perform tasks. Task knowledge includes skills and abilities possessed by the BG unit and its agents. For example, task knowledge may include knowing how to machine a part, build an assembly, or inspect a feature.

Task knowledge may also include a list of equipment, materials, and information required to perform a task. It may include a set of conditions required to begin or continue a task, a set of constraints on the operations that must be performed, a set of priorities, and a mode of operation. Task knowledge may include error correction procedures, control laws, and rules that describe how to respond to failure conditions.

For any task to be successfully accomplished, knowledge of how to do the task must exist. Task knowledge must either be available a priori, or the system must have a procedure for discovering it. A priori task knowledge may be provided in the form of schema or algorithms designed by a programmer. It may be discovered by heuristic search over the space of possible behaviors. It may be acquired through learning, or generated by genetic algorithms. Task knowledge is invoked by task commands that provide parameters such as priority, mode, speed, force, or timing constraints. Task knowledge can be represented in a task frame.

Df: task frame
a data structure specifying all the knowledge necessary for accomplishing a task

A task frame is essentially a recipe consisting of a task name, a goal, a set of parameters, a list of materials, tools, and procedures, and a set of instructions of how to accomplish a task. A task frame may include:

1. *task name* (index into the library of tasks the system can perform) The task name is a pointer or an address in a database where the task frame can be found.
2. *task identifier* (unique id for each task command) The task identifier provides a method for keeping track of tasks in a queue.
3. *task goal* (a desired state to be achieved or maintained by the task) The task goal is the desired result of executing the task.
4. *task goal time* (time at which the task goal should be achieved, or until which the goal state should be maintained)
5. *task objects* (on which the task is to be performed) Examples of task objects include parts to be machined, features to be inspected, tools to be used, targets to be attacked, objects to be observed, sectors to be reconnoitered, vehicles to be driven, weapons or cameras to be pointed.
6. *task parameters* (that specify, or modulate, how the task should be performed) Examples of task parameters are speed, force, priority, constraints, tolerance on goal position, tolerance on goal time, tolerance on path, coordination requirements, and level of aggressiveness.
7. *agents* (that are responsible for executing the task) Agents are the subsystems and actuators that carry out the task.
8. *task requirements* (tools needed, resources required, conditions that must obtain, information needed) Tools may include instruments, sensors, and actuators. Resources may include fuel and materials. Conditions may include temperature, pressure, weather, visibility, soil conditions, daylight or darkness. Information needed may include the state and type of parts, tools, and equipment, or the state of a manufacturing process, or a description of an event or situation in the world.
9. *task constraints* (upon the performance of the task) Task constraints may include speed limits, force limits, position limits, timing requirements, visibility requirements, tolerance, geographical boundaries, or requirements for cooperation with others.
10. *task procedures* (plans for accomplishing the task, or procedures for generating plans) Plans may be prepared in advance and stored in a library, or they may be computed on-line in real-time. Task procedures may be simple strings of things to do, or may specify contingencies for what to do under various kinds of circumstances.
11. *control laws and error correction procedures* (defining what action should be taken for various combinations of commands and feedback conditions) These typically are developed during system design, but may be developed through learning from experience.

Some of the slots in the task frame are filled by information from the command frame. Others are a priori properties of the task itself and what is known about how to perform it. Still others are parameters that are supplied by the WM.

Df: task object
a thing in the world that is acted upon by an agent in order to accomplish a task. For any object, there can be assigned a word (noun) which is the name of the object upon which the task is performed.

Df: task parameters
modifiers that specify, identify, or prioritize a task, or modulate the manner in which a task is to be performed

Df: tool
a device that may be used to perform a task

Task Decomposition

Df: task decomposition
a process by which a task given to a BG process at one level (i) is decomposed into a set of sequences of subtasks to be given to a set of subordinate BG processes at the next lower level (i-1)

Task decomposition consists of five generic functions:

1. a *job assignment* function is performed by a JA subprocess whereby:
   a) a task is divided into jobs to be performed by agents,
   b) resources are allocated to agents
   c) the coordinate frame in which the jobs are described is specified
2. *scheduling* functions are performed by SC subprocesses whereby a job for each agent is decomposed into a sequence of subtasks (possibly coordinated with other sequences of subtasks for other agents). Working together, JA and SC functions generate tentative plans.
3. *plan simulation* functions are performed by a WM process whereby the results of tentative plans are predicted
4. *evaluation* functions are performed on the results of tentative plan simulations by a VJ process using cost/benefit analysis
5. a *plan selection* function is performed by a PS subprocess in selecting the "best" of the alternative plans for execution

The result of task decomposition is a plan consisting of a set of planned subgoals connected by planned actions. The objective of task decomposition is to maximize the likelihood of success, maximize the benefit, and minimize the cost and risk of achieving the commanded task goal.

Planned actions can be used by the EX subprocess to generate feedforward commands. Planned subgoals can be used by the EX to generate desired states. A string of desired states is sometimes called a reference trajectory. The desired states can be compared with observed states, and differences can be treated as error signals that are submitted to a control law to compute feedback compensation. Feedforward and feedback compensation can be combined to produce sequences of subtask commands to BG processes at the next lower level in the control hierarchy.

Thus, each task command input to a BG process generates a behavior described by a set of sequences of subtask commands output to a set of subordinate BG processes. The JA subprocess within a BG process accepts task commands with goals and priorities. The JA, SC, and PS subprocesses develop or select a plan for each commanded task by using a priori task knowledge, heuristics, and value judgment functions combined with real-time information and simulation capabilities provided by world modeling functions. The planning process generates the best assignment of tools and resources to agents, and finds the best schedule of actions (i.e., the most efficient plan to get from an anticipated starting state to a goal state). EX subprocesses control action by stepping through the plan. At each control cycle, the EX subprocesses can both issue feedforward commands and compare current state feedback with the desired state specified by the plan. The EX subprocesses merge feedforward actions with feedback error compensation to generate task commands to the next lower level BG processes.

Plans and Planning

Df: plan
a set of subtasks and subgoals that are designed to accomplish a task or job

A plan consists of one or more paths through state-space from an anticipated starting state to a desired goal state [16, 17, 18]. A plan also may include a list of required resources such as tools or materials, and a set of conditions, constraints, tolerances, and priorities that must be satisfied.

Df: job plan
a plan for a single agent to accomplish a job

In general, a job plan can be defined by a state-graph wherein the nodes correspond to desired subgoals, and edges correspond to conditions and actions that cause the agent to transition from one subgoal to the next. A job plan can also be represented as a state-table wherein each line corresponds to an IF predicate consisting of a state and set of conditions, and a THEN consequent consisting of an action and next state transition.

Df: task plan
a set of job plans for a set of agents in a BG process to accomplish a task

The distinction between a task plan and a job plan is illustrated in Figure 5. This shows the decomposition of a task into a task plan consisting of three job plans for three agents. Each job plan is represented as a state graph where subgoals are nodes and subtasks are edges that lead from one subgoal to the next.

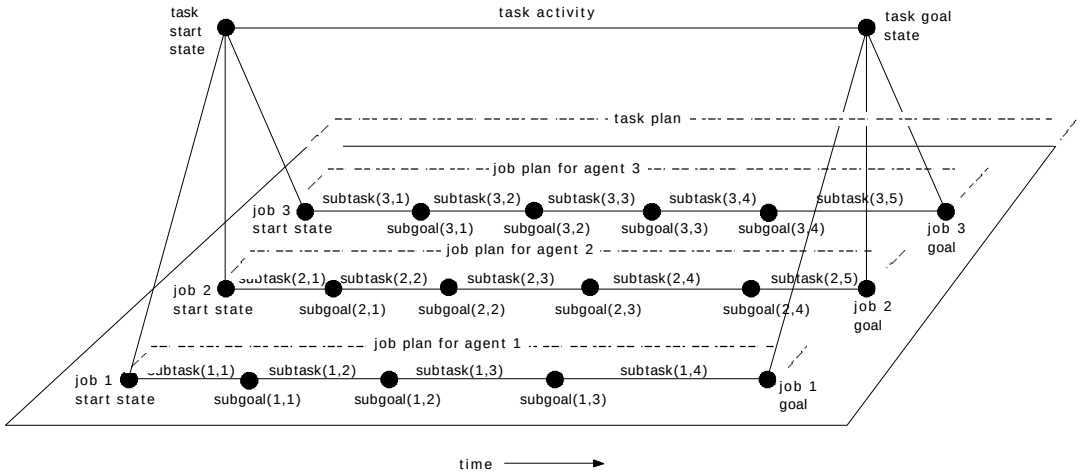


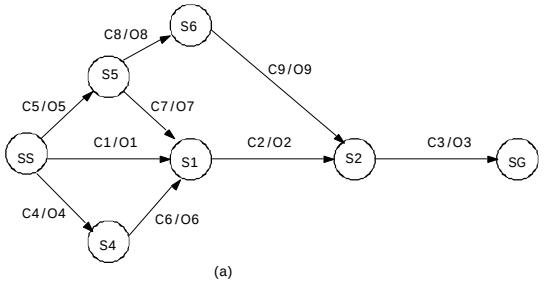
Figure 5. A task plan for a BG with three agents. In this figure, a plan for a single task consists of three parallel job plans for three agents. Each job plan consists of a string of subtasks and subgoals. The entire set of subtasks and subgoals is a task plan for accomplishing the task.

The task plan illustrated in Figure 5 consists of three job plans defined by linear lists of subtasks and subgoals. Coordination between multiple agents can be achieved by making state transitions in the job plan of one agent conditional upon states or transitions in job plans of other agents, or by making state transitions of coordinated agents dependent on states of the world reported in the world model.

There are many different ways to represent a plan. The task plan shown in Figure 5 is represented in the form of a Gantt chart, wherein activities and milestones are plotted against time for each job. In this case, activities correspond to subtasks that occur over a time interval and milestones correspond to subgoals that occur at points in time. In some cases, path plans may be abbreviated to include only strings of subgoals expressed as waypoints where the activity of <go to> between waypoints is understood. A string of subgoals that define a planned path is sometimes called a “reference trajectory.” In other cases, plans may be abbreviated to include only strings of activities that cause the system to follow a planned path. Plans for continuous motion may be represented by splines, or curves through state space defined by differential equations.

In general, any plan for discrete actions may be represented by a state-graph, Petri net, or Pert chart. Figure 6(a) illustrates a plan expressed as a state-graph where nodes represent desired states (or subgoals) and edges represent actions that cause state changes (or subtasks). Figure 6(b) illustrates a plan expressed as a state-table or set of IF/THEN statements. For any state-graph, a state-table can be constructed such that each edge in the state-graph corresponds to a line in the state-table, and each node in the state-graph corresponds to a state in the state-table. The state-table form has the advantage that the IF/THEN statements can be directly executed as case statements of a computer program.

This form suggests how knowledge expressed in the form of an expert system can be directly translated into plans.



(a)

IF		THEN	
State	Feedback	Next State	Output
SS	C1	S1	O1
SS	C5	S5	O5
SS	C4	S4	O4
S1	C2	S2	O2
S5	C7	S1	O7
S5	C8	S6	O8
S6	C9	S2	O9
S2	C3	SG	O3

(b)

Figure 6. Two forms of plan representations. (a) shows an example of a state-graph representation of a plan that leads from starting state SS to goal state SG. (b) shows the state-table that is the dual of the state-graph in (a).

A state-graph can represent conditional branching and thus can specify alternative courses of action depending on conditions detected by sensors. For example, a job plan might contain guarded moves or emergency actions that would be triggered by out-of-range conditions. A job plan may contain alternative actions such as to accept or reject a part depending on the value of a state variable or the setting of a switch.

In general, any state-graph can be implemented by a finite state automaton. Of course, every state graph has a dual of a state table, making it mathematically equivalent to a set of rules in an expert system. A discrete plan can also be represented procedurally as a computer program that controls an agent or mechanical device in generating behavior that leads from a starting state to a goal state. A series of commands or statements that produce actions that cause a system to transition from one subgoal to the next along the path to the goal state is sometimes called a procedure.

In all cases, a plan defines a trajectory through state space. A plan typically specifies what each agent is responsible for doing and a schedule of when each action should be done. The schedule may specify the timing of events and requirements for coordination between the actions of two or more agents or devices.

Df: schedule
the timing or sequencing specifications for a plan.

A schedule can be represented as a sequence of activities that are indexed by time or triggered by events. A schedule may specify that activities or goal events should occur within certain time constraints [19]. Or a schedule may simply specify the order in which activities should occur regardless of the time. For example, the sequence of program steps in a robot assembly task typically does not depend on clock time, but on events that occur as the assembly task proceeds. If a plan is represented as a state-graph or state-table, conditions for transition from one subtask activity to the next can be specified in terms of either time or events, or both [20].

Df: process plan
a plan, typically without a specified schedule, that defines the sequence of tasks required to manufacture a single part, or part type.

In manufacturing, process plans for specific parts most often are generated by humans that are experienced in process planning. Computer-aided process planning systems are available in the market to assist human process planners. Often, process plans are generated by minor modifications from previously generated plans for similar parts. NC programs are a type of process plans developed for individual machine tools. Process plans typically are generated long before production plans. Process plans set constraints for production planning.

Df: production plan
a plan, typically with a specified schedule, that defines the sequences of tasks required to operate a facility or a group of machines during the manufacture of a set of parts.

Production plans are often generated by computer programs that compute schedules for manufacturing operations. These are usually generated off-line in batch mode, shortly before production begins. They often become obsolete shortly after production begins.

Df: planning
a process of generating and/or selecting a plan.

In general, planning involves a search over the space of possible futures to generate or select a series of actions and resulting states (or a series of desired states and required actions) that lead from a starting state to a goal state.

Typically, planning consists of the following elements:
1. assigning jobs to agents
2. allocating resources to agents
3. transforming coordinates into agent coordinate frames
4. generating a tentative schedule of concurrent and possibly coordinated activities for each of the agents
5. simulating the likely results of this tentative schedule of activities

6. evaluating the cost/benefit value of these likely results
7. selecting the tentative schedule with the best evaluation as a plan to be executed.

A diagram of this generic planning process is shown in Figure 7.

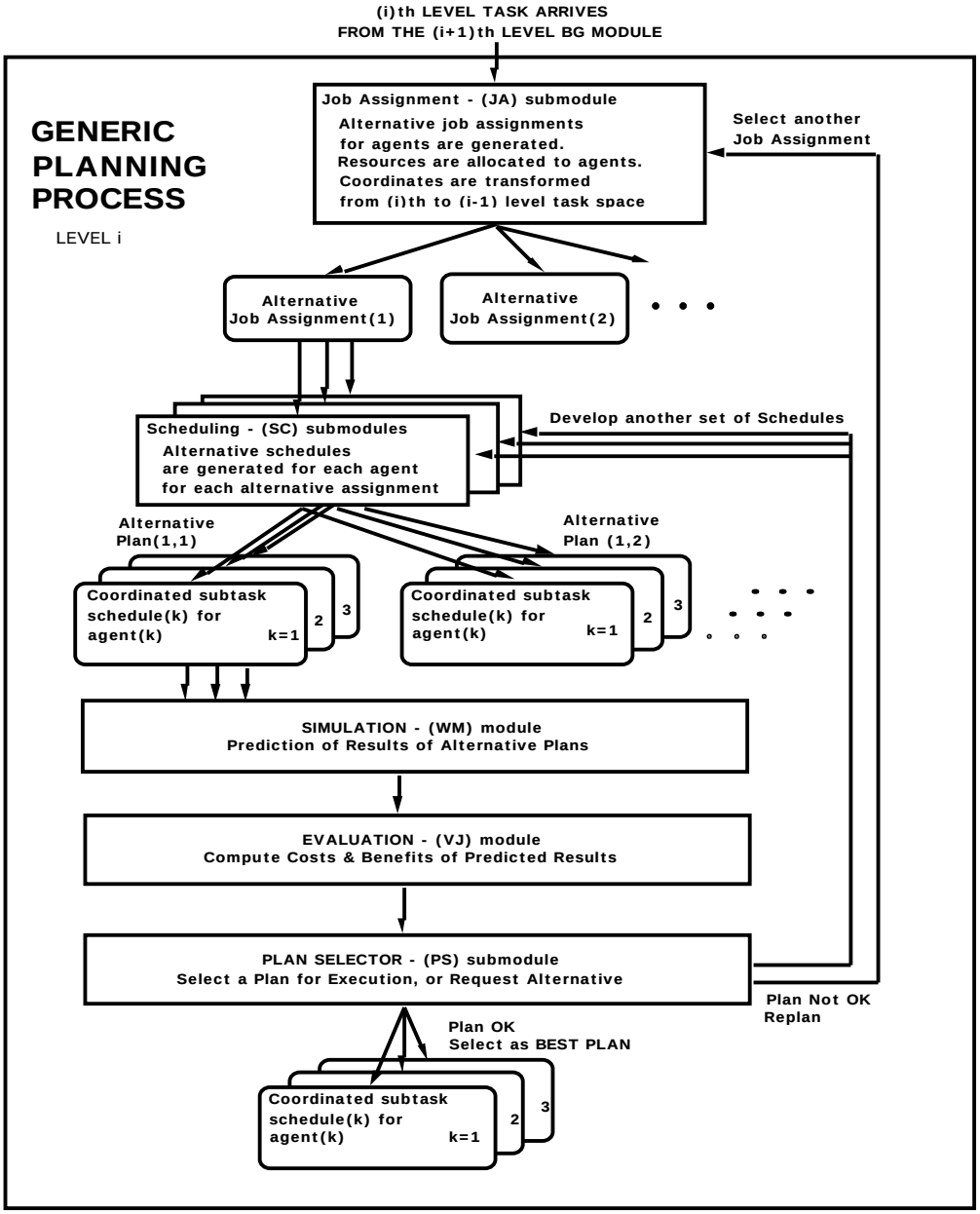


Figure 7. The diagram of planning operations within a BG process. Boxes with square corners represent computational functions. Boxes with rounded corners represent data structures.

Elements 1, 2, and 3 are performed by JA, which assigns jobs and resources to agents and transforms coordinates from task to job coordinates (e.g. from end-point, or tool coordinates to joint actuator coordinates, or from a shop frame of reference to a cell frame of reference). Element 4 is performed by SC, which computes a temporal schedule of subtasks for each agent and coordinates the schedules between cooperating agents (e.g. coordinate joint actuator trajectories to generate a desired end-point trajectory, or coordinate activities between cells to optimize shop work flow).

Together, the assignment of jobs and resources to agents, the transformation of coordinates, and the development of a coordinated schedule for each agent, constitutes the synthesis of a plan. Therefore, output from the JA and SC is a tentative alternative of a plan. JA and SC may generate many tentative plans. Which of these is best (i.e., most efficient, or most effective, or lowest cost, or highest payoff) can be determined by simulating the results of various alternative tentative plans in the World Model (element 5), and by applying a Value Judgment cost/benefit evaluation function to each of the alternatives (element 6). The evaluations are returned to the Plan Selector (PS) process for a decision as to the best plan of action (element 7). This process can be iteratively repeated until the entire universe of possible plans is searched. Heuristic search methods are typically used to reduce the number of alternatives that are actually examined. Eventually, the best plan is submitted to the EXecutors for execution.

These seven elements may not always occur in the order in which they are listed. Specifically, elements 1-4 may be reordered, or intermingled, and iterated many times to generate many alternative assignments of jobs or resources, or alternative schedules, each combination of which need to be simulated and evaluated by elements 5 and 6 before a plan emerges from element 7.

There are two fundamental approaches to planning:

One is to

a) hypothesize a series of actions,
b) use a world model to predict the results,
c) use value judgment to evaluate both actions and predicted results, and
d) choose the action series producing the best evaluation as a plan.

The second is to

a) hypothesize a series of states that lead from the starting state to the goal state,
b) use an inverse model to compute the required series of actions,
c) use value judgment to evaluate both actions and results, and
d) choose the states producing the best evaluation as a plan.

Both approaches involve a world model or inverse to predict the future. Both involve a search procedure to explore the space of possible future actions and result. Both require value judgment to evaluate the cost, benefit, risk, and payoff of each hypothesized plan. Both result in a series of actions and subgoals.

It should be noted that the hypothesized series of actions or states need not be generated in real-time. A hypothesis may be precomputed and simply retrieved from memory. A

behavioral hypothesis may even be built into hardware, so long as it can be triggered when desired. The essence of planning is not in generating a series of actions and states by an exhaustive search of the entire space of all possible actions and states. Such a search could easily be infinite. The essence of planning is in:

1) using a hypothesized series of actions and/or states (however generated) to predict the future,
2) evaluating how desirable the predicted results are, and
3) selecting the behavioral hypothesis with the best predicted results.

Planning can be accomplished through a variety of methodologies. These can range from simple table look-up of precomputed plans (or scripts), to dynamic programming, heuristic AI search in state space or configuration space, or game theoretic algorithms for multi-agent systems. Planning techniques vary widely at different hierarchical levels in the manufacturing enterprise. At the Servo level, planning may consist of performing linear or circular interpolation between actuator goal points. At the Primitive level, planning may consist of finding dynamically desirable paths between specified tool positions. At the E-move level, planning may consist of finding collision-free paths between desired end-effector poses. At the machine Task level, planning may consist of finding sequences of E-moves that accomplish machining or assembly tasks. At the Workstation level, planning may consist of scheduling part handling, inspection, and machining operations. At the Cell level, planning may consist of scheduling batches of parts and tools into workstations. At the Shop level, planning may consist of organizing parts into batches and scheduling them into Cells.

There are many different planning methods in current use. For example, robot programs are often generated by teach methods, or by graphic programming tools. Programs for machine tools tend to be represented as sequences of commands for discrete actions and tool motion expressed as straight lines and circular arcs that are produced by APT programs or CAD systems. At higher levels, dynamic programming, game theory, or genetic algorithm approaches may be used to generate production scheduling and control plans. There is a variety of production planning software available on the market for doing manufacturing resource planning (MRP II), manufacturing requirements planning (MRP I), and computer aided process planning. There are systems for assembly planning, inspection planning, and quality planning, and many for simulation of manufacturing systems and operations. A survey of manufacturing planning systems is available in the SIMA Background Study [21].

At higher levels within a corporate enterprise, management units may perform strategic planning to define goals, set objectives, and define priorities. For reconfiguring manufacturing facilities for production of new products, project management tools may be used to define milestones, allocate resources, assign staff, and set schedules. For building new factories, Program Evaluation and Review Technique (PERT) and critical path methods may be used to plan construction activities.

Current manufacturing practice is that planning is done well in advance of execution, and plans are selected at execution time by simple rules. Modifications, if necessary, are done

by manual ad hoc methods that depend on the skills and intuition of machinists and shop foremen. However, as demands for agility, just in time delivery, concurrent engineering, and real-time optimization increase, the requirements for real-time planning will increase as well.

Planning often is a distributed process that is done by many different planners, working in many different places, at different times. For example, design is a form of planning that typically occurs in a different organization from production, and long before production begins. Process planning may be done at a different place and time than design. Production planning and scheduling typically is done just prior to production. Design specifications and process plans can be viewed as constraints for production planning.

Regardless of how, where, or when planning is done, in each stage of planning, the seven elements listed above typically come into play. Regardless of how plans are synthesized, a plan eventually must specify the decomposition of a task into a set of job assignments for agents, an allocation of resources, and a schedule of subtasks for each agent ordered along the time line. This may need to be done iteratively and hierarchically through many levels and by many different agents in different organizations before the plans are ready for execution.

In all cases, plans must be synthesized prior to execution. In highly structured and predictable environments, plans may be computed off-line, long before execution. For example, completely precomputed plans, such as NC machine tool code can be developed months or years before execution. In this case, real-time planning consists simply of loading the appropriate programs at the proper time. However, this only works for processes where there are no unexpected events that require replanning, or where human operators can intervene to make the necessary adjustments. For processes such as production scheduling that depend on the state of many factors that cannot be known far ahead of time, plans need to be computed close to execution time, and recomputed whenever conditions require. Shop level planning is often done off-line in a batch mode computing environment, once a day, or once a week. But sometimes this is not frequently enough, and production plans often become obsolete shortly after execution begins. As the uncertainty in the environment increases, and as the demands for agility grow, plans need to be computed nearer to the time when they will be executed, and be recomputed as execution proceeds in order to address unexpected events and to take advantage of unpredicted availability of resources.

The repetition rate of replanning must be at least such that a new plan is generated before the old plan is completed. However, in highly uncertain environments, it is best if the planner generates a new plan before the executor completes the first step or two of the old plan. Of course, the ability to react is also a function of the EXecutor, and as manufacturing incorporates more and more on-line sensing and in process measurement, the importance of EXecutor reactive measures will increase as well.

ISAM is structured to support real-time planning. Planning within the ISAM architecture is distributed at many hierarchical levels, with different planning horizons at different levels. This facilitates the ability to replan fast enough to keep up with the demands of a rapidly changing environment. At each hierarchical level, the planning functions compute plans for a planning horizon determined by the temporal and spatial resolution which, together with planning horizon and the execution bandwidth, define the distinguishing characteristics of a level. In the ISAM, on average, planning horizons shrink by about an order of magnitude at each lower level, and the number of subtasks to the horizon remains relatively constant. Thus, the temporal resolution of subtasks increases about an order of magnitude at each lower level. Planning horizons at high levels may span months or years, while the planning horizon at the bottom level typically is less than 30 ms. The number of levels required in an ISAM hierarchy is therefore approximately equal to the logarithm of the ratio between the planning horizons at the highest and lowest levels.

However, ISAM can accommodate either precomputed plans, or planning operations that recompute plans on demand, or on repetitive cyclical intervals. For example, NC part programs can be treated as stored plans and called when appropriate. The ISAM generic planning capability can also accommodate partially completed plans. For example, process plans computed in advance can be used by ISAM real-time planners to provide constraints for real-time generation of production plans.

Regardless of how they are derived, plans must be expressed in a form that can be executed by Executors.

Plan Execution

For each agent in the BG process, there is a job plan and an EXecutor to execute that job plan. An example Executor is shown in Figure 8. Each Executor is a process that runs to completion (i.e., it reads input, computes, and writes output) every time it is triggered. At higher levels, Executors may be event triggered, but at lower levels, they are clock triggered at regular intervals. The trigger rate for each Executor is selected to satisfy the loop bandwidth of its control loop. For example, servo loops for machine tools may require Executors to trigger every 100 _s. Servo loops for industrial robots typically require Executors to trigger about once per ms. At higher levels, Executor trigger rates can be considerably slower. However, at all levels, the Executor trigger rate is many times faster than the planning rate of the BG planner that fills the plan buffer. Outputs from Executors at each level become input task commands to Job Assignors in BG processes at the next lower level at all except the lowest level. At the lowest level, outputs from Executors go directly to Actuators.

At the top of the Executor is a plan buffer that contains the latest job plan generated by the BG planner. This buffer is updated by the Plan Selector as often as is both necessary and possible at typically irregular intervals dictated by the BG replanning cycle. The plan buffer provides a means to isolate the slower and often uncertain timing of the BG

planning process from the faster and typically regular real-time demands of the Executor reactive control loop.

In ISAM, job plans can be represented in any one of a variety of formats, from a simple linear list of subgoals or subtasks to a complete state-table or set of expert system rules. This choice must be made at design time, however, because the design of the Executor depends on the representation chosen. The Executor shown in Figure 8 represents a job plan as a set of three linear lists: a list of planned actions ap(k), a list of planned subgoals gp(k), and a list of planned subgoal times gt(k), where k = 1, 2, . . . , 10. For each step k in the plan, the planned action ap(k) is designed to achieve the subgoal gp(k) at a time gt(k).

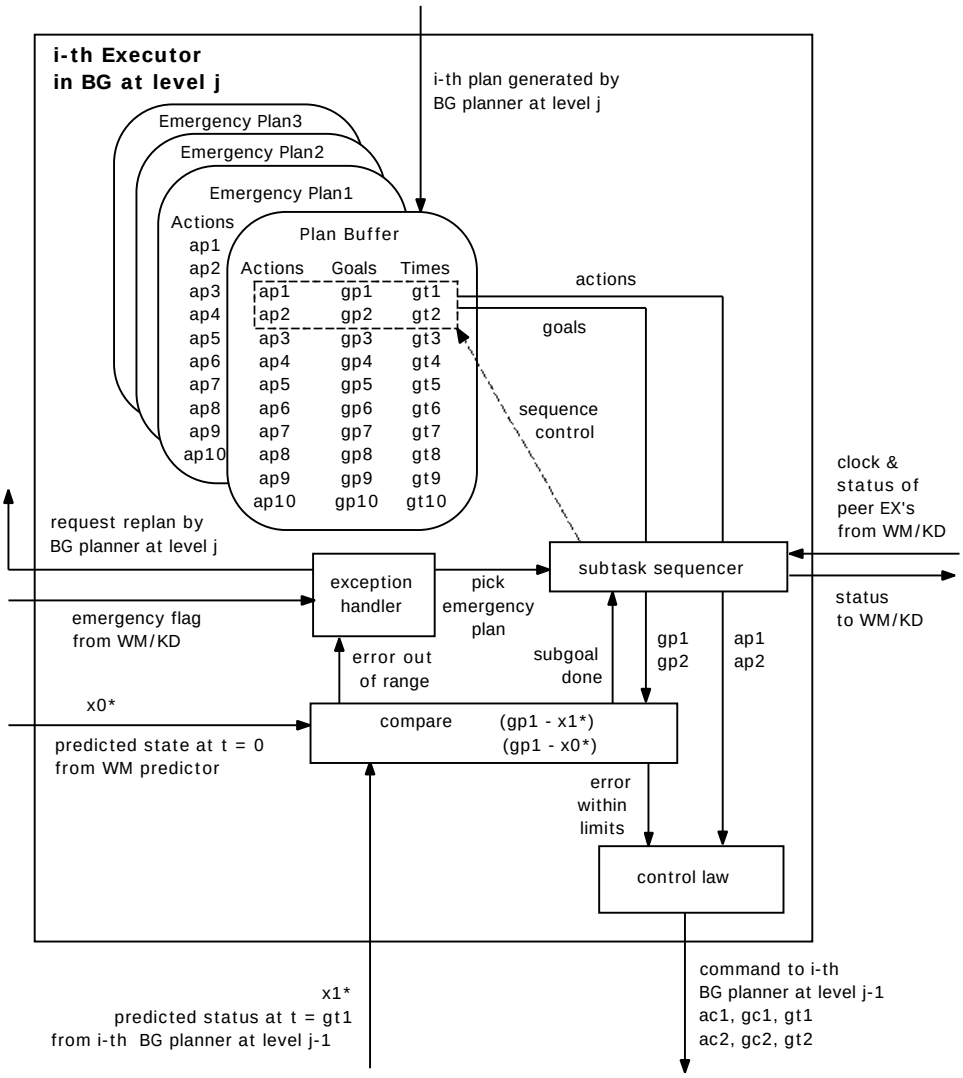


Figure 8. The inner structure of the Executor (EX) process. The plan buffer in the EX is loaded by the Plan Selector. Feedback to the Executor arrives in the form of status

reports from the subordinate BG planner, estimated (or predicted) state feedback from WM, and status from peer EX subprocesses. Output from the EX consists of commands to subordinate BG processes, status to the WM/KD, and requests for replanning to its own planner.

Each time the Executor is triggered, it selects two actions, goals, and goal times as indicated by the subtask sequencer. In addition, it reads input from four sources:
1) feedback from sensors,
2) status from subordinate BG processes,
3) status from peer agents, and
4) commands from the Operator Interface.

Feedback from sensors is provided by a local version of the knowledge database (KD). This local KD is kept up to date by sensory processing (SP) and world modeling (WM) processes. For simple control loops, the KD may contain sampled, digitized, and scaled signals directly from sensors. In more sophisticated systems, the KD may contain filtered, estimated, or predicted state-variables generated by a model based recursive estimation processes. In Figure 8, input from the KD is the predicted state of the world x0* at time t = 0. Status from planner in the subordinate BG processes consists of predicted status x1* at the future subgoal time t = gt1. Status from peer agents consists of the state of subtask sequencers in peer Executor processes.

During each Executor compute cycle, the first of the actions selected by the subtask sequencer becomes a feedforward command sent to the control law. The first of the goals becomes a desired subgoal sent to a compare process. When the compare process detects that the difference between the desired subgoal and the predicted state is less than a goal tolerance, then the compare process reports to the subtask sequencer that the subgoal is done, i.e.,

$$\text{If} \quad (|gp(k) - x0^*| < \Delta g), \quad \text{Then} \quad [\text{report subgoal done}]$$

where

$gp(k)$ = desired subgoal
$x0^*$ = predicted state at $t = 0$
Δg = goal tolerance

When the compare process observes that the error between the planned subgoal and the predicted status from the lower level planner is less than a goal tolerance, the compare process does nothing because the system is performing according to plan, i.e.,

$$\text{If} \quad (|gp(k) - x1^*| < \Delta g), \quad \text{Then} \quad [\text{do nothing, plan is on-track}]$$

where

$x1^*$ = predicted status at $t = gt(k)$

For continuous systems, the predicted state feedback x0* and x1* may be time dependent state variables that can be used to compute trajectory following-errors. For discrete event systems, x0* and x1* may represent predicted status of a sensor, or busy/done from a lower level BG process.

Note that the subtask sequencer in Figure 8 selects two subtasks and two subgoals that are sent to the subordinate BG process. This is so that the subordinate BG planner will always have a commanded action and goal that lies beyond its planning horizon.

When the compare process observes that the error between the planned subgoal and the predicted status from the lower level planner is greater than a goal tolerance but less than an emergency threshold, the compare process computes a feedback error than can be used by the control law to compensate for the error and servo the system back to the plan, i.e.,

If $(\Delta g < |gp(k) - x^*1| < Te)$, Then [compute error and apply compensation]

where

Te = emergency threshold

Emergency Action

When the compare process observes that the error between the planned subgoal and the predicted status from the lower level planner or the WM is greater than an emergency threshold, or if an emergency flag is set in the KD, then the compare process reports that the error is out of range and the subtask sequencer branches to an emergency routine, i.e.,

If $([\ |gp(k) - x1^*|$ or $|gp(k) - x1^*|\ ] > Te)$
or
If (emergency flag is set), Then [select emergency routine and replan]

Under these conditions, an exception handler is activated to select an appropriate emergency plan from a stored library to replace the current plan. For example, an emergency plan may be selected to slow down and stop all on-going operations. Or an emergency plan may be to back up or retract to a safe position. The exception handler simultaneously requests the Planner to generate a new plan.

Executor processes provide the first line of defense against failure at every level in the ISAM hierarchy. Each Executor acts immediately and reflexively, as soon as an error condition is detected, to correct it by feedback compensation, if possible, or to invoke preplanned emergency actions, if necessary. As soon as a compare process detects that an emergency or out-of-limits condition has occurred, or that a problem has been encountered by a subordinate planner, the exception handler causes the subtask sequencer to branch immediately to an emergency action plan. Simultaneously, the BG planner is

requested to generate a new plan. If this procedure is successful, the problem is solved and there is no need to involve higher level processes.

However, if error correction and replanning is unsuccessful within BG at the j-th level, the Executor at the next higher level (j+1) will detect it and attempt a reactive compensation strategy at that level. If this fails, the BG planner at the j+1 level will select or generate a new plan. If this fails, the problem will be detected at the next higher level (j+2). Thus, errors are addressed and corrected at the lowest level possible where response can be quickest. Errors that are within tolerance are handled by servo feedback compensation. Errors that are out of tolerance, or are due to conditions that cannot be corrected by feedback compensation, trigger immediate emergency action.

Uncorrected errors ripple up the hierarchy, from executor to planner within each level and from lower to higher levels within the BG chain of command, until either a work-around solution is found, or the entire system is unable to cope, and a total system failure occurs. At each level, preplanned emergency actions such as stop, or retract allow time for higher level processes to replan and continue if possible, or invoke a different strategy, or abort and start a different job. At any level, errors are routinely reported and requests can be made to a human operator for assistance.

Timing

Time is a critical factor in sensing and control. The most obvious thing about time is that is flows in one direction and there is a unique point called the present that divides the past from the future. For each ISAM node, the origin of the time line is the present (where t = 0). All actions take place and all sensory observations occur at the instant t = 0. Everything to the right of t = 0 is in the future. Everything to the left is in the past. If we quantize time with a ticking clock, the present exists during the zeroth tick of the clock that separates the future from the past.

Figure 9 is a diagram of the temporal relationships involved in representing the time line. At each level of the ISAM reference model, time is represented at a different scale. The triangles on the time line represent significant points in time. At the lowest level, time is represented at the highest resolution. At each successivelyhigher level, the planning horizon expands by an order of magnitude and the resolution along the time horizon decreases by an order of magnitude. At low levels, time is measured in milliseconds, seconds and minutes. At high levels, time is measured in hours, days, weeks, or years. At all levels, t = 0 separates the future from the past. At all levels, future plans scroll left to be executed at t = 0, and then become past events which recede into the past as a historical trace. At low levels, time appears to flow rapidly. At higher levels, time appears to move more slowly.

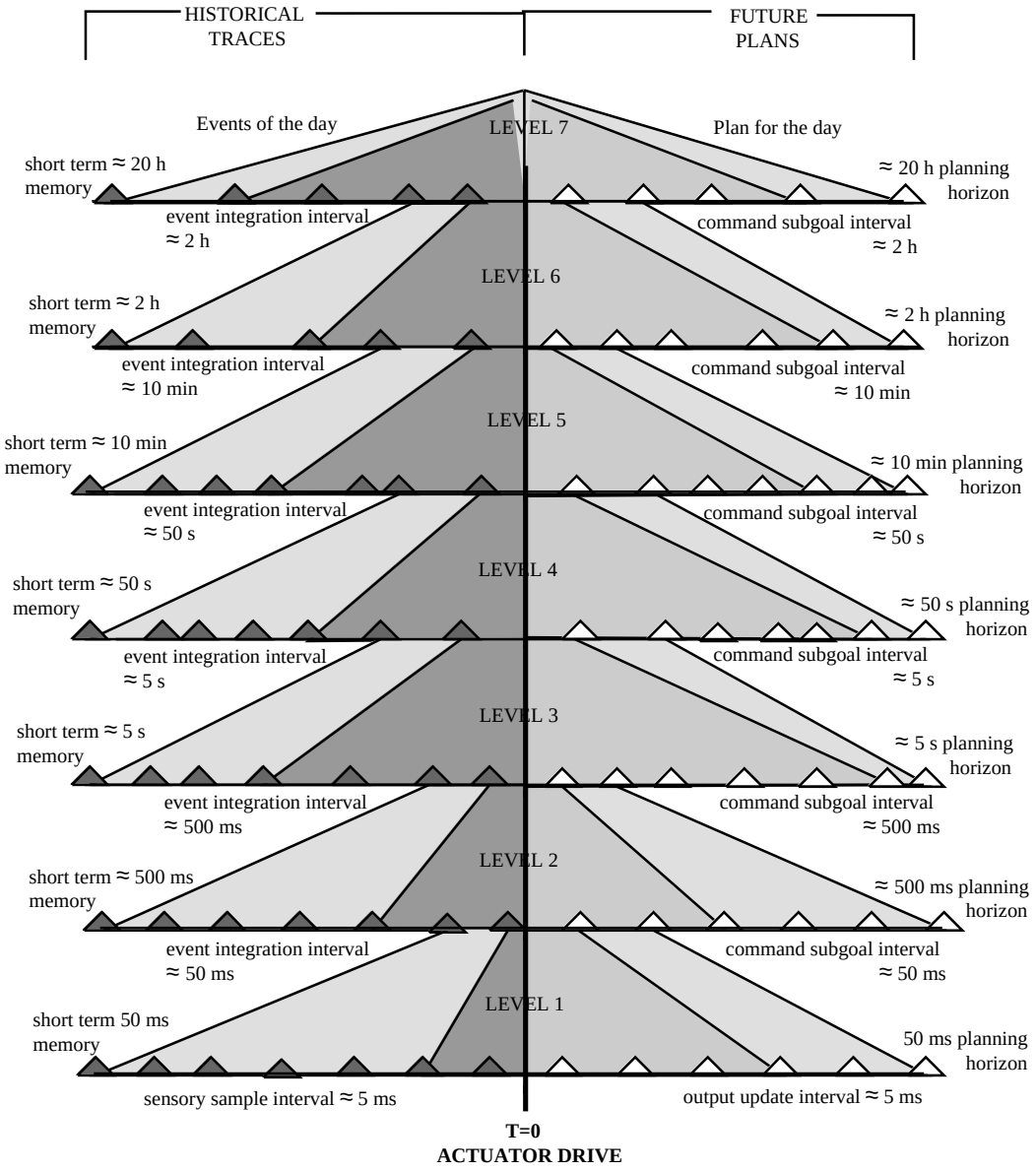


Figure 9. A timing diagram for the ISAM reference model architecture showing the temporal range and resolution for planning and short term memory at different hierarchical levels.

At each level, input tasks are decomposed into plans that contain sequences of subtasks that are output as commands to the next lower level. To the right of t = 0, the triangles along the time line represent goals and subgoals. Lines between triangles represent task activity that is required to achieve subgoals. Inputs to each level are commands that specify the current and next task goal in the higher level plan. Each level has a planner that generates a set of planned subtasks and subgoals out to its own planning horizon.

The plan to the current goal is shown as a gray area. The plan beyond the current goal to the planning horizon is shown as a hatched area. At lower levels, plans are highly detailed but extend only a brief period into the future. At higher levels, plans are less detailed but extend further into the future. The number of subtasks in a plan is approximately the same at all levels.

Each level has a short-term memory that maintains a historical trace of about ten events that are significant to that level. At each level, events held in short-term memory are grouped (clustered or integrated) onto patterns that are detected and labeled as higher level events. To the left of t = 0, the triangles mark the beginning and ending of detected events. The horizontal lines between triangles correspond to the historical intervals during which events occur. Events may include the successful accomplishment of planned goals, or the failure to achieve goals. Events may represent unexpected happenings, surprises, or emergencies. At the lowest level, events represent single samples of sensor outputs. At higher levels, events represent patterns or clusters of lower level events (or subevents). The detection of an event may signal the occurrence of a state or situation in the world. In Figure 9, the clustering of a set of subevents into a higher level event is shown as a gray area. The set of subevents remaining in short-term memory that belong to a previously detected event are shown as a hatched area.

Low level events have high-frequency components, but extend only briefly into the past. At higher levels, events have lower frequency components, but cover a longer historical period. The number of events stored in short-term memory is approximately the same at all levels.

Note that Figure 9 shows symmetry between future plans and past events. The ISAM reference model maintains a historical trace in short-term memory with duration approximately equal to the planning horizon at each level. Thus, the number of events in short-term memory is approximately the same as the number of subtasks in a plan for each level.

The diagram in Figure 9 indicates specific timing for planning horizons and short-term memory. These timing intervals are examples only. For any particular application, timing will depend on the dynamics of the system being controlled. Typically, the numerical values of timing parameters are determined by the dynamics of the physical system being controlled. Events must be sampled at a rate that is at least twice the highest frequency in the input signal in order to prevent aliasing. This is the theoretical limit set by the Nyquist criterion. In practice, the sampling rate is typically set at ten times the highest frequency in the input. Sensor signals are typically filtered before sampling to assure that this criterion is met. Similarly, outputs to actuators must be updated fast enough to control the highest frequency modes of the system being controlled.

In a feedback control system, the repetition rate and latency between sensing and acting are critically important for system stability. The repetition rate of the sensor->SP->WM->EX->actuator computational cycle corresponds to the bandwidth of the control loop at

that level and the latency in the sensor->SP->WM->EX->actuator pathway corresponds to a phase delay. In order to insure stable performance, the repetition rate should be at least an order of magnitude faster than the average interval of time discretization in the plan, and hence about a hundred times the reciprocal of the planning horizon at each level. Other design criteria related to gain and phase shift are required to assure system stability for specific implementations.

It is important that systems with nested control loops maintain separation between the loop bandwidth of the inner and outer loops. Otherwise, the inner and outer control loops may interact in very undesirable ways to produce instabilities. In Figure 9, the bandwidth of each successively higher level loop is defined to be approximately an order of magnitude lower than the level that it controls. For example, at the output from level 1, feedback loops are closed and commands are sent to actuators every 5 ms. At the output from level 2 (input to level 1) commands are sent to groups of actuators every 50 ms. At the output from level 3 (input to level 2) new commands are sent to primitive level subsystems approximately every 0.5 s. At the output from level 4 (input to level 3) new commands are sent to elemental move level subsystems on average every 5 s. At the output from level 5 (input to level 4) new commands are sent to machines on average every 50 s. At the output from level 6 (input to level 5) new commands are sent to groups of machines in workstations about every 10 minutes. At output from level 7 (input to level 6) new commands are sent to cells of workstations about every 2 h. At the input to level 7, new commands to the shop are received about every 24 h. These specific times are for illustration only. For different environments, timing requirements may vary.

This hierarchical structure does not necessarily prevent higher levels from responding quickly to new information in the world model. At each level, any executor can respond within one compute cycle trigger interval. However, temporal integration of events in the sensory processing hierarchy limits the bandwidth of the control loop that is closed through each level and thus prevents higher levels from fluctuating rapidly in response to high frequency components in sensory feedback.

The particular grouping of actuators into subsystems and systems is application dependent. For example, level 1 might control actuators on machine tools, robots, and inspection systems. Level 2 might coordinate actuators to define desired tool motion or robot gripper position. Level 3 might develop obstacle free paths and coordinated motions for arms, grippers, and tools. Level 4 might coordinate activities between subsystems of a machine tool or robot. Level 5 might plan and control coordinated activities between a robot, machine tool, andmaterial delivery system within a manufacturing workstation. Level 6 might plan and control coordinated activities of a group of workstations in a group technology cell. Level 7 might plan and control activities of cells in a shop.

Integration of Reactive and Deliberative

Activities of Job Assignors, SChedulers, WM simulators, VJ cost/benefit evaluators, and Plan Selectors are deliberative functions that are focused on planning for the future. Planning is designed to discover a trajectory through time and state space from an anticipated starting state to a future goal state. The servo functions of the EXecutors are reactive, or reflexive, and are focused on following the plan as closely as possible as events occur in the present. The EXecutors use control laws to compensate for the error between the reference trajectories generated by planning and the feedback generated by Sensory Perception and World Modeling. The ISAM architecture thus mixes deliberative and reactive elements in BG processes at each level of the architecture.

As illustrated in Figures 1-3, computational nodes at all levels process input from sensors, or from lower level SP processes, so as to estimate the state of the World as perceived at that level's resolution in space and time. At all levels, BG processes accept task commands from higher levels, and close a control loop. At each level, knowledge is represented in a form, and with a spatial and temporal resolution that meets the processing requirements of the nodes at that level. At each level, there is a characteristic loop bandwidth, a characteristic planning horizon, a characteristic kind of task decomposition, a characteristic range of temporal integration of sensory data, and a characteristic window of spatial integration. At each level, information is extracted from the sensory data stream in order to keep the world model knowledge database accurate and up to date.

At each level, sensory data is processed, entities are recognized, world model representations are maintained, and tasks are deliberatively decomposed into parallel and sequential sub tasks, to be performed by cooperating sets of agents. At each level, feedback from sensors reactively closes a control loop allowing each agent to respond and react to unexpected events. At each level, tasks are decomposed into sub tasks with subgoals, and behavior is planned and controlled. The result is a system that tightly integrates deliberative and reactive capabilities, not at a single centralized site, but distributed over the entire hierarchy in all of the ISAM nodes. Thus, reactive feedback loops are simultaneously closed at all levels of resolution in space and time. Deliberative plans are simultaneously being generated at all levels of range and resolution in space and time. And integration between planning and reaction is simultaneously taking place within every ISAM node in the entire architecture.

World Modeling (WM)

In each node, WM processes perform four basic functions:

1) Maintenance of the Knowledge Database (KD).

WM processes maintain the KD, keeping it current and consistent. WM processes update the KD based on correlation and variance between world model predictions and sensory observations at each node. This enables recursive estimation of attributes and state-variables over any desired window in time and space. WM processes update both images and symbols and perform transformations from iconic to symbolic and vice versa. WM processes enter new entities and events into the KD, and delete entities that are estimated to no longer exist in the world. WM processes maintain pointers between KD data structures that define relationships between entities, events, situations, and episodes. WM processes also maintain pointers that link images and maps with entities, events, classes, lists, and frames.

2) Prediction.

WM prediction processes generate expectations of sensory inputs that enable SP to compare expectations with observations and perform correlation and predictive filtering. WM uses knowledge in the KD to predict where objects in the environment will appear to each sensor at the next sample. Knowledge of object dynamics, sensor motion, behavioral actions, and projective geometry enable WM processes to predict how objects can be expected to move in an image between successive frames. Knowledge of camera state enables the WM to transform predicted images into sensor coordinates and to compensate for effects of self motion on sensory input.

Prediction of sensory input enables SP processes to be surprised by non-events, e.g., to set a flag when something expected fails to happen. Prediction of sensory input also enables SP to focus attention and select algorithms that are appropriate for processing the expected sensory input. Prediction of feature states can suggest where edges, corners, and holes can be expected to appear in a camera image, or where surfaces can be expected to be encountered by tactile probes. Prediction of object state variables can enable SP to anticipate how objects are expected to move and where they are expected to appear in an image or on a map. Predictions of target state variables enable BG to track target motion and maximize the performance of feedback control loops.

When there is a high degree of correlation between predictions and observations, the WM can increase the confidence assigned to information stored in the KD. When there is significant variance between predictions and observations, the WM can reduce the level of confidence assigned to information stored in the KD. If confidence in the information in the KD falls below some threshold, the WM may search for an alternative interpretation of the sensory data.

3) Query response

WM provides data retrieval functions in support of BG, VJ, and SP processes within the same node and WM processes in other nodes. WM responds to "What Is?" queries from the BG planning and control functions regarding the estimated state of the world or state of the controller. At low levels, this may require nothing more than to return the contents of a symbolic variable. At higher levels, it may require question answering and logical reasoning services that deduce responses to queries for information that is not explicitly stored in the KD. WM processes also respond to queries from the Operator Interface.

4) Simulation.

WM simulators use knowledge about the world in the KD to predict the results of hypothesized actions generated by BG planning processes. Structural and dynamic models and rules of physics and logic embedded in WM simulator processes enable the WM to predict the results of current and hypothesized future actions. Inverse models enable the WM to compute what actions would be required to produce a desired series of goal states. This provides the ability of the WM to respond to "What If?" queries with simulated predictions of what the results would be if a particular string of hypothesized actions were performed, or what actions would be required to achieve a particular string of hypothesized subgoal states.

WM simulation processes transform symbolic representations into iconic images that can be used as masks or windows to support attention and model matching. Images developed by WM simulation processes can be used for template matching, area based correlation, and visual servoing of cameras and probes. Simulation can be used to build a graphic representation of what particular parts or assemblies are expected to look like or how they should be positioned and oriented in the work place. Simulation can be used to analyze predicted results of machining processes, or predicted flow of materials through a manufacturing facility. Simulation can even show how a new or modified product might be expected to look.

At Primitive or E-move levels, simulation might be used to predict the shape of a part feature that would be produced by a segment of a NC machine tool program. At the Machine or WorkStation level, simulation might be used to predict how long it might take to perform an assembly task. At the Cell level, simulation might predict the work flow from a particular production plan. At the Shop level, simulation might be used to predict inventory levels and anticipated sales for some period in the future.

These four functions are illustrated in Figure 10.

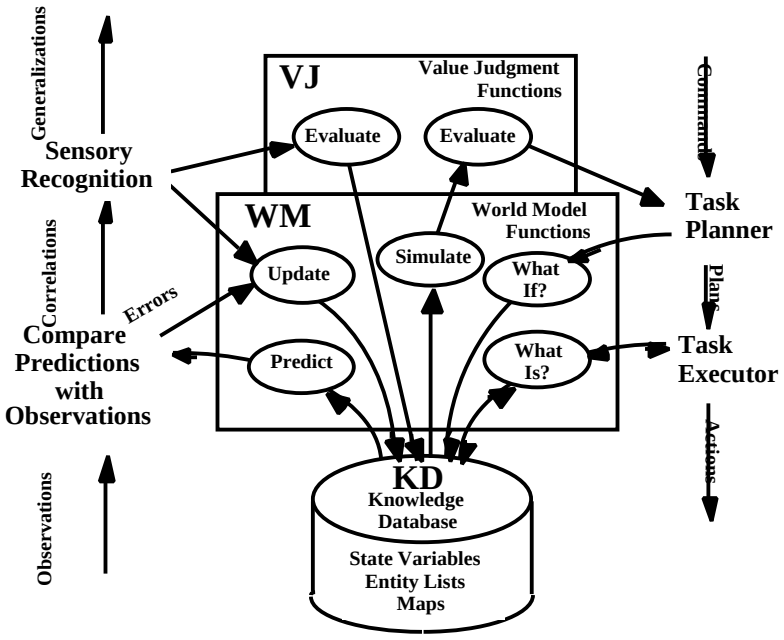


Figure 10. World Modeling (WM) and Value Judgment (VJ) processes. WM and VJ processes typically exist in every node of the ISAM architecture.

All four basic WM functions are well understood in the computer science and systems engineering community [22]. Database management, recursive estimation and prediction, dynamic modeling, and simulation are well practiced arts that have been applied many times in many different domains both in the laboratory and in the commercial world.

The ISAM WM groups these four processes together and specifies the real-time constraints that drive the design of the computing environment necessary to implement these processes in each ISAM node. The ISAM architecture distributes the WM processes and KD information over the ISAM nodes such that the amount of knowledge in the KD and processing in the WM in any node is limited to only what is needed for BG and SP processes in that node. Sufficient memory and computing power can then be allocated to WM and KD in each node to achieve the real-time performance requirements.

Knowledge Database (KD)

KD contains the data structures and information needed to support the BG, SP, WM, and VJ processes in each node. Types of data structures include scalars, vectors, arrays, symbols, strings, lists, structures, classes, objects, frames, and graphs. Information in the

KD includes both symbolic and iconic representations of signals, images, maps, entities, events, places, and situations along with state variables and attributes that describe or modify them. Knowledge in the KD may also include rules, equations, recipes, schema, programs, algorithms, procedures, narrative descriptions, and formulae. This knowledge enables the WM to analyze the meaning of past sensory experience and to predict what might happen under hypothesized future scenarios.

Entities are data structures that represent features, objects, or groups that exist in the world, or in the world model. An entity frame is a symbolic data structure that consists of a name, a list of attributes and state variables, and a set of pointers that define relationships with other entities or events. These relationships can represent semantic or pragmatic meaning and enable symbolic reasoning.

An entity image is an iconic array of pixels each of which is labeled with (i.e., has a pointer to) the name of the symbolic entity to which it belongs. Entity images are important for visualization, image processing, and geometric reasoning.

Symbolic events are representations of state changes, or states or situations that occur at particular times and places, or of a sequence of states, or situations, that transpire over intervals of time and space in the world. A symbolic event can also be represented by a list, or frame, consisting of a name, a set of attribute-value pairs, and a set of pointers that define relationships with other symbolic events and entities.

Rules and equations, such as If/Then rules, the predicate calculus, differential equations, control laws, geometrical theorems, and system models can express physical and mathematical laws and describe the way the world works and how things relate to each other in time, space, and causality.

Structural models describe how physical structures are kinematically connected and how forces and stresses are distributed. Dynamic models define how force, motion, and inertia interact with each other in time and space.

Task knowledge is knowledge of how to act in order to accomplish goals. Task knowledge includes skills and abilities required for manipulation, locomotion, communication, and attention. For example, task knowledge may describe how to drill a hole, weld a seam, repair a TV, or assemble an automobile. Task knowledge is primarily used by BG for planning and control.

Images are two-dimensional arrays of attribute values. Images may be generated in a number of ways. For example, an image may be formed by a drawing or sketch on a sheet of paper, or by the projection of electrons on the face of a cathode ray tube, or by the optical projection of light from a scene in the world through a lens onto an array of photoreceptors (or pixels) such as a CCD TV camera. An image may also consist of attributes such as spatial or temporal gradients, or disparity, or range, or flow, that are computed from other images over spatial windows and temporal intervals. An image

may also be generated by internal mechanisms (such as a computer graphics engine), from information stored in symbolic entity frames.

Maps are also two-dimensional arrays of pixels, wherein icons or names of symbolic entities, in addition to attributes, are attached to pixels. A map may represent the layout of a plant floor, or the transportation routes between manufacturing plants.

The KD has three parts:
1. Immediate experience consists of signals from sensors and estimated or predicted attributes and state variables generated by recursive estimation from sensory observations. Immediate experience persists only so long as it is refreshed by incoming sensory data.

2. Short-term memory consists of symbolic representations of entities, events, and situations that are the subject of current attention. Entities-of-attention typically are either specified by the current task, or are note-worthy entities observed in recent sensory input. Predicted images generated from short-term memory can be used to filter, mask, or window incoming data, or to fuse incoming sensory observations with internally generated images.

3. Long-term memory consists of symbolic representations of all the entities, events, situations, and rules that are known to the intelligent system. Symbolic entity and event frames may be transferred from short-term memory to long-term memory, or vice versa. When a long-term entity is specified as the object of a task command, it is transferred into short-term memory to become an entity-of-attention. When a short-term entity or event is recognized or otherwise designated as note-worthy it is transferred into long-term memory for permanent storage.

Sensory Processing (SP)

SP processes process data from sensors, including visual, optical, acoustic, tactile, position, velocity, acceleration, force, torque, temperature, pressure, magnetic, electrical, and chemical sensors. SP processes contain filtering, masking, blurring, expansion, contraction, differencing, correlation, shifting, scrolling, warping, matching, and recursive estimation algorithms, as well as feature detection, pattern recognition, clustering, chaining, and grouping algorithms. Interactions between WM and SP processes can generate a variety of filtering and detection processes such as Kalman filtering and recursive estimation. Interactions between SP and WM processes can produce Fourier transforms, phase-lock loops, and tracking filters, as well as detection, recognition, and model matching. All of these algorithms can be performed on either attributes of entities in symbolic frames, or attributes of pixels in images or maps. In the vision system, SP processes process images to measure brightness, color, and range, to compute gradients and discontinuities, motion, and stereo disparity. SP processes may utilize a variety of event detection and pattern recognition algorithms to analyze signals and images and to extract the information needed for manipulation, locomotion,

communication, attention tracking, and spatial-temporal reasoning. Regions of the image in which predictions and observations are highly correlated are labeled recognized, or understood. Regions of the image where significant differences exist between what is predicted and what is observed can be labeled unrecognized and possibly worthy of attention.

In tactile inspection systems such as coordinate measuring machines, SP processes may process touch information to measure dimensions of features such as surfaces, holes, pockets, and grooves. Measured values may be compared with design values to determine whether parts are manufactured within tolerance.

At higher levels, SP processes may compute the flow of materials through a plant, or the flow of inventory through a "just-in-time delivery pipeline, or the yield in a semiconductor plant. Sensory perception is the process of extracting this type of information from sensors and reports. World modeling is the maintenance of this knowledge in the knowledge database, and providing predictive simulations for planning and control. The operator interface makes it possible for operators to visualize this type of data in various presentation formats.

Value Judgment (VJ)

VJ processes contain algorithms for computing the cost, risk, and benefit of actions and plans, for estimating the importance and value of objects, events, and situations, and for assessing the reliability of information. VJ processes may compute Bayesian or Dempster-Schafer statistics on the reliability of information about the world based on the correlation and variance between observations and predictions in order to assign confidence values to data from various sources. VJ processes evaluate perceived and planned situations thereby enabling behavior generation to select goals and set priorities. VJ computes what is important (for attention), and what is rewarding or punishing (for learning).

An Example Node

An example of the relationships and interactions between the BG, WM, KD, SP, and VJ processes in a typical node in the ISAM architecture is shown in Figure 11. The Behavior Generating (BG) processes contain sub processes of Planner (PL) and Executor (EX). Planner has sub-subprocesses of Job Assignment (JA), Scheduling (SC) and Plan Selector (PS). The World Modeling (WM) process contains the Knowledge Database (KD), with both long term and short term symbolic representations and short term iconic images. In addition, WM contains a Simulator where the alternatives generated by JA and SC are tested for comparison in PS. The Sensory Processing (SP) process contains filtering, detecting, and estimating algorithms, plus mechanisms for comparing predictions generated by the WM process with observations from sensors. It has algorithms for recognizing entities and clustering entities into higher level entities. The Value Judgment (VJ) process evaluates plans and computes confidence factors based on the variance between observed and predicted sensory input.

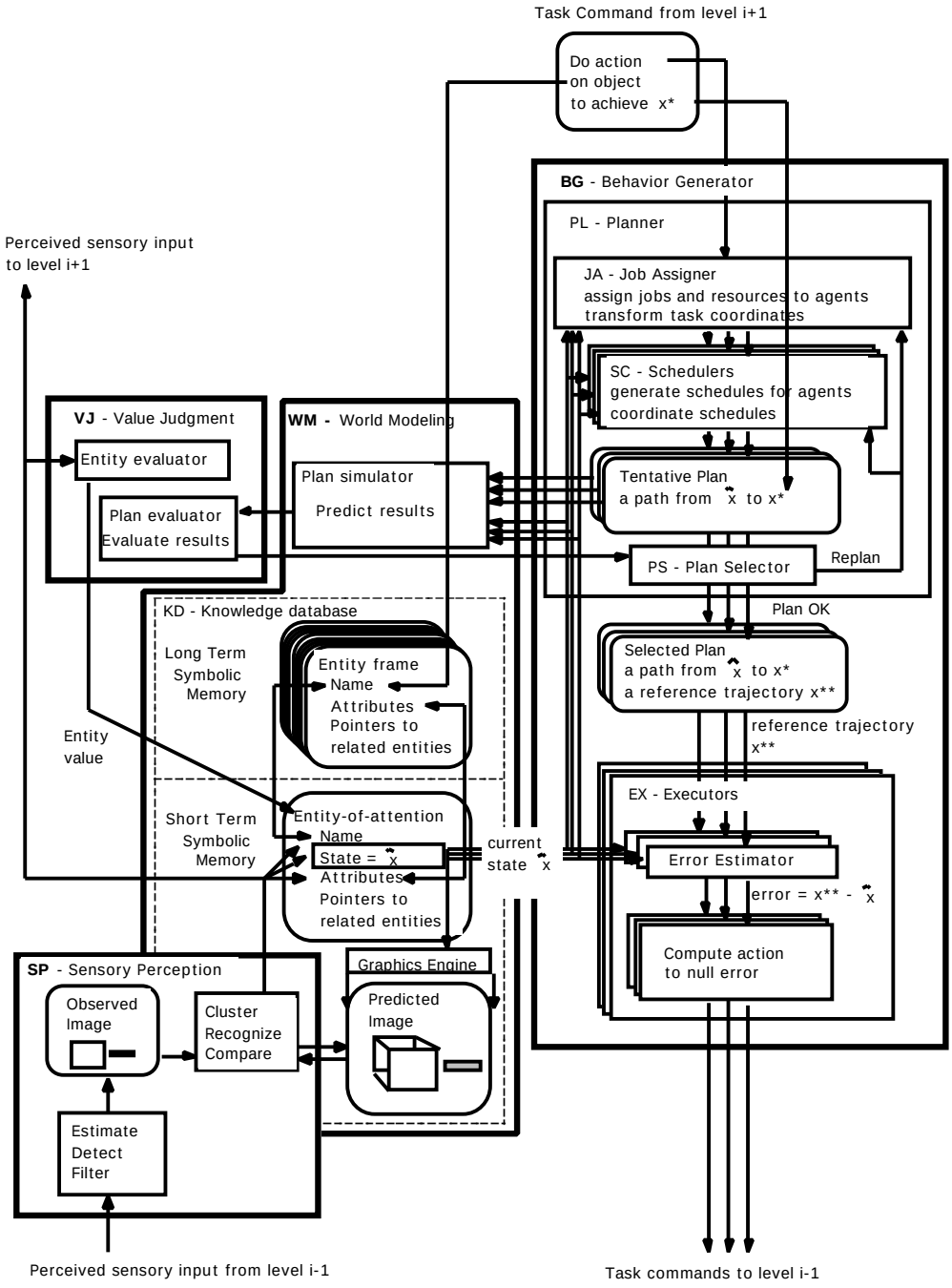


Figure 11. Relationships within a single node of the ISAM architecture A task command from level i+1 specifies the goal and the object. The object specification selects an entity from long term memory and moves it into short-term memory. The BG process generates a plan that leads from the current state to the goal state. The EX sub processes execute the plan using estimated state information in the WM KD short-term memory. The SP process processes sensory input from level i-1 to update the WM KD within the node, and sends processed information to level i+1.

Each node of the ISAM hierarchy closes a control loop. Input from sensors is processed through sensory processing (SP) processes and used by the world modeling (WM) processes to update the knowledge database (KD). This provides a current best estimate of the state of the world x. Depending on the concept of estimation used, a particular control law is applied to this feedback signal arriving at the EX sub process. The EX sub process computes the compensation required to minimize the difference between the planned reference trajectory and the trajectory that emerges as a result of the control process. The value of "best estimate" is also used by the JA and SC sub-subprocesses and by the WM plan simulator to perform their respective planning computations. It is also used in generating a short-term iconic image that forms the basis for comparison, recognition, and recursive estimation in the image domain in the sensory processing SP process.

Elementary Loop of Functioning

The bandwidth of the control loop through each node is determined by the sampling rate of the sensors, the filter properties of the SP and WM sub processes, and the computation update frequency of the EX sub process. The control loop, or elementary loop of functioning, through a typical control node is shown in Figure 12.

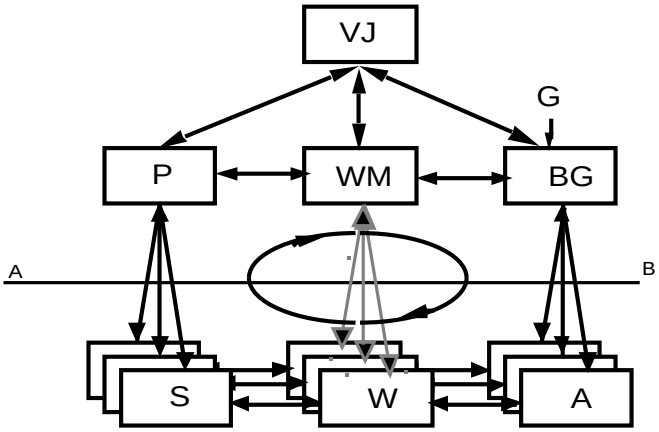


Figure 12. An Elementary Loop of Functioning

A task command specifying a goal G arrives into BG-process of a generic control node. The function of the BG process is to transform the task into a set of sub tasks which should be submitted to the set# of actuators {A} of the subsystems. All subsystems operate in the set of their Worlds {W}; the latter are equipped by the set of sensors {S}; the signals of the sensors are integrated within the sensory processing process (SP) which updates the World Representation contained in the Knowledge Database. The best sensor integration, best model, and best control solution are selected with the help of the Value Judgment process (VJ). The dotted lines between the WM and the world {W} indicate a presence of the virtual correspondence between the real world W and its representation in the world model WM; these links are not real channels of communication but they exist virtually. The knowledge in WM is the system's best estimate of the real state of W. The

virtual links represent a noisy channel through the sensors {S}, the SP and WM processes by which the system perceives the world. To the extent that this perception is an accurate, or at least adequate, representation of reality, behavior is more likely to be successful in achieving the goal{G}. To the extent that the perception is incorrect, behavior is less likely to be successful.

Figure 13 describes the Loop of Functioning consisting of two domains -- Controller System and Controlled System. The line AB is a divider between these domains. It denotes the fact that the processes above this line are components of the controller system while the processes underneath are the controlled system.
However, we will see that in the ISAM hierarchy, the controlled system can be regarded as a "control system of higher resolution", which leads to the multilevel structure of Figure 13, where a second control level is shown. The upper, or outer loop, has lower resolution in both time and space. The outer loop has a lower bandwidth, a longer planning horizon, a larger span of control over subordinate agents, and typically deals with larger range of space. The inner loop, has a higher loop bandwidth, a shorter planning horizon, a smaller span of control, and deals with a smaller spatial range with higher resolution.

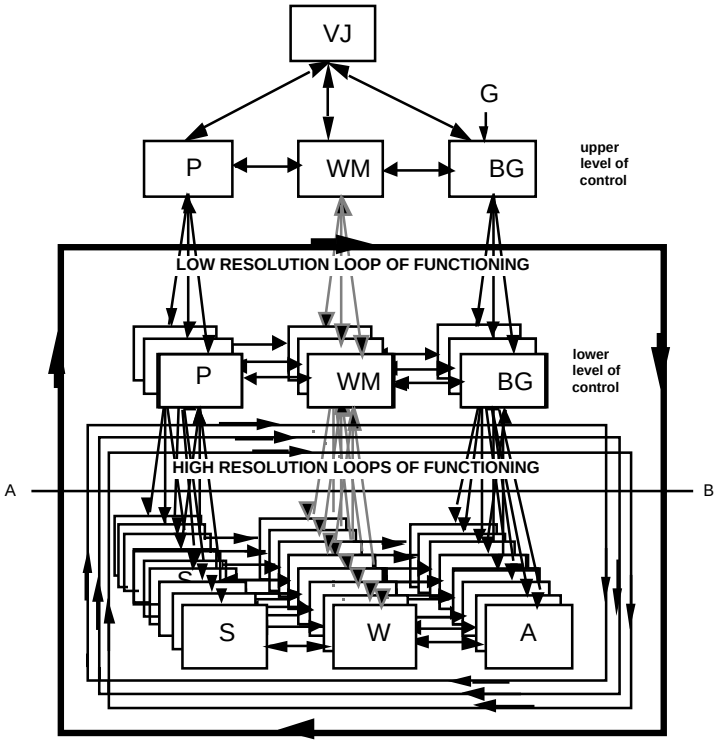


Figure 13. An Elementary Loop with 2 Control Levels

Each node in the hierarchy shown in Figure 3 closes a loop of functioning. At each higher layer in the hierarchy, each node generates an additional outer loop of functioning that is closed around the nodes in the sub trees beneath it. At each higher level, the outer loop has a bandwidth that is an order of magnitude lower, with a planning horizon that is an order of magnitude longer, and a span of control that is an order of magnitude larger. At each higher level, representations of knowledge cover a larger spatial and temporal range, with correspondingly lower resolution.

It is easy to anticipate that when the number of levels grows, the number of loops nested in each other grows too. Each process of the outer loop is obtained as a result of generalization of the processes of the adjacent inner loops. The whole inner loop (its full set SP-WM-BG-A-W-S) serves for the outer loop as a triplet of its A-W-S, or as the virtual actuator, virtual sensors, and virtual world.

Virtual Actuators and Virtual Sensors

Only the lowest level are EXecutors directly connected to actuators that carry out output commands. However, to every EXecutor at every level it appears as if it were connected to an actuator capable of directly executing its output commands. This is because when any Executor sends commands to its subordinate BG process, and receives status feedback exactly as if the subordinate BG process were an actuator. Thus, to an Executor, its subordinate BG process and the entire hierarchy beneath acts like a "virtual actuator." The virtual actuator makes it appear to the Executor process at every level as if its output commands are directly executable.

In a similar way, sensors that measure the environment exist only at the lowest level in the ISAM hierarchy. However, when a SP process receives input from its subordinate SP processes, it appears to that SP process as if it were connected to a sensor that directly measures the state variables that the subordinate SP process computes. Thus, each SP process is a "virtual sensor" to the SP processes above it in the sensory perception hierarchy.

At all levels, the world model represented in the KD and WM provides an internal "virtual world."

Three Perspectives of ISAM

ISAM is a complex and multifaceted architecture. Figure 14 illustrates ISAM from three different perspectives. On the left is the organizational hierarchy. This illustrates how each node in the ISAM reference model is an organizational unit that receives commands from a superior unit and sends commands to one or more subordinate units.

A second perspective shown in the middle of Figure 14 illustrates the computational hierarchy, or chain of command, supported by the world modeling and sensory processing functions. This perspective represents a "side view" of a single chain of command through the organizational hierarchy. This perspective emphasizes how a

control loop is closed through each node. It illustrates how each BG process is supported by WM and VJ processes plus a KD that are maintained locally and serviced by SP functions. Tight coupling between BG and WM supports planning. Tight coupling between WM and SP supports recursive filtering, recognition, and world model maintenance functions.

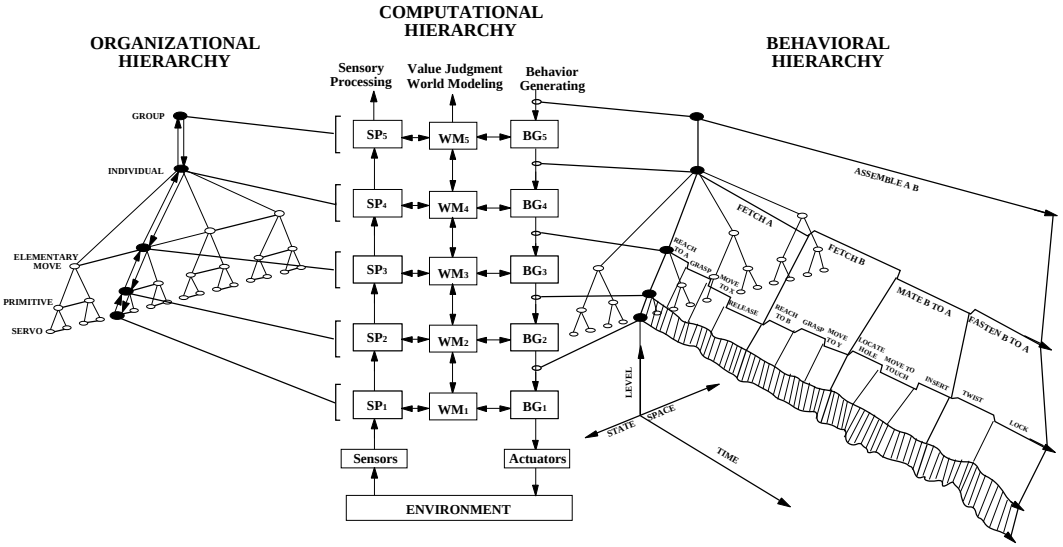


Figure 14. Three aspects of the ISAM reference model.

A third aspect is the evolution of behavior along the time line shown on the right of Figure 14. Commands and state-variables in messages flowing between BG, WM, SP, and VJ processes within and between units can be plotted versus time to reveal the past history and future plans of each unit. In this figure, messages between BG processes are shown. This view emphasizes the temporal nature of the planning, control, sensory processing, and world modeling processes in the ISAM architecture.

The Computational Hierarchy for an Intelligent Machine Tool

Figure 15 shows an example of a computational hierarchy for a machine tool with capabilities for in-process inspection.

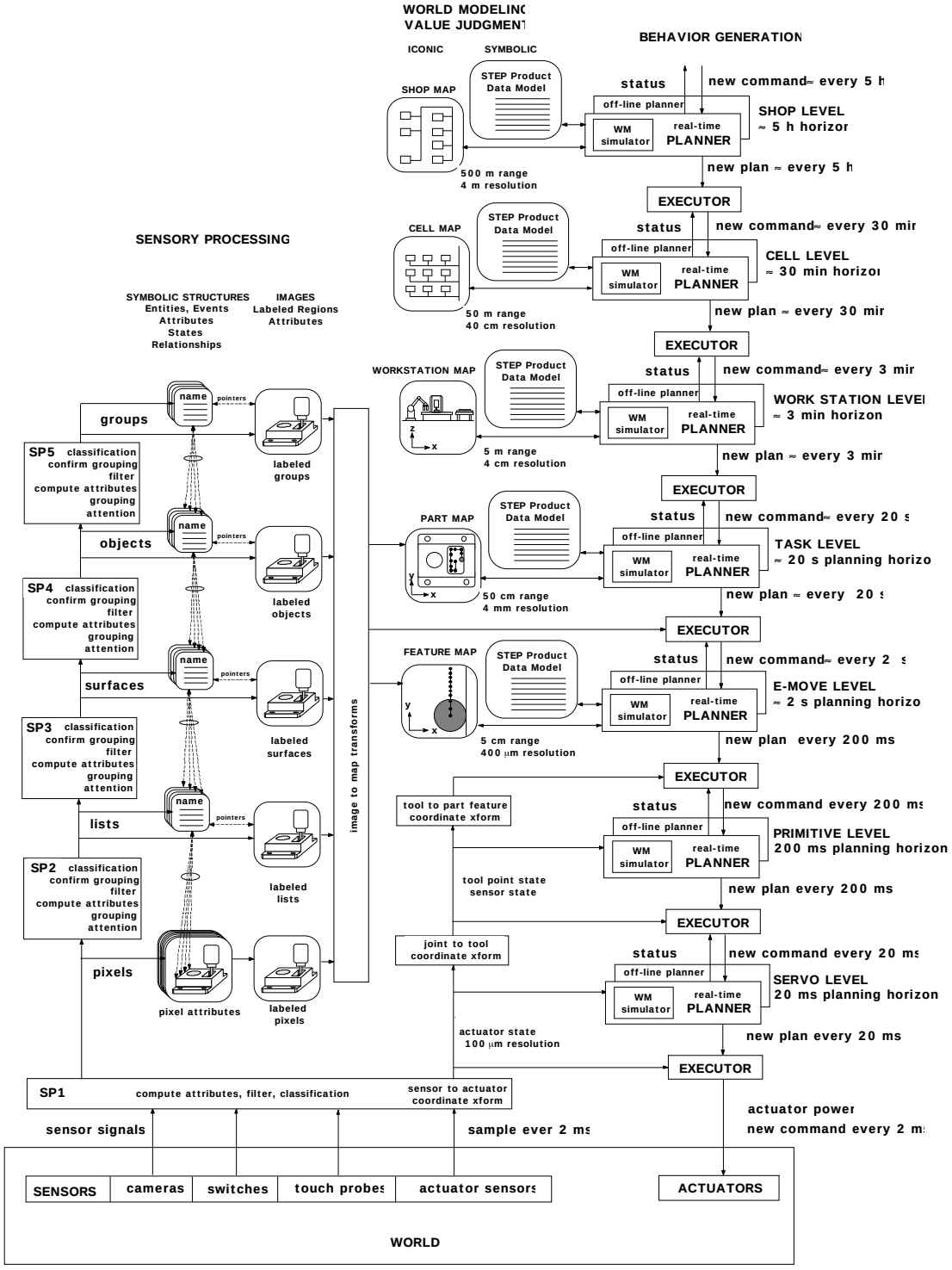


Figure 15. A computational hierarchy for a machine tool with capabilities for in-process inspection

An RCS Implementation of ISAM

The ISAM architecture can be implemented using the Real-time Control System (RCS) [23, 24, 25] developed at NIST and elsewhere over the past quarter century. The basic functional module of RCS consists of an augmented finite state automata, or state machine, that computes mathematical, logical, iconic, or symbolic operations on input vectors to produce output vectors [26]. Each input vector consists of an input command from a supervisor and/or an operator, feedback from the world model, and status from one or more subordinates. Each output vector consists of an output command to one or more subordinates and status to the supervisor and operator display as illustrated in Figure 16. The input/output relationships for a RCS_MODULE can be described as a many-to-one vector mapping from an input space to an output space. This is the type of functional mapping performed by a single neuron in the brain, or by a small population of neurons with only one or two synapses between input and output.

The elementary RCS_MODULE executes cyclically. During each execution cycle, the module reads from its input buffers, performs a pre-process, a decision process, a post-process, and writes to its output buffers. The pre-process translates the input vector into a format that can be accepted by the decision process. The post-process translates the decision process output into a format that can be written to the output buffers. The post-process may also run a procedure that is called by the decision process. Both pre- and post-processes may also perform various housekeeping functions, diagnostics, and performance measures. The execution cycle runs to completion each time it is triggered. A typical RCS_MODULE takes only a few _s to execute a cycle. The time interval between one execution trigger and the next depends on the sample rate required by the process being controlled. For example, servo control for a typical industrial robot requires a sample and control rate of around 1000 Hz. Servo control for a high precision machine tool may require rates up to 10,000 Hz. At higher levels in the control hierarchy, the RCS_MODULE may cycle less frequently and the execution trigger may be event driven.

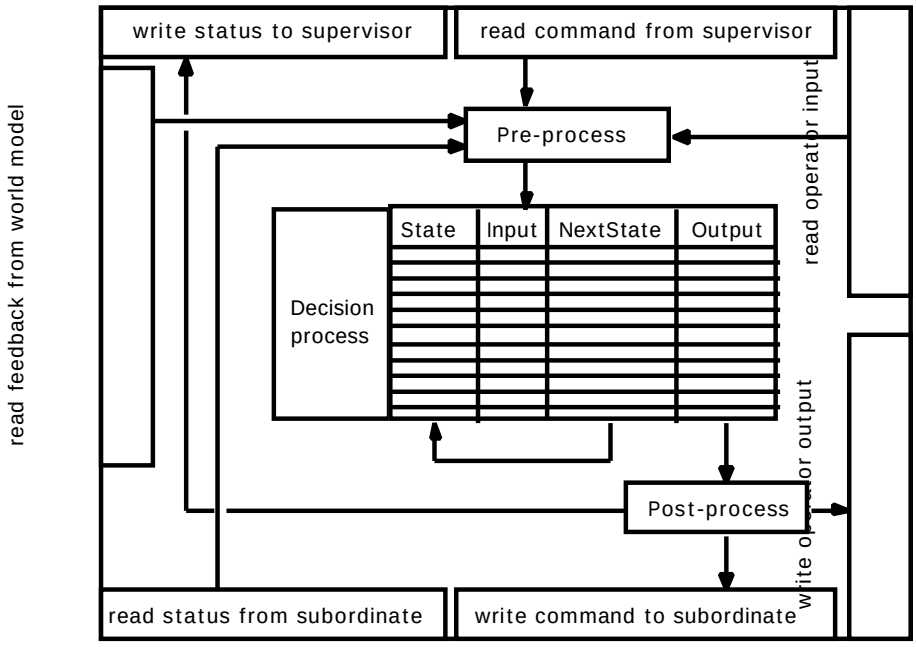


Figure 16 An elementary RCS_MODULE. Each module has a set of input and output buffers. Input consists of a command from a supervisor and/or an operator, feedback from the world model, and status from one or more subordinates. Output consists of commands to one or more subordinates and status to a supervisor and operator display. The decision process illustrated here is a state-table.

The elementary RCS_MODULE can be arranged in a hierarchical architecture in which commands and feedback into each module are used to decompose input commands into strings of commands to the next lower level. The RCS_MODULE can also be used as a generic functional module with a relationship between input and output that implements a wide variety of processes other than hierarchical command decomposition. Many different mathematical or logical functions can be called from within the pre-processing, decision-processing, or post-processing stages to implement planning, world model updating, simulation, prediction, windowing or grouping of sensory data, computation of attributes, filtering, recognition, and computation of value judgment costs and benefits. RCS_MODULEs can be arranged in a client/server relationship such that queries to world model modules are serviced and replies are generated on demand; or they can be used to sample and process sensory input, compare observations with world model predictions, compute states and attributes, detect events, recognize objects and situations, and establish and maintain correspondence between the system's world model and the real world. The elementary RCS_MODULE is a standardized functional module that can be interconnected and nested to perform any kind of functional operation on an input vector to produce an output vector. The RCS_MODULE thus provides a perfectly general tool for designing an intelligent machine system.

NML Communications

Communication between modules of an ISAM architecture based on RCS can be implemented by a real-time communication process called the Neutral Message Language (NML) [27]. NML consists of a set of communications channels and a vocabulary of messages that can be transmitted over those channels. NML includes a process that moves information from output buffers to the appropriate input buffers. At the end of each computational cycle, a RCS module calls NML to move the information stored in its output buffers into an external NML buffer, or mailbox. Once this is done, any RCS_MODULE can call NML to move the information from the NML mailbox into its input buffers. This is illustrated in Figure 17. Each pathway from a RCS_MODULE output buffer to a RCS_MODULE input buffer is called an NML channel.

The mailboxes defined by NML contain information. The entire set of mailboxes contain all of the information that is currently flowing between RCS_MODULES. The NML mailboxes are a subset of the Knowledge Database. RCS can thus be described either by a functional block diagram where the functional modules are drawn as boxes with arrows indicating communication pathways, or by a data flow diagram where data flowing between functions is drawn as circles with arrows indicating the functional transformations between input and output data sets.

The NML communications channels are defined by a NML configuration file. A configuration file consists of a set of lines containing ASCII characters. There is a line in the configuration file for every NML buffer, a line for every process that writes to that buffer, and a line for every process that reads from that buffer.

Buffer lines specify the buffer name, the buffer type, the host computer in which the buffer resides, the size of the buffer, and whether the data stored in the buffer is being communicated between processors with incompatible data formats and therefore needs to be converted into a neutral format. Buffer lines also specify some additional information that we need not discuss here. A detailed discussion of how to write NML messages and configuration files is contained at *http://www.isd.mel.nist.gov/projects/rcs_lib*

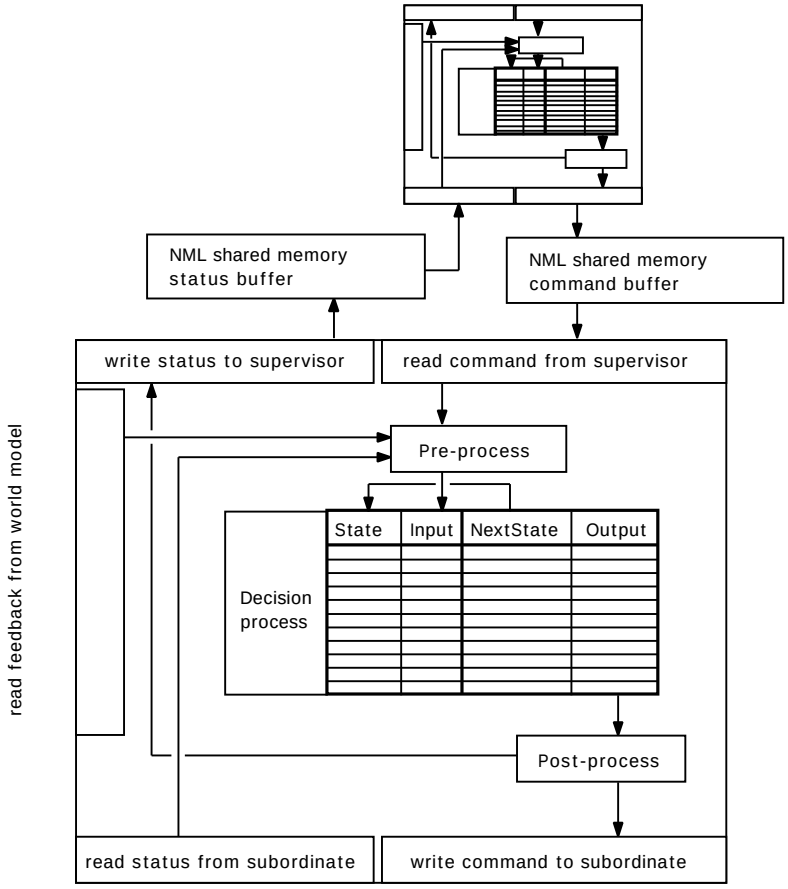


Figure 17. A pair of NML communications channels between two RCS_MODULEs.. In this diagram, the NML mailboxes are implemented as shared memory buffers.

NML supports a variety of communications protocols including common memory, point-to-point messaging, queuing or overwrite message delivery, and blocking or non-blocking read mechanisms. A typical NML message consists of a unique identifier, the size, and the message body. The message body can contain a data structure such as a C struct or a C++ class.

NML provides a mechanism for handling many different kinds of messages. It can support communicating between local modules that reside on the same computer or remote modules that reside on different computers. NML can be configured to provide communication services between modules on a single computer or multiple computers, on singleboard or multiple boards.

NML is very portable and has been implemented on a wide variety of operating systems including SunOS, Unix, Linux, VxWorks, and Windows NT. It can handle messages to

be transmitted between different types of computers across a backplane, via a local area network, or across the Internet.

Summary and Conclusions

The ISAM model addresses the application of intelligent systems to the manufacturing enterprise at three degrees of abstraction:

1. At a high degree of abstraction, ISAM provides a conceptual framework for the entire manufacturing enterprise, including machines, processes, tools, facilities, computers, software, and human beings operating over time on materials to produce products. The ISAM conceptual framework can be applied to the entire range of manufacturing operations, from those that take place over time periods of microseconds and distances of microns to those that take place over time periods of years and distances of many kilometers. The ISAM model is intended to allow for the representation of activities that range from detailed dynamic analysis of a single actuator in a single machine to the combined activity of thousands of machines and human beings in hundreds of plants comprising the operations of a multinational corporation.

2. At a middle degree of abstraction, ISAM provides a reference model architecture for the design of manufacturing systems and software. ISAM addresses the integration of perception, cognition, knowledge representation, simulation, reasoning, and planning with reactive control. The ISAM reference model architecture provides a sound theoretical foundation for:

    a. metrics, measures, and procedures that can quantitatively measure and evaluate the performance of intelligent control systems.

    b. interface standards that can support dynamic real-time interactions between production goals and sensed conditions in the industrial environment.

    c. open architecture standards that can enable manufacturing system software from a variety of vendors to work together without extensive effort required for software integration.

3. At a low degree of abstraction, ISAM provides engineering guidelines for implementing specific instances of manufacturing systems, including controllers for machine tools, robots, inspection machines, and material handling systems organized into workstations, cells, shops, and factories. An intelligent manufacturing system is defined in terms of a hierarchy of nodes that employ elementary processes of Behavior Generation, World Modeling, Sensory Processing, and Value Judgment. Functionality of each of these processes is described, a Knowledge Database is defined, and a communications system that can be used to integrate all these systems together is provided. These engineering guidelines suggest how the elemental processes can be implemented using current software engineering tools, languages, and practices.

Publications that describe a number of applications that have used the ISAM reference model are contained in the following annotated bibliography.

Enhanced Machine Controller - EMC

The Enhanced Machine Controller (EMC) program is a NIST effort to develop and validate a specification for interfaces to open architecture controllers. The EMC is part of a broader industry group, Open Modular Architecture Controller. The purpose of the group is to establish a specific set of Application Programming Interfaces (APIs) to be used by vendors to sell controller products and services to the aerospace and automotive industry. EMC participation is focused on validating the APIs and developing measures of conformance.
Several controllers have been installed on testbeds for the purpose of validating that the APIs work in real-world applications. A second purpose for these testbeds is to further the NIST Real-time Control System (RCS) methodology. For further information, see:

http://www.isd.mel.nist.gov/projects/emc/emc.html

Feature-based Inspection and Control System - FBICS

The FBICS system provides the capability to derive required data for machining and inspection of a part from a feature-based representation of the part. For information on planning and control with FBICS see:

Kramer, T. R., Huang, H., Messina, E., Proctor, F. M., Scott, H., "A Feature-based Inspection and Machining System;" Journal of Computer Aided Design, Vol. 33, Issue 9, August 2001.

Kramer, T. R., Proctor, F.M., "Feature-Based Control of a Machining Center," NISTIR 5926, National Institute of Standards and Technology, Gaithersburg, MD, December 1997.

Inspection Workstation - IWS

The Inspection Workstation (IWS) serves as a development testbed for many of the concepts, methodologies, and technical approaches described by ISAM. The facility consists of a coordinate measuring machine (CMM), vision system, measurement probes, simulation components, and numerous control nodes operating in a highly distributed fashion in a number of computing systems. Several IWS research activities are further described in the following:

Messina, E.R., Huang, H.M., Scott, H.A., "An Open Architecture Inspection System,"

Proceedings of the Japan-USA Symposium on Flexible Automation, Ann Arbor, MI, July 23-26, 2000.

Horst, J.A., "Real-Time and Object-Oriented Issues for an Inspection Workstation Application," Proceedings of the 5th International Workshop on Object-Oriented Real-Time Dependable Systems, Monterey, CA, November 18 - 20, 1999.

Messina, E., Horst, J.A., Kramer, Tom, Huang, H.M., Tsai, T.M., Amatucci, E., "A Knowledge-Based Inspection Workstation," Proceedings of Intelligence in Automation & Robotics Symp., part of IEEE International Conference on Intelligence, Information & Systems, Bethesda, MD, Oct 31-Nov 3, 1999.

Scott, H.A., "The Inspection Workstation-based Testbed Application for the Intelligent Systems Architecture for Manufacturing," Proceedings of the International Conference on Intelligent Systems: A Semiotic Perspective, Gaithersburg, MD, October 20-23, 1996, (1997).

Huang, H.M, "Intelligent Manufacturing System Control: Reference Model and Initial Implementation," Proceedings of the 35th IEEE Conference on Decision and Control, Kobe, Japan, December 11-13, 1996.

Generic Shell

A description of a template-based approach for constructing ISAM systems is described in the following. Several components of the Inspection Workstation control system follow this technique.

Huang, H.M., Albus, J.S., Shackleford, W., Scott, H.A., Kramer, Tom, Messina, E., Proctor, F.M., "An Architecture and Tool for Large-scale System Control with a Manufacturing System Application," Proceedings of the 4th International Software Architecture Workshop in conjunction with the 22nd International Conference on Software Engineering, Limerick, Ireland, June 4-5, 2000.

Real-time Control System (RCS) Libraries

The Real-Time Control Systems library is an archive of free C++ and Java code, scripts, tools, makefiles, and documentation developed to aid programmers of software to be used in real-time control systems, especially those using the ISAM/RCS Reference Model. See:

http://www.isd.mel.nist.gov/projects/rcslib/index.html

Control Shell Implementation

A description of a graphically assisted, commercially available, component-based approach for constructing ISAM systems is described in the following. Several components of the Inspection Workstation control system follow this technique.

Horst, J.A., “Architecture, Design Methodology, and Component-Based Tools for a Real-Time Inspection System,” Proceedings of the 3rd IEEE International Symposium on Object Oriented Real-time Distributed Computing (ISORC 2000), Newport Beach, CA, March 15-17, 2000.

Real-time Control System (RCS) Handbook

A guide for developers of real-time controls systems software, based on a C++ implementation of the Real-Time Control Systems (RCS) methodology for distributed hierarchical control:

"Passino, K. M., Gazi, V., Moore, M. L., Shackleford, W. P., Proctor, F. M., Albus, J. S., The RCS Handbook: Tools for Real-Time Control Systems Software Development, John S. Wiley and Sons, New York, 2000.

Planning

An examination of planning in an RCS system may be found in:

Albus, J.S., Lacaze, A., Meystel, A., “Planning in the Hierarchy of NIST-RCS for Manufacturing,” Proceedings of the International Conference on Intelligent Systems: A Semiotic Perspective, Gaithersburg, MD, October 20-23, 1996.

Work continues in many areas, including machine tool accuracy enhancement, inspection machine automation and self calibration, feature-based process planning, open architecture controllers, knowledge-based manufacturing, and large scale hierarchical systems engineering. Related work is proceeding in parallel in the area of unmanned ground vehicles for military applications.

References

1. Initial Graphics Exchange Specification IGES 5.3
   www.ansi.org

2. ISO 10303-11:1994, Industrial automation systems and integration - Product data representation and exchange - Part 11: The EXPRESS Language Reference Manual, ISO, Geneva, Switzerland, 1994.

3. ISO 10303-21:1994, Industrial automation systems and integration - Product data representation and exchange - Part 21: Clear Text Encoding of the Exchange Structure, ISO, Geneva, Switzerland, 1994.

4. ISO 9506-1:2000 Industrial automation systems -- Manufacturing Message Specification -- Part 1: Service definition.

5. ISO 9506-2:2000 Industrial automation systems -- Manufacturing Message Specification -- Part 2: Protocol specification.

6. ISO/IEC 9506-5:1999 Industrial automation systems -- Manufacturing message specification -- Part 5: Companion standard for programmable controllers.

7. ISO/IEC 9506-6:1994 Industrial automation systems -- Manufacturing message specification -- Part 6: Companion Standard for Process Control.

8. The Common Object Request Broker: Architecture and Specification, Revision 2.0. Framingham, MA: Object Management Group, 1995.

9. Component Object Model (COM)
   www.microsoft.com/com

10. Open Modular Architecture Controls (OMAC)
    www.arcweb.com/omac

11. Metrology Automation Association (MAA)
    www.metrologyautomation.org

12. IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems.

13. Maes, P. (1994), "Agents that Reduce Work and Information Overload", Communications of the ACM, 37, #7, pp. 31-40.

14. Maes, P. (1990) "Situated Agents Can Have Goals," Robotics and Autonomous Systems, *6*, pp.49-70.

15. Lieberman, H., Nardi, B., and Wright, D. (1999), "Training Agents to Recognize Text by Example", Proceedings ACM Conference on Autonomous Agents, Seattle, Washington

16. McDermott, D. (1985) "Reasoning about plans," In Formal Theories of the Commonsense World, (Hobbs, J.R. and Moore, R.C., Eds.) Ablex.

17. Georgeff, M. (1984) "A Theory of Action for Multiagent Planning," Proc AAAI 84, Texas

18. Stefik, M. (1981) "Planning with Constraints," Artificial. Intelligence., 16, pp. 111-140

19. McDermott, D. (1982) "A Temporal Logic for Reasoning about Processes and Plans," Cognitive Science. *6*, pp. 101-155

20. Hopcroft, J. and Ullman, S. (1979) Introduction to Automata Theory: Languages and Computation, Addison-Wesley, Reading, MA

21. *Background Study: Requisite Elements, Rational, and Technology Overview for the Systems Integration for Manufacturing Applications (SIMA) Program*, NISTIR 5662, Editors: Edward J. Barkmeyer, Theodore H. Hopp, Gaylen R. Rinaudot, Contributors: Neil Christopher, Shaw Feng, Simon Frechette, Al Jones, Mark Luce, Kevin Lyons, Chuck McLean, Stephen A. Osella, Steven Ray, Bradford Smith, Evan Wallace, Peter Wilson, National Insitute of Standards and Technology, September 1995.

22. Schank, R. and Colby, K. Computer Models of Thought and Language, W.H. Freeman, San Francisco, 1973.

23. Albus, J., “The NIST Real-time Control System (RCS): An Application Survey,” Proceedings of the AAAI 1995 Spring symposium Series, Stanford University, Menlo Park, CA, March 27-29, 1995

24. Albus, J. and Meystel, A., “A Reference Model Architecture for Design and Implementation of Intelligent Control in Large and Complex Systems,” International Journal of Intelligent Control and Systems, Vol. 1, No. 1, pp. 15-30, 1994.

25. Albus, J.S., “The NIST Real-time Control System (RCS): An approach to Intelligent Systems Research,” Journal of Experimental and Theoretical Artificial Intelligence 9 pp. 157-174, 1997.

26. Barbera, A.J., Fitzgerald, M.L., Albus, J.S., Haynes, L.S., “RCS: The NBS Real-Time Control Systems,” Proceedings of the Robots 8 Conference and Exposition, Volume 2 - Future Considerations, Detroit, MI, June 4-7, 1984.

27. Shackleford, W., Proctor, F.M., Michaloski, J.L., “The Neutral Message Language: A Model and Method for Message Passing in Heterogeneous Environments,” Proceedings of the 2000 World Automation Conference, Maui, HI, June 11 - 16, 2000.