3

Model-Based Architecture Concepts for Autonomous Systems Design and Simulation

Bernard P. Zeigler and Sungdo Chi
AI-Simulation Group
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721

Abstract

This chapter presents a coherent methodology to integrate planning, operation, and diagnosis within autonomous systems. After briefly reviewing the functional aspects of high autonomy, we describe a systematic methodology to integrate these aspects within a model-based architecture. Such an architecture is based on suites of models developed to support the various functional aspects or tasks. A general approach to task-based model development is then summarized in a Hierarchical Encapsulation and Abstraction Principle (HEAP) and this principle is illustrated in the planning, operations and diagnosis task domains.

1. OVERVIEW

To cope with complex objectives, an autonomous system requires integration of symbolic and numeric data, qualitative and quantitative information, reasoning and computation, robustness and refinement, discrete event system specification and continuous system specification. In general, the AI approach is too qualitatively oriented to handle quantitative information very well. For example, classic AI planning approaches [1,2,3] do not consider the timing effects, which should be of primary concern in representing our dynamic world. On the other hand, control researchers have a fairly narrow view-point, so that they mainly focus on refinement rather than robustness of a system [4], and they usually consider only the normal operational aspects of a system [5,6]. However, autonomous systems have to deal with abnormal behavior of a system as well. Thus, it is crucial to have a strong formalism and an environment that allows coherent integration of symbolic and numeric information in a valid representation process to deal with our complex dynamic world.

1.1 Model-Base Autonomous System Architecture

Figure 1.1 presents a model-based autonomous system architecture to integrate decision, action, and prediction components. The architecture features a model base at the center of its planning, operation, diagnosis, and fault recovery strategies. In this way, it integrates AI symbolic models and control-theoretic dynamic models into a coherent system.

Approaches to design various autonomous component models for planning, operation, and diagnosis have previously been developed in their respective research fields so that there are many overlaps as well as inconsistencies in assumptions. In an integrated system, such components cannot be considered independently. For example, planning requires execution, and diagnosis is activated when anomalies are detected during execution.

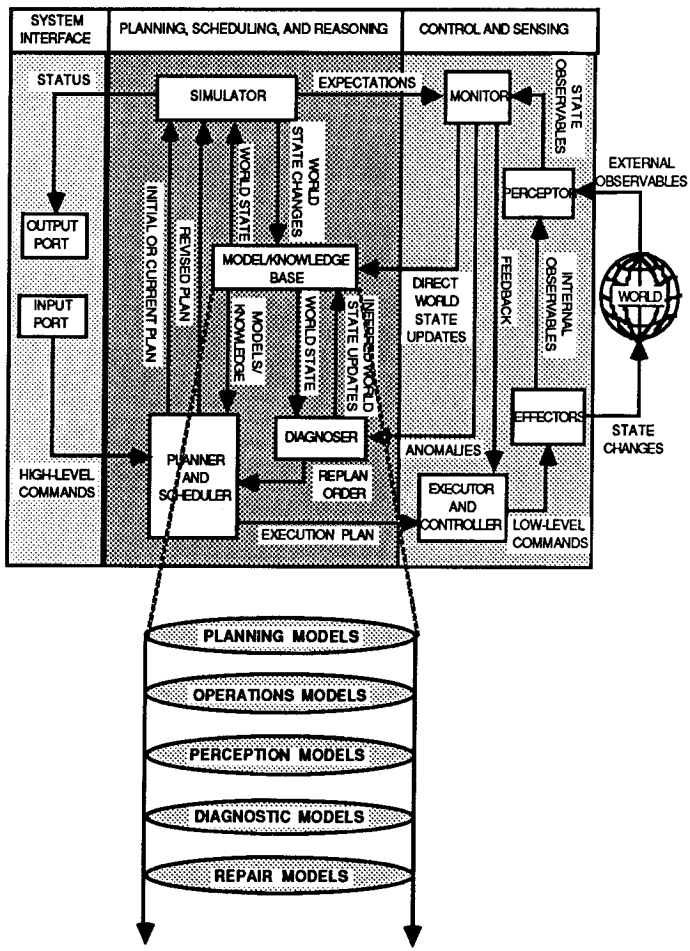


Figure 1.1. Autonomous System Architecture and its Model Base Class

Planning is defined as "Reasoning about how to achieve a given goal." It employs a suitable model to map a connecting sequence of states from an initial state to a goal state within a normal operation envelope. Once a plan is set up, it should be faithfully executed. "Execution with verification" maps from the input command and its expected normal responses to success/failure. As long as the execution is successful, it continues until the goal is achieved. However, if it fails, the diagnosis function will be activated. Diagnosis is defined as "discovering the cause of a failure." It maps back from one or several observed symptoms to one or several plausible anomalies that may have caused the observed symptom(s). Having identified the causes, the autonomous system should be able to recover, i.e., plan a path from the faulty state of the real system to a normal state on the original plan.

1.2 Model-based Planning

There are two major approaches to task planning: one considers planning as searching and the other considers planning as a representation problem. The former deals with the initial planning problem where no prior experience is employed. In contrast, the latter, so-called case-based planning, views planning as remembering, i.e., retrieving and modifying existing plans for new problems [7].

In our model-based approach, planning is achieved by pruning a System Entity Structure (SES) to select Pruned Entity Structures (PES) from alternatives [8]. Although space limitations preclude a detailed review of the SES concepts and the associated simulation environment, we present an example in the Appendix. Further detail is available in [9]. The PESs are in turn transformed into simulation model structures for execution. The non-experienced initial planning, which means the pruning of SES alternatives, can be achieved by using a rule-based approach. Every action (or state) node has several rules associated with system constraints, pre-conditions, and post-conditions. The resultant PES is saved with an index into an Entity Structure Base (ENBASE) for reuse. In contrast with non-experienced planning, the experienced planning is done by retrieving PESs from the ENBASE. The planner first retrieves a plan that might be used to achieve a given goal, or generates a new trial plan from partial plans if no existing plan is suitable. This candidate plan is then projected forward via simulation by attaching component models in a model base (MBASE), where low level planning is embedded.

Once an execution model is synthesized, a lower level planner produces a goal table that is a list of 4-tuples: state, goal, command, and time-to-reach-goal. From these, a time optimal path from an initial state to a goal state is readily derived. Since discrete event models embody timing it is natural to base optimal sequencing on predicted execution time. The planner works by developing paths backward from the goal until the given initial states (possible starting states of the given system) are reached [8,9].

1.3 Model-based Operation

The event-based control paradigm realizes intelligent control by employing a discrete eventistic form of control logic represented by the DEVS formalism [10,11]. In this control paradigm, the controller expects to receive confirming sensor responses to its control commands within defined time windows determined by its model of the

system under control. An essential advantage of the event-based operation is that the error messages it issues can bear important information for diagnostic purposes.

The operational model used by event-based operation has a state transition table that is abstracted from a more detailed model that represents both normal and abnormal behavior. The state transition table keeps a knowledge of states, commands, next states, outputs, and time windows. The window associated with a state is determined by bracketing the time-advance values of all transitions associated with the corresponding states in the more detailed model. This divergence arises due to variations in parameters and initial states considered to represent normal behavior.

The operator uses the goal-table from the planner and the state-table of its operational model to issue commands to the controlled device. When proper response signals are received the operator causes the model to advance to the next state corresponding to the one in which the device is supposed to be. The operator ceases interacting with the device as soon as any discrepancy, such as a too-early or too-late sensor response, occurs and calls on an associated fault diagnoser.

1.4 Model-based Diagnosis

A model-based diagnoser assumes that its model of the system is correct, and looks for discrepancies between the behavior of its model and the behavior of the real system. If such a discrepancy has been detected, it assumes that the system has undergone some change which is considered a fault. In our approach, the diagnoser will make use of a fault model to match the observed anomalous behavior of the system. Diagnosis in the model-based architecture is performed by local and global diagnosers, to find single or multiple faults using knowledge of structure and behavior.

By local diagnosis we mean the diagnostic description of a component model: the local diagnoser looks for a fault that might have occurred within the currently activated model unit. Once the controller has detected a sensor response discrepancy, the local diagnoser is activated. Data associated with the discrepancy, such as the state in which it occurred, and its timing, are also passed on to the local diagnoser. From such data, as well as information it can gather from auxiliary sensors, the local diagnoser tries to discover the fault that occurred.

If the local diagnosis fails, the global diagnoser is activated to analyze the enclosing coupled model . The global diagnostic model is a cause-effect table obtained by symbolic simulation to generate all fault-injected trajectories that reach states exhibiting the detected symptom. Faults injected in such trajectories represent candidate diagnoses [12,13].

2. ENDOMORPHIC SYSTEM CONCEPT

Endomorphism refers to the existence of a homomorphism from an object to a sub-object within it, the part (sub-object) then being a model of the whole [9]. As illustrated in Figure 2.1, in order to control an object, a high autonomy system needs a corresponding model of the object to determine the particular action to take. The internal model used by the system and its world base model are related by abstraction, i.e., some form of homomorphic (i.e., endomorphic) relation. The inference engine asks its internal model for the necessary information for interacting with the real world object. By "world base model" we mean the most comprehensive model of

the world available to the system whether it exists as a single object or as a family of partial models in the model base.

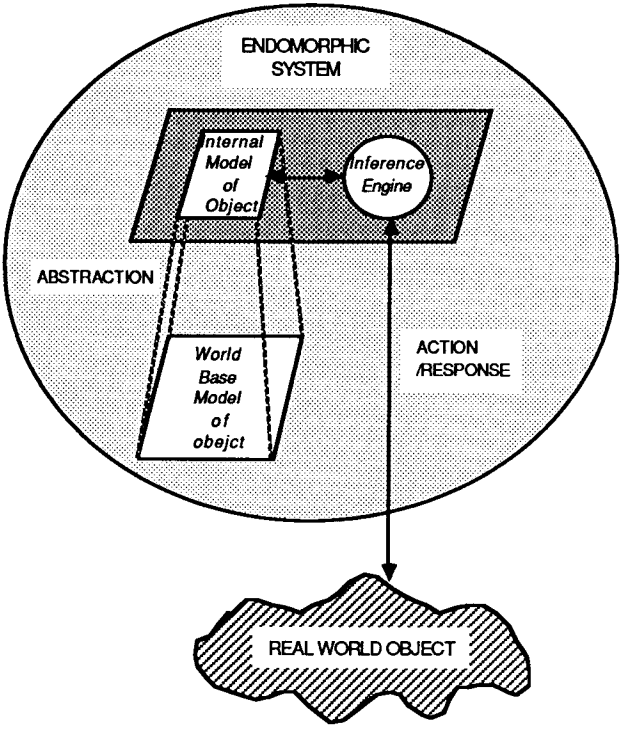


Figure 2.1. Endomorphic System Concept

3. ENGINE-BASED DESIGN METHODOLOGIES

Typical expert systems comprise a domain-independent inference engine and a domain-dependent knowledge base. The inference engine examines the knowledge base and decides the order in which inferences are made. The engine-based modelling approach provides a clear separation between the domain-dependent model base and the domain-independent inference engine. It facilitates the automatic generation of a model base using endomorphisms. Figure 3.1 shows the engine-based modelling concept and examples of autonomous system components realized using the concept.

4. HIERARCHICAL DEVELOPMENT OF INTELLIGENT UNITS

The autonomous system components described above have to be coupled within a unit in order to interact with each other. We use the term "intelligent unit" to denote the smallest unit that encapsulates all engine-based components as depicted in Figure 4.1.

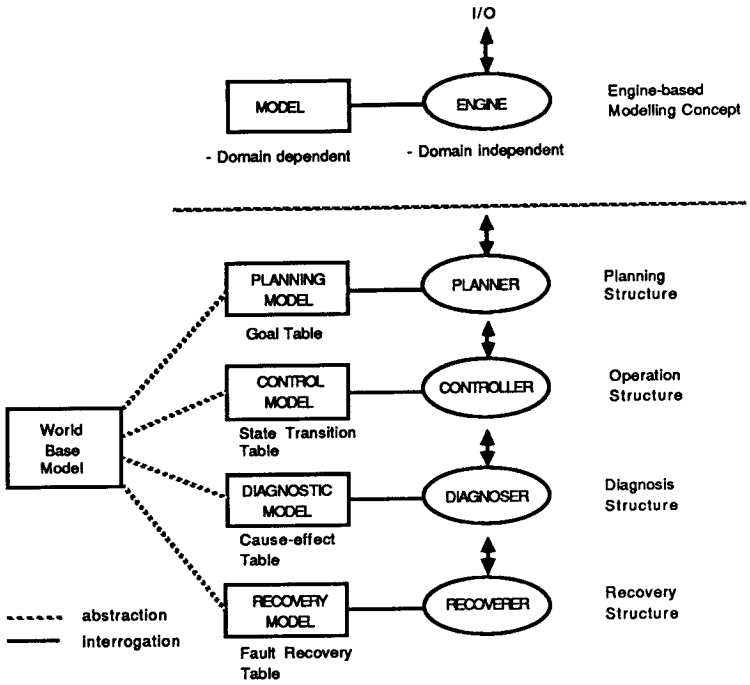


Figure 3.1. Engine-based Modelling Concept for Endomorphic System Design

To cope with complex problems, an autonomous system requires multiple intelligent units coupled in a hierarchical fashion. Models in the hierarchy must have valid abstraction relations to each other. Figure 4.1 illustrates an autonomous system development based on hierarchical abstraction and integration. Intelligent units at the leaf nodes of the execution structure employ internal models directly abstracted from the world base model. Units at higher levels employ internal models that are abstractions of coupled models composed of immediately inferior internal models. Model-base development will be further considered after the following framework for autonomous system generation has been presented.

5. METHODOLOGY FOR AUTONOMOUS SYSTEM GENERATION

The overall methodology for autonomous system generation in a model-based simulation environment is shown in Figure 5.1.

The task formulation module receives an objective. It then retrieves an SES from the ENBASE and generates a plan structure by using the goal sub-SES of the SES (for initial planning) or partitioned PESs (for experience-based planning). A simulation structure is obtained by synthesizing the models in MBASE, where models can exist in advance or be generated automatically.

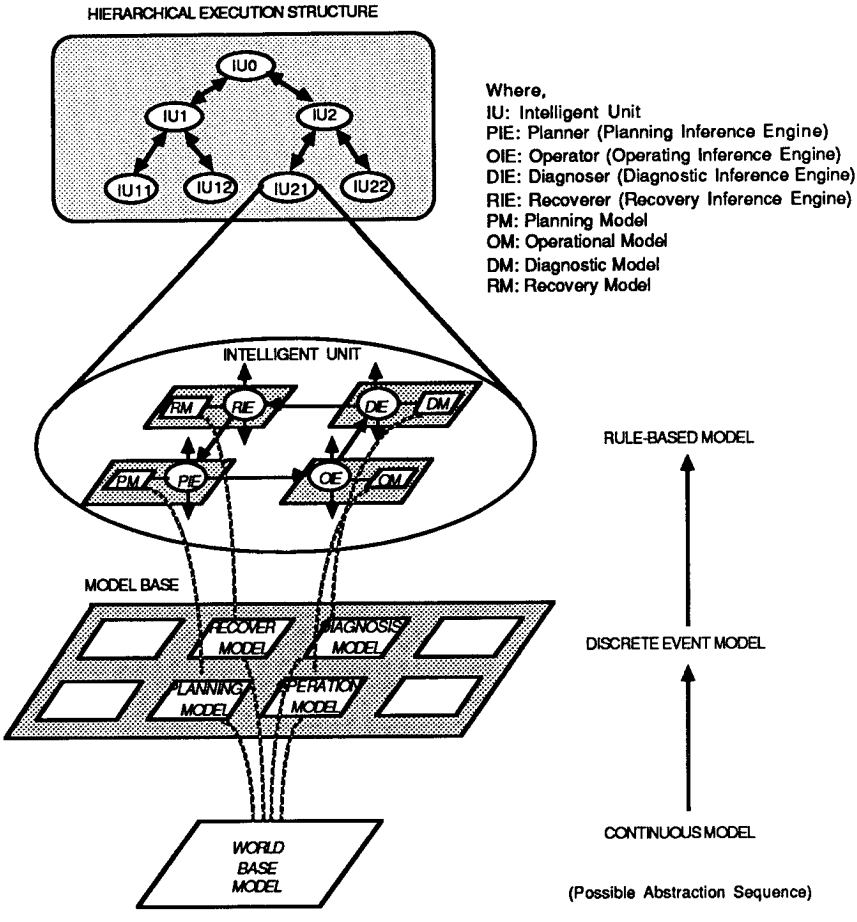


Figure 4.1. Hierarchical Abstraction and Integration of High Autonomy System

The initial environment for generating an autonomous system can be obtained using the layered development concept of DEVS-Scheme [9]. The first layer is the Lisp-based, object-oriented programming system that provides the foundation on which the system is built. The next layer is the SES/MB system where the behavioral and structural models can be built and saved in the MBASE and ENBASE, respectively. Models in MBASE are base models as well as internal models abstracted from base models.

5.1 Phase I: Plan generation

Once the basic environment is built, the next phase is the planning structure generation. When receiving a goal command, the task hierarchy is generated in a top-down fashion (task decomposition) as shown in Figure 5.2(a).

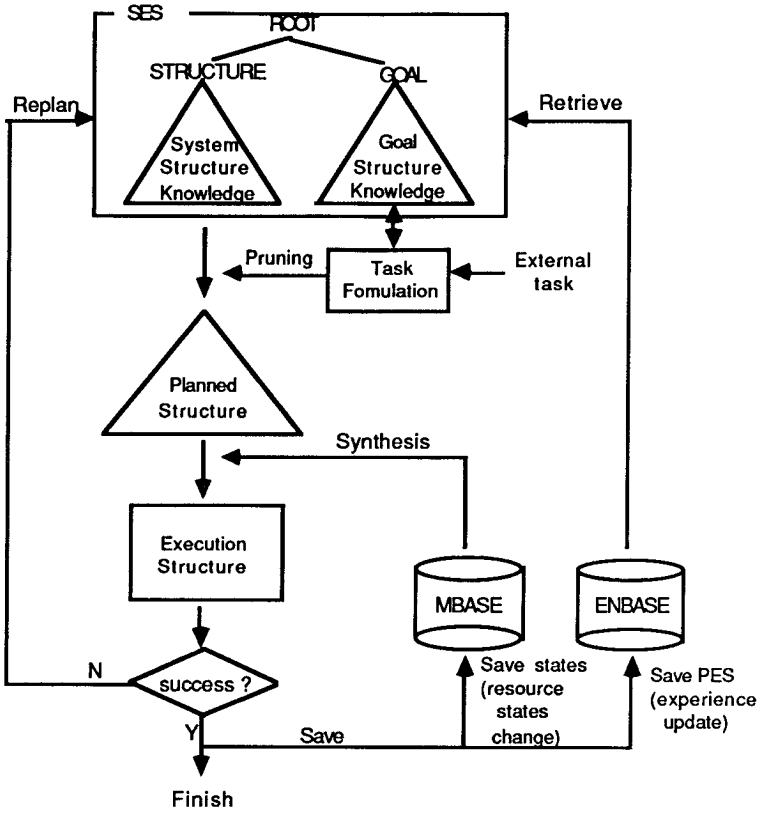


Figure 5.1. Autonomous System Generation Methodology

5.2 Phase II: Model construction

The next phase is the model base construction illustrated in Figure 5.2(b), where the necessary models can be retrieved from MBASE or automatically generated from the lower level models. This multi-layered hierarchical model generation and abstraction can be done in a bottom-up fashion. The resultant structure represents the domain dependent knowledge base structure.

5.3 Phase III: Engine attachment/integration

By attaching domain independent engines such as a planner, an operator, a diagnoser, and a recoverer which are able to interrogate corresponding models, we have a multi-

agent structure. Now by coupling those agents, we can obtain the autonomous system architecture shown in Figure 5.3(c).

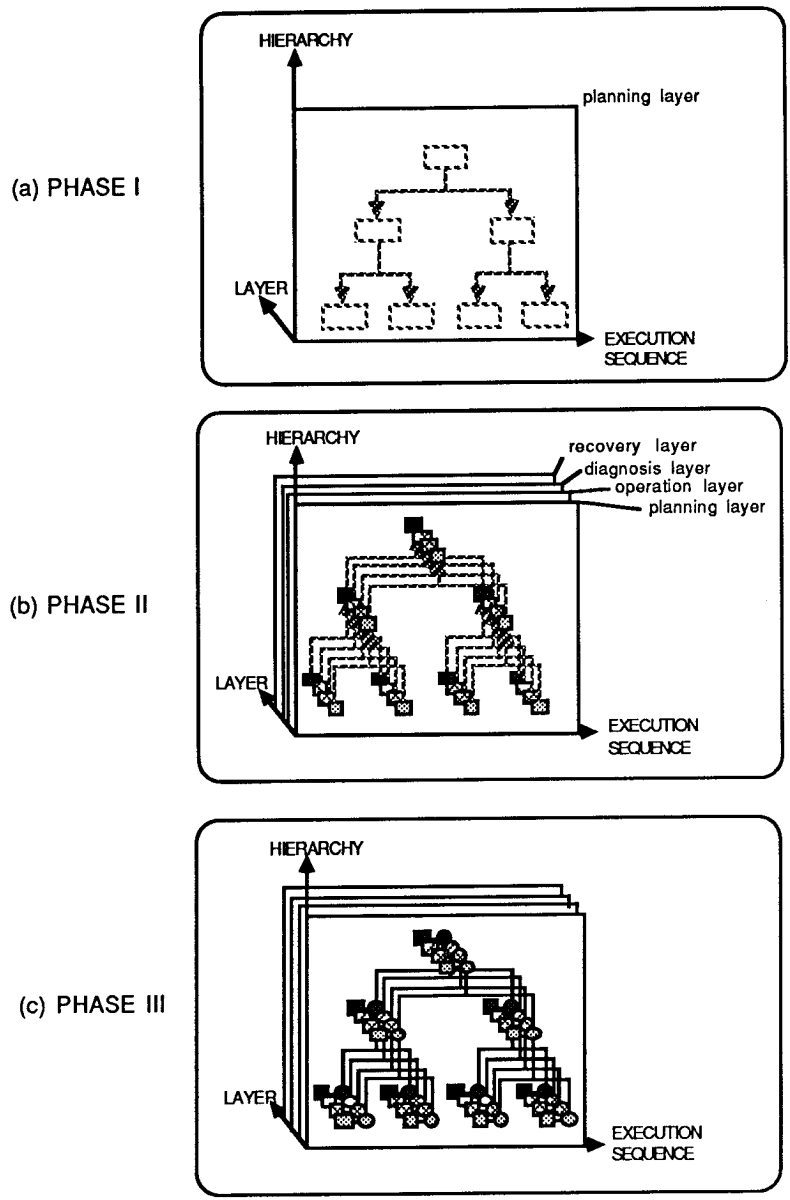


Figure 5.2. Autonomous System Generation Procedure

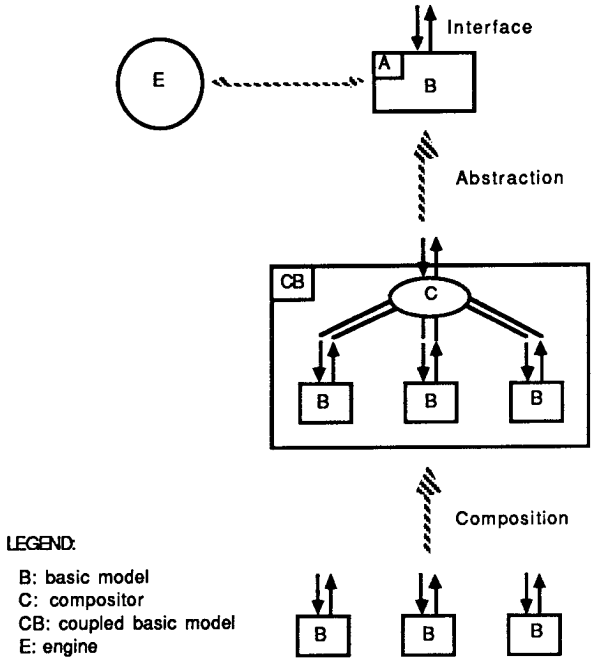
6. TASK-BASED MODEL DEVELOPMENT

So far we have given a general outline of the model-based approach to high autonomy system design. Crucial to the success of this approach is the development of models and associated engines to support the various tasks. This section further elaborates on the endomorphic system approach that has emerged in our research. Due to space limitations we cannot illustrate with actual examples but theses are available in great detail in two doctoral theses relating to our application to space laboratory automation [14,15].

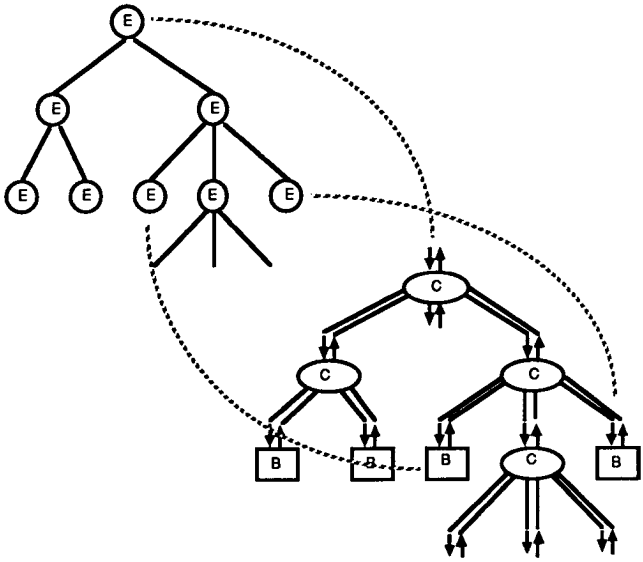
The general approach to task-based model development is summarized in the *Hierarchical Encapsulation and Abstraction Principle* (HEAP) illustrated in Figure 6.1. We start with an interface specification (illustrated by a pair of arrows in the figure). This is a set of input and output port types that we wish all our models to have for a given task. A Basic Model, B, is a model that has such an interface and therefore can be coupled to other Basic Models of the same class. Such a Basic Model can also be interrogated by a suitable Engine, E, to execute the task. However, the engine-model interaction is mediated by a different interface, that may, or may not, include elements of the model-model interface (Figure 6.1(a)).

The two sub-classes of Basic Model are Atomic and Coupled. Hierarchical construction is based on a modular synthesis process in which several basic models are coupled together through a compositor, C, such that the result has the required interface specification (Figure 6.1(b)). This Coupled Basic Model is a valid Basic Model, i.e., it can be coupled to an Engine of type E, and with other Basic Models, in the same manner as any Atomic Basic Model. A class of basic models is said to be *closed under coupling* if any of its members can be coupled together to form coupled basic models with the same interface characteristics. A coupled model may be abstracted to produce an Abstraction that has the the same input-output port types as its components. There is no requirement that the information flowing in such ports be of the same level of resolution as in the original. Indeed, a proper abstraction is obtained when this information is expressed using descriptors that are at once more coarse and more extended in space and/or time than provided by those of its components. An example will make this clearer in a moment.

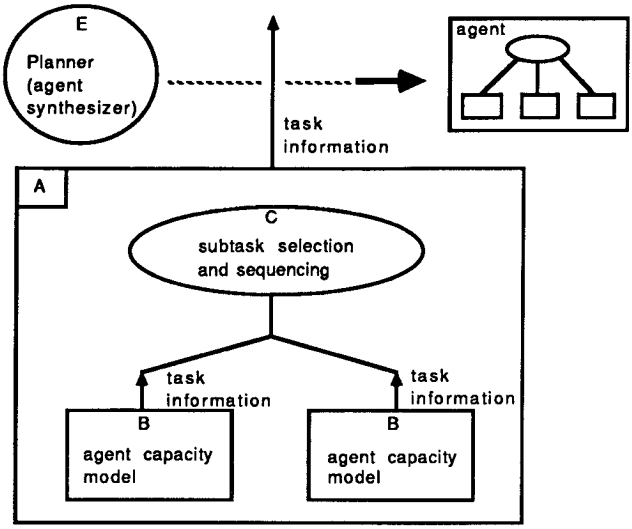
As shown in Figure 6.2, hierarchical construction is possible for classes that have the closure under coupling property. This is so since any Basic Model component of a Coupled Model can itself take the form of a Coupled Model. This leads to a recursive expansion that can be terminated by selecting Atomic Models. Paralleling the hierarchical model structure is a hierarchy in which Engines are attached, one-to-one, to the Basic Models. Such engines may also be linked to each other resulting in a hierarchy of control.



(a). The Hierarchical Encapsulation and Abstraction Principle



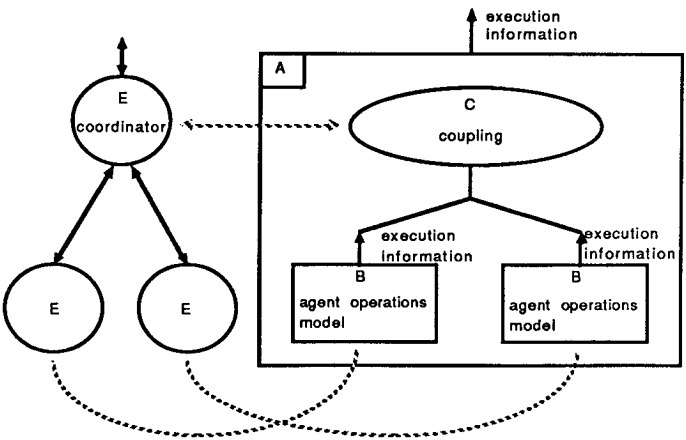
(b). Model Hierarchy and Associated Engine Hierarchy



task information:

inital world state set --> final world state set
•
•
inital world state set --> final world state set

Figure 6.2. HEAP Applied to Hierarchical Agent Synthesis (Planning)



execution information:

initial world state set --> final world state set
initialization/completion
time window

Figure 6.3. HEAP Applied to Hierarchical Agent Operation

6.1 HEAP Applied to Planning as Hierarchical Agent Synthesis

Figure 6.3 illustrates the application of the hierarchy encapsulation and abstraction principle to planning. Planning is viewed as *Agent Synthesis*, i.e., the output of a planning process is an agent capable of achieving a given goal. The Basic model, B is an *agent capability model* which provides a *task information* as its output interface to the next higher level. The *task information* is a set of *capabilities*, each one being a pair *(initial world state set, final world state set)*. This means that the agent is capable of transforming the world from a state in the initial set into a state in the final set.* The sets are described in a language suitable to the current level of abstraction. For example, a robot may have the capability to move a bottle from one table to another. At this level, the capability description might be: *(initial: on(bottle,table1), final: on(bottle, table2)*.

The Planning Engine is given a goal described as world state transformation in the same language as the task information offered by its attached agent capability model. Its job is to select a capability from the agent's information matching the goal specification.

A coupled model results from combining the capabilities of the component agent models in plausible compositions. The component capabilities may be described in a language that offers greater refinement in describing a world state. For example, the decomposition of moving a bottle from one table to another might involve lower level agents, such as grippers that work with the location of the bottle on a table-centered co-ordinate system, and movers that work with the position of the table in room co-ordinates, and so on. Since the descriptor *ontable* is at lower level of resolution then spatial co-ordinates the resulting coupled model is in fact an abstraction, A.

The choice of description for the ground Atomic level must match the level of task granularity that itself matches the capabilities of the "off-the-shelf" hardware available. Closure under coupling requires that A is again a task capability model that can be employed by a Planning Engine to satisfy a given goal. We employ an System Entity Structure (SES) to represent the hierarchy corresponding to Figure 6.2 that results from application of HEAP to planning. Pruning of the SES yields, in general, a hierarchically structured agent capable of satisfying the given goal. More details are available in [14].

6.2 HEAP Applied to Hierarchical Agent Operational Modelling

We use a second application of HEAP to model the operation of a hierarchical agent as shown in Figure 6.3. Here, the Basic Model, B, is an *agent operations model* which provides the following output as interface to the next higher level: a) the world state transformation is can perform (selected from its information in the planning stage), b) specification of the internal state it must be set into in order to carry out this transformation, c) specification of the output that will signal the completion of

---

* A capability is a generalization of the standard notions of pre-, and post-, conditions for actions. The initial world state set includes any preconditions needed for the agent to initiate is operation, but also includes other information such as initial values of world state variables that will be transformed in the action. The latter may be represented symbolically, rather than as fixed constants, so that effect of the action can be described in the final world state set, which is equivalent to a postcondition.

its task, and d) a time window in which the completion of the task should occur under **normal** operation. At the atomic level, we include "off-the-shelf" hardware in our set of agents, so that completion is signaled by designated sensor states. At supra-atomic levels, the agent must generate a message when it reaches a terminal state to indicate completion.

Coupling in this case is the sequencing of agents (chosen in planning). An Abstraction, A is the operations model whose output interface is computed as a result of this sequencing of the component operations models. More specifically, its world state transformation is the composition of those of its components in the specified order. Its initialization is that required to set each of its components into their specified initial states. Its completion signal is that of the final component. Its time window is obtained as an aggregation of those of its components.

A Coordinator Engine interrogates its associated agent's operations model to control execution of the actual agent. Consider a hierarchical model and its associated control hierarchy of Coordinator Engines. At any supra-atomic level, upon receiving an initiation request from its superior, a Coordinator Engine requests that the Coordinator Engine of the first (in the planned sequence) sub-agent issue the initialization protocol to that sub-agent (obtained from the latter's model) to set it working. The higher level Coordinator Engine awaits the first agent's completion signal. If this signal arrives in the model-designated time window, the Coordinator Engine requests initiation of the second sub-agent, and so on. If at any point the expected completion signal does not occur within the expected time window, execution is stopped and control is transferred to the diagnosis sub system. If the completion signal of the last sub-agent in the sequence is received as expected, the Coordinator Engine passes this signal upwards to its superior, if one exists.More details are available in [15].

6.3 HEAP Applied to Hierachical Agent Diagnosis

The HEAP applied to the diagnostics sub-system takes a similar form (Figure 6.4). Here the Basic Model is a model of **abnormal** agent behavior (in contrast to the operations model which is based on normal behavior). Each such model contains two kinds of subcomponents: generators of *breakdown* events that can occur and with each such event, the *effects* that it causes. The hierarchy of such models parallels that of the hierarchy of operations model. That is, there is a breakdown/effects model corresponding to each normal operations model that tells how the behavior of the latter would change under each of the breakdown events. The coupling of such breakdown/effects models propagates the effects of breakdowns in one components to downstream components (feedback of effects is **not** prohibited in this analysis). The abstraction, A, must represent such overall breakdown-to-effect mappings in the more abstract language of the next higher level. How to perform such abstraction is a focus of our current research.

A Diagnoser Engine interrogates a breakdown/effects model to discover the possible breakdowns that could result in a given set of symptoms. This assumes that a subset of effects can be detected by the agent's sensors. The Diagnoser Engine accepts as diagnostic hypotheses the set of breakdowns which lead to detectable effects that match the given symptom set. At the atomic level, these hypotheses can be tested directly against the real world state. At the supra-atomic level, each hypothesis identifies a set of possible abnormal components and with each, the hypothetical breakdown event that occurred. To test such an hypothesis, the

Diagnoser Engine activates Engines associated with each of the proposed abnormal components to generate and test, in similar fashion, a set of lower level hypotheses consistent with the proposed breakdown. More details are available in [14].

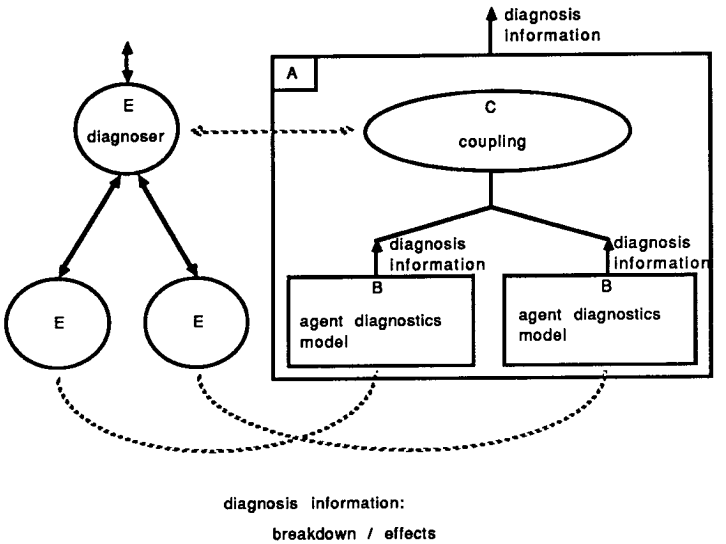


Figure 6.4. HEAP Applied to Hierarchical Agent Diagnosis

7. CONCLUSIONS

The main characteristics of the developed model-based architecture are as follows:

* The time-based formalism (DEVS) provides coherent integration between symbolic and numeric models.

* The SES/MB environment provides a hierarchical modularity, reusability, and testability.

* Model-based deep reasoning supports a powerful diagnostic capability.

* The endomorphism concept supports intelligent unit design and a consistent model base.

* The engine-based design builds a domain-independent architecture instantiated with compiled models for application under real time constraints.

* The intelligent unit encapsulates various autonomous components such as planning, operation, diagnosis, and recovery coherently to cope with complex problems.

* Model-based planning supports the reusability of experienced plan structures.

* The event-based control logic guarantees robustness, reduced sensor complexity, and increased diagnostic capability.

The framework has been implemented in a simulation study of space-borne laboratory automation as a proof of the concept. Current research seeks to apply the methodology to an(actual) experimental laboratory prototype of an oxygen generation plant that would eventually operate on Mars. Such work presents the challenge of transferring concepts that have been proved in a simulation environment to real world operation. Progress made in this research is described in [16].

APPENDIX: Space Laboratory Automation Example of System Entity Structuring

This example is based on a project to develop a technology that will allow carefully designed and specially constructed laboratory robots to perform many routine tasks in a space laboratory under remote supervision from the ground. Therefore the robots must be able to perform simple operations autonomously, and communicate with the ground only at the task level and above [17]. Such robots should be able to judge the adequacy of a proposed action plan on the basis of expectations of its effects on the laboratory, materials, instruments, etc. For this purpose, it is important that simulation models at various levels of granularity can be automatically generated at run time from a set of generic master models [4].

In designing the robot models, we assume that necessary mobility, manipulative and sensory capabilities exist so that we can focus on task-related cognitive requirements. Such capacities, the focus of much current robot research, are treated at a high level of abstraction obviating the need to solve current technological problems.

A.1 SES Representation of a Space Laboratory

The laboratory environment illustrated in Figure A.1 is realized on the basis of object-oriented and hierarchical models of laboratory components within DEVS-Scheme. Laboratory configurations will be determined by issuing commands that invoke a pruning operation of the entity structure knowledge representation. The laboratory model is designed to be as generic as possible. However, as stated earlier, the focus will be upon fluid handling in microgravity, which presents a variety of problems that are unique to space.

Figure A.2 illustrates the System Entity Structure (SES) for the laboratory environment. The Space Station Laboratory (SSL) entity is decomposed into its structure knowledge part and its experiment (goal) knowledge part; STRUCTURE and EXPERIMENT. The former relates to the execution structure which is concerned with how to construct the system, while the latter relates to the goal structure which is concerned with how to achieve a desired goal efficiently. After pruning the STRUCTURE, the entity structure is transformed into a hierarchical model containing controlled models at two levels. Each of the entities will have one or more classes of objects (models) expressed in the underlying simulation environment to realize it.

The EXPERIMENT contains a three level abstraction hierarchy distinguishing between high-level task planning models (HM), middle-level task planning models (MM), and low-level task planning models (LM) that are associated with so-called model plan units (MPUs): HMPU, MMPU, and LMPU in the STRUCTURE part, respectively, which are the cognitive control elements of the system. Each level is designed to represent experimental knowledge needed to decompose a given task into a composition of subtasks.

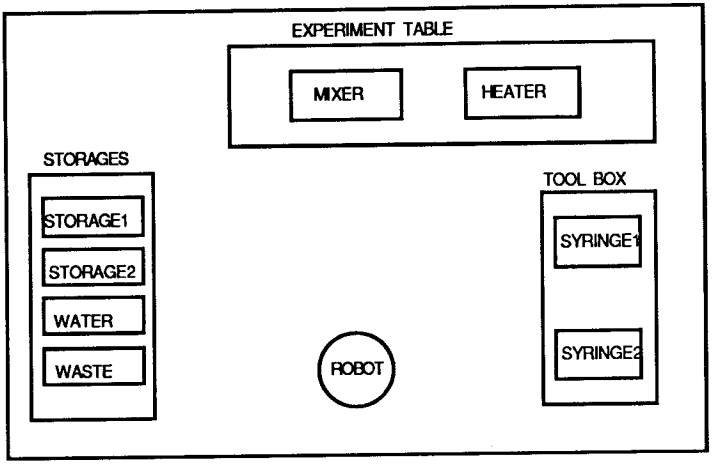


Figure A.1. Space-borne Chemical Laboratory View

The SPACE and OBJECTS decomposed from STRUCTURE are designed for the simulation-oriented knowledge representation (Figure A.2). The SPACE is basically a modelling artifact -- necessitated by the object-oriented, modular modelling paradigm -- to conveniently represent knowledge of where objects are and with whom they can communicate and interact.Motion and communication of ROBOTs are managed by the SPACE as shown in Figure A.3. When a ROBOT moves around, its MOTION system sends its new location and direction to the SPACE which keeps track of the ROBOT's positions and directions. When a ROBOT wishes to communicate with other ROBOTs, it sends message via its SENSE system to the SPACE which relays the message only to those ROBOTs within the range of the sender. The range is determined by the channel on which the message is sent. Thus different transmission media and sensory modalities can be modeled, such as light and vision, sound and hearing, pressure and touch, etc.

Since the SPACE has complete knowledge of locations, it can detect collisions between ROBOTs. SPACE is viewed as a kind of resource shared by its occupants so that collisions represent attempts to occupy the same space more than once at the same time. The SPACE can report such an event but do nothing to prevent it. However, the SPACE may be given greater intelligence to co-ordinate the ROBOTs, for example to prevent collisions, and to perform other space resource management tasks. In this case, it assumes the role of an "actual" system component, at least in part.

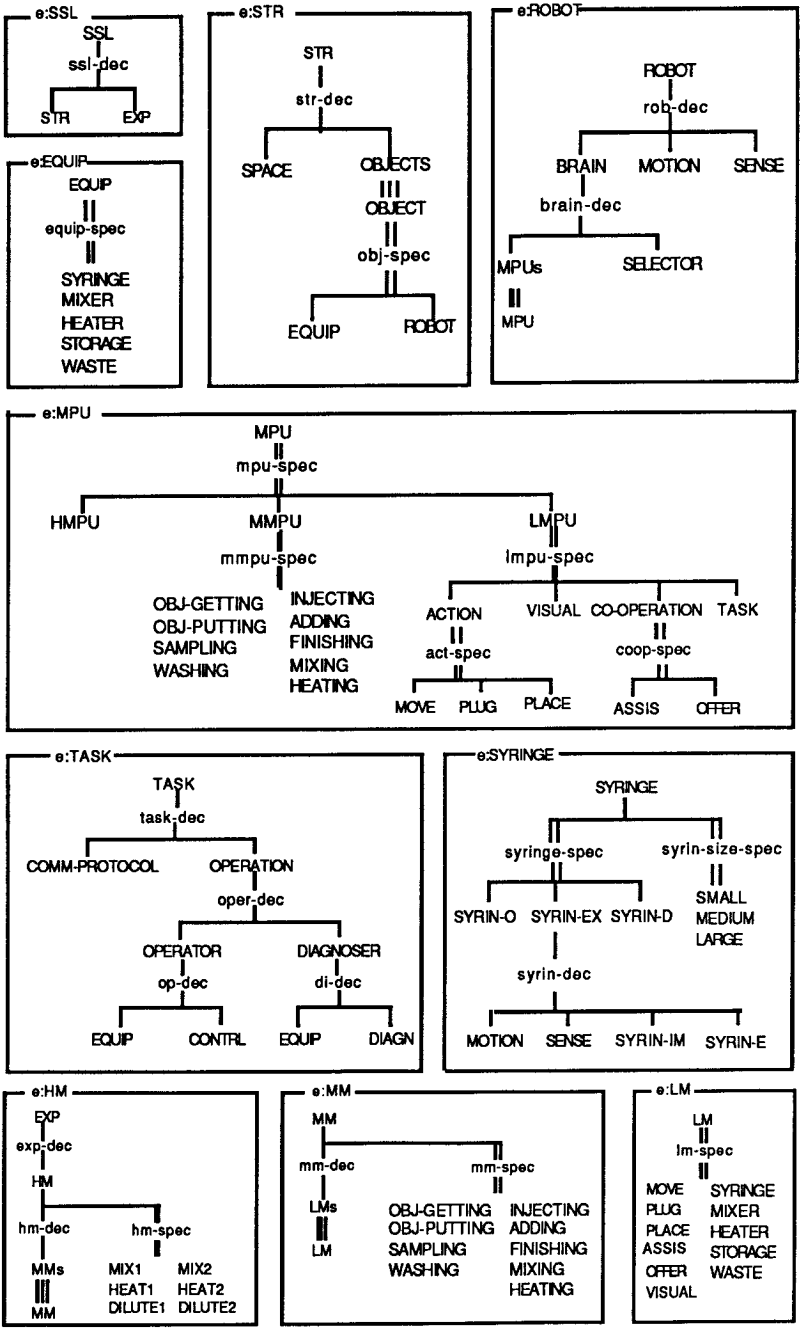


Figure A.2. Partitioned SES of Robot-Managed Fluid Handling Laboratory

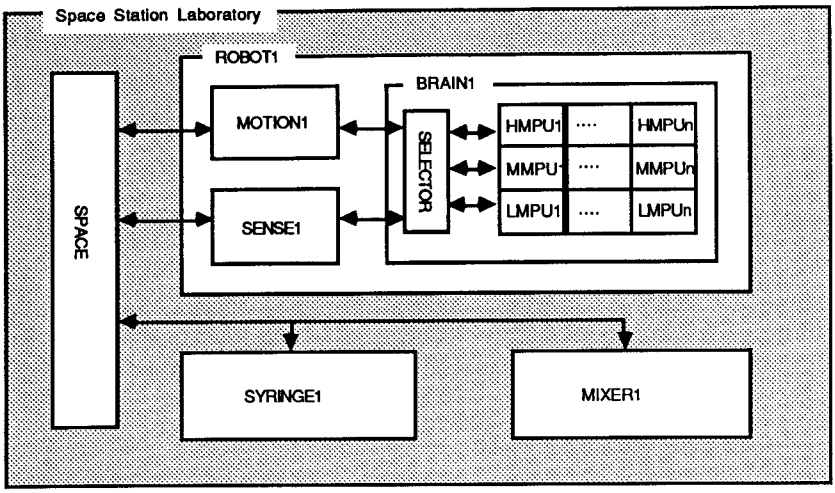


Figure A.3. Robot System Organization Example

Each OBJECT is specialized into ROBOT and EQUIP. Each ROBOT is decomposed further into MOTION, SENSE, and BRAIN. The EQUIP is a generic entity for the laboratory equipment that is modeled much in the same way as the ROBOT. However, EQUIP has no BRAIN, and its MOTION and SENSE subsystems are always passive. Note that OBJECTS are also defined as a multiple entity. Therefore, any number of OBJECTS may be generated, and with the pruning discussed earlier, we can have any desired number of ROBOTs and EQUIPs in the laboratory.

Each Robot's Cognition-system (BRAIN) is also implemented as a controlled model containing a SELECTOR as its controller, and several MPUs as its controllees.

The SELECTOR provides the communication channel between the SENSE subsystem and the MPUs. Essentially it is a bi-state (open/closed) device whose state is determined by the MPU responses. In its closed state, it passes on the incoming sensory inputs to the activated MPU. Upon completion of the activated plan or upon receiving a discrepancy alert, it switches to the open state in which MPUs may vie for activation.

The MPUs are specialized into HMPUs, MMPUs, and LMPUs, respectively, corresponding to the EXPERIMENT abstraction hierarchy previously mentioned.HMPU is a high level MPU that manages the actions of the middle level MMPUs, each of which performs the same function with respect to the low level LMPUs. The latter employs event-based control logic to interact with the real world. These three types of MPUs are represented at the same level in the entity structure but conceptually they reflect the hierarchical decomposition structure of the EXPERIMENT models.

The LMPUs comprising the robot's brain are of two kinds: those specialized for carrying out specific tasks, and those specialized for more general tasks involving communication, motion, vision, cooperation, etc.

* TASK_LMPU: is a LMPU specialized for executing a particular task (i.e., is further specialized in the SES). When help or visual identification of an object is needed in performing this task, it relinquishes control to the ASSIS_LMPU or VISUA_LMPU, respectively. A detailed discussion of the decomposition of the TASK_LMPU is given in [14].

* ASSIS_LMPU: is a LMPU specialized for the task of requesting help from other robots. When it is activated, it initiates a COMM_PROTOCOL which tries to make contact with ROBOTs within its range and engage one that can provide the needed assistance.

* OFFER_LMPU: is a LMPU specialized for the task of dealing with incoming requests for help emitted from ASSIS_LMPUs of other ROBOTs. When activated, it decides if help can be offered, and if so, engages in a dialogue with the ASSIS_LMPU of the help-seeking ROBOT and sets up a rendezvous. It relinquishes control to the ACTION_LMPU to bring the ROBOT to the requestor's work site.

* ACTION_LMPU: is a LMPU specialized for directing the motion subsystem to bring the ROBOT to a given destination or position. It requests the current motion state from the MOTION component, and sends it new parameters (direction, speed, and time-step) for traveling to the vicinity of the destination. Once there, it directs the MOTION component in physically contacting the object or ROBOT at the destination. The touch channel is used for judging when contact has been established. Three ACTION_LMPUs are being implemented so far; MOVE_ACTION_LMPU for robot walking, PLUG_ACTION_LMPU for pull/push motion, and PLACE_ACTION_LMPU for put/get motion.

* VISUAL_LMPU: is a LMPU specialized for the task of visual identification of objects. It relinquishes control to ACTION_LMPU to bring the ROBOT to new locations when different viewing distances and perspectives are needed. Once there, it resumes control to accept a visual image and to identify objects by consulting its built-in recognition system[15].

This concludes the description of the structural and experimental decomposition of the space-borne fluid handling laboratory environment as described by its SES.

ACKNOWLEDGEMENTS

This research is supported by NASA-Ames Co-operative Agreement No.NCC 2-525, "A Simulation Environment for Laboratory Management by Robot Organization" and UA/NASA Space Engineering Research Center Project "High Autonomy Intelligent Control of an Oxygen Extraction Plant.

REFERENCES

[1] Nilsson, N.J., *Principles of Artificial Intelligence*, Tioga Pub. Co., Palo Alto, CA, 1981.

[2] Sacerdoti, E.D., *A Structure for Plans and Behavior*, Elsevier North-Holland, Inc., 1977.

[3] Steel, S., "Topics in Planning," in Lecture Notes in Artificial Intelligence, J. Siekmann eds., Springer-Verlag, 1987.

[4] Wang, Q. and F.E. Cellier, "Time Windows: An approach to Automated Abstraction of Continuous-Time Models into Discrete-Event Models," Proc. on AI, Simulation, and Planning in High Autonomy Systems, Tucson, 1990.

[5] Lioyd, M., "GRAFCET - Graphical Function Chart Programming," Proc. of the Conf. on Programmable Controllers '85, pp. 51-56, London, Olympia, July, 1985.

[6] Struger, O.J., "Programmable Controllers - Past and Future," Proc. of the Conf. on Programmable Controllers '85, pp. 1-6, London, Olympia, July, 1985.

[7] Hammond, K.J., *Case-Based Planning*, Academic Press, 1989.

[8] Chi, S.D., B. P. Zeigler, and F. E. Cellier, "Model-based Task Planning System for a Space Laboratory Environment," Proc. of SPIE Conf. on Cooperative Intelligent Robotics in Space, Boston, Nov., 1990.

[9] Zeigler, B.P., *Object-Oriented simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic systems*, Academic Press, 1990.

[10] Zeigler, B.P., "DEVS Representation of Dynamical Systems: Event-Based Intelligent Control," IEEE proc. Vol.77, no.1, Jan.1989. pp.72-80.

[11] Chi, S.D. and B.P. Zeigler, "DEVS-based intelligent Control of Space Adapted Fluid Mixing," Proc. of 5th conf. on Artificial Intelligence for Space Applications, May, 1990.

[12] Zeigler, B.P. and S.D. Chi, "Symbolic Discrete Event System Specification," *IEEE Trans. on System, Man, and Cybernetics*, Vol. 22, No. 4, July, 1992.

[13] Chi, S.D. and B.P. Zeigler, "Model-based Hierarchical Diagnosis for High Autonomy System," submitted to *Jour. of Intelligent and Robotic Systems*, 1992.

[14] Chi, S.D. *Modelling and Simulation for High Autonomy Systems*, Ph.D. Dissertation, Univ. of Arizona, 1991.

[15] Luh, C.J., *Abstraction Morphisms for High Autonomy Systems*, Ph.D. Dissertation, Univ. of Arizona, 1992.

[16] Cellier, F.E., A. Doser, G. Farrenkopf, J. Kim, Y. Pan, L.C. Schooley, B. Williams, and B.P. Zeigler, "Watchdog Monitor Prevents Martian Oxygen Production Plant From Shutting Itself Down During Storm," Proc. ISRAM'92 -- ASME Conf. on Intelligent Systems for Robotics and Manufacturing, Santa Fe, N.M., November 8--11, 1992 (invited paper).

[17] Albus, J.S., H.G. McCain, and R. Lumia, *NASA-NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)*, NBS Technical Note 1235, Robot Systems Division, Center for Manufacturing Engineering, National Technical Information Service, Gaithersburg, MD, 1987.