'91 KACC 1991. 10. 22~24

MODEL-BASED DESIGN FOR HIGH AUTONOMY SYSTEMS

S.D. Chi, B.P. Zeigler and S.H. Park*

Electrical and Computer Engineering
University of Arizona
Tucson, AZ, 85721, U.S.A.

*Electrical Engineering
Yonsei University
Seoul, Korea

Abstract

This paper presents the principles for design of autonomous systems whose behavior is based on models that support the various tasks that must be performed. We propose a model-based architecture aimed at reducing the computational demands required to integrate high level symbolic models with low level dynamic models. Model construction methods are illustrated to outfit such an architecture with the models needed to meet given objectives.

1. Introduction

Intelligent control is viewed as a new paradigm for solving control problems [1]. Its relatively narrow interpretation of the "control problem" does not include for example, the control needed by a system to diagnose and repair itself after significant insults to its physical structure. However, requiring greater degrees of autonomy from a system forces a more expanded view. To achieve autonomy, artificial intelligence is a one means to this end.

Although criteria for artificial intelligence[3,4] implicitly include autonomy, robotics research is realizing that autonomy requires the integration of AI decision making components together with perception and action components [5]. To include such diverse systems as self-operating factories, and telerobotic laboratories along with autonomous land and space vehicles in the autonomy umbrella, we employ a definition [7]:

> *Autonomy is the ability to function as an independent unit or element over an extended period of time, performing a variety of actions necessary to achieve pre-designated objectives while responding to stimuli produced by integrally contained sensors.*

Saridis [8] developed a three layer hierarchy (execution, coordination and management) for intelligent control which is supposed to reflect increasing intelligence with decreasing precision. Antsaklis *et al.* [2] refine the hierarchy to an arbitrary number of layers, depending on the particular application. The coupling of control and information at various layers characterizes the framework recently proposed by Albus [9,10,11]. Albus elaborates an architecture of seven levels, each of which has its own Task Decomposition and Execution, Sensory Processing, World Model, Global Memory, and Value Judgement components. Components at one level communicate with each other and with corresponding components at higher and lower levels. The Albus architecture is the most general and elaborate attempt to integrate perception, decision and action to achieve intelligence/autonomy. However, it leaves many details unspecified, particularly of interest here, the role of models.

In the proposed *model-based architecture*, knowledge is encapsulated in the form of models that are employed at the various control layers to support the predefined system objectives. It recognizes that an autonomous system must maintain models in a variety of formalisms and at various levels of abstraction. Lower control layers are more likely to employ conventional differential equation models with symbolic models more prevalent at higher layers. A key requirement is the systematic development and integration of dynamic and symbolic models at the different layers. In this way, traditional control theory can be interfaced with AI techniques.

An autonomous system could in principle, base various functional aspects such as planning, operation, diagnosis, and other activities on a single comprehensive model of its environment. However, such a model would be extremely unwieldy to develop and lead to intractable computations in practice. Instead, our architecture employs a multiplicity of partial models to support system objectives. As indicated, such models differ in level of abstraction and in formalism. The partial models, being oriented to specific objectives, should be easier to develop and computationally tractable. However, this approach leads to sets of overlapping and redundant representations. Concepts and tools are needed to organize such representation into a coherent whole. Structure and behavior preserving morphisms from model theory[12,13,14] can connect models at different levels of abstraction so that they can be developed to be consistent with each other and can be consistently modified. An organized model base enables the agent to deal with the multiplicity of objects and situations in its environment and to link its high level plans with its actual low level actions through well-defined morphism mappings [15].

2. System Entity Structure / Model Base Concept

The *System Entity Structure/Model Base (SES/MB)* framework was proposed by Zeigler [14] as a step toward marring the dynamic-based formalism of simulation with the symbolic formalism of AI. It consists of two components: a *system entity structure* and *model base*. The *system entity structure*, declarative in character [13,14], represents a knowledge of decomposition, component taxonomies, and coupling specification and constraints. The *model base* contains models which are procedural in character, expressed in dynamic and symbolic formalisms. The *entities* of entity structure refer to conceptual components of reality for which models may reside in the model base. Also associated with entities are slots for attribute knowledge representation. An entity may have several *aspects*, each denoting a decomposition and therefore having several entities. Associated with an aspect is *coupling* information needed to interconnect the entities of that aspect. An entity may also have several *specialization*, each representing a classification of the possible variants of the entity.

One application of the SES/MB framework is to the design of systems. Here the SES serves as a compact knowledge representation scheme of organizing and generating the possible configurations of a system to be designed. To generate a candidate design we use a process called *pruning* which reduces the SES to a so-called *pruned entity structure*(PES). Such structures are derived from the governing structure by a process of selecting from alternatives where ever such choices are presented. Not all choice maybe selected independently. Once some alternatives are chosen, some options are closed and others are enabled. Moreover, rule may be associated with the entity structure which further reduce the set of configurations that must be considered. In the model-based architecture of high

autonomy systems, the pruning is a initial planning process so that the sequenced/planned model can be selected.

As shown in Figure 1, pruned entity structure are stored along with the SES in files forming *the entity structure base*.

Hierarchical simulation models may be constructed by applying the *transform* function to pruned entity structures in working memory. As it traverse the pruned entity structure, *transform* calls upon a retrievals process to search for a model of the current entity. If one is found, it is used and transformation of the entity subtree is aborted. *Retrieve* looks for a model first in working memory. As it traverses the pruned entity structure, *transform* calls upon a retrieval process to search for a model of the current entity. If one is found, it is used and transformation of the entity subtree is aborted. *Retrieve* looks for a model first in working memory. If no model is found in working memory, the *retrieve* procedure searches through model definition files, and finally, provided that the entity is a leaf, in pruned entity structure files. A new incarnation of the *transform* process is spawned to construct the leaf model in the last case. Once this construction is complete, the main *transform* process is resumed. The result of a transformation is a model expressed in an underlying simulation language such as DEVS-Scheme [14] which is ready to be simulated and evaluated relative to the modeler's objective. The fact that the *transform* process can look for previously developed pruned entity structures, in addition to basic model files, has an important consequence for reusability.

Thus, the SES/MB framework provides an ability to develop model-based design of high autonomy systems. It can supports:

1) multiplicity of partial models to support system objectives (multifacetted modelling).
2) integration of dynamic and symbolic models at different layer (hierarchical architecture).
3) multi-abstraction to integrate related models (system morphism).
4) selection/retrieval of initial/changed planning models (pruning and reusability).

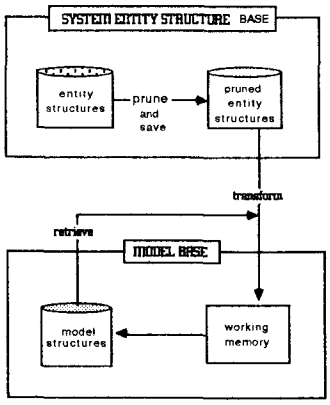

Figure 1. The System Entity Structure/Model Base (SES/MB) Environment

3. Functional Aspects of High Autonomy System

With valid models [12], an autonomous system is able to manage various tasks such as planning, operation, diagnosis, and error recovery to deal with complex objectives. Approaches to these aspects have been developed in each research field so that there are many overlaps as well as discrepancies between each aspect. In an integrated system, such aspects cannot be considered independently. For example, planning requires execution, and diagnosis is activated only when abnormalities are detected during execution. Figure 2 shows an framework for integration.

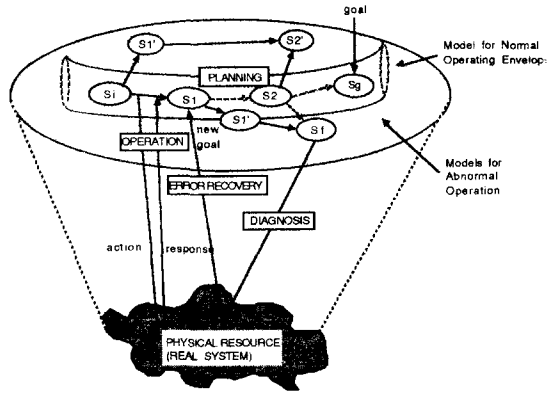

Figure 2. Characteristics of Functional Aspects

Planning is defined as "Reasoning how to achieve given goal." It employs a suitable model to map from initial state (S_i) and goal state (S_g) to a sequence of states (S_i, S_1, S_2,.., S_g) and associated commands within a normal operation envelope. Once a plan is setup, it should be faithfully executed. "Execution with verification" maps from the input command and its expected normal responses to success/fail. As long as execution is successful, it continues until achieving the goal. But, if it fails (reaching a state S_f), the diagnosis function should be activated. Diagnosis is defined as "Discovering the cause of the failure". It maps from observed symptoms to reachable abnormalities. Having identified the causes, the autonomous system should be able to recover the faulty state of real system or model to normal state. The error recovery is defined as "Knowing how to characterize the problem". It maps from causes to new goal (S_1). Note that, in contrast with planning/operation functions, the diagnosis/recover functions are activated only when the abnormality is detected.

3.1 Planning Function

There are two major approaches in 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 (case-based planning) views planning as remembering, i.e., retrieving and modifying existing plans for new problems [16].

Figure 3 illustrates our planning approach to build autonomous execution structures, where planning is viewed as a pruning operation which generates a candidate structure. In our model-based approach, planning is achieved by pruning a System Entity Structure (SES) to select Pruned Entity Structures (PESs) from alternatives [17]. 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 embeded.

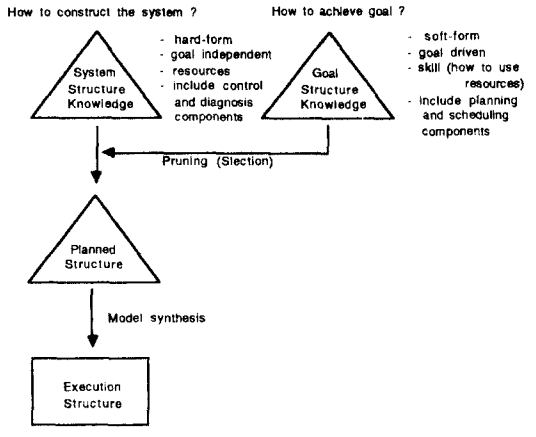

Figure 3. Planning Concept: Viewing Planning as Pruning

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[14,18].

3.2 Operation Function

The event-based control paradigm realizes the intelligent control by employing a discrete eventistic form of control logic represented by DEVS formalism [18]. In this control paradigm, the controller expects to receive confirming sensor responses to its control commands within definite time windows determined by its external 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. Figure 4 depicts the event-based operation concept using time window.

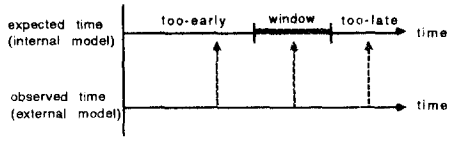

Figure 4. Event-based Control Concept

The operational model used by event-based operation has a state transition table that is abstracted from more detailed model. The state transition table keeps a knowledge of a state, input, next state, output, next-time-advance (lower bound), and its window time. The window associated with a state is determined by bracketing the time-advance values of all transitions associated with the corresponding states in its lower models.

The operator obtains information from the planner and expected responses times and windows, using the state table of its operational model. The operator issues these commands to the lower level operator or lowest level 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. Thus if the model is valid and operation proceeds normally, the underlying homomorphic relation is maintained between the model and lower levels. The operator ceases interacting with the device as soon as any discrepancy, such as *too-early* or *too-late*, occurs in this relationship and calls on a diagnoser to investigate it.

3.3 Diagnosis Function

Diagnosis 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: when detected the symptom, the behavioral diagnoser can discover the fault which is occurred within currently activating model unit. Once the controller has detected a sensor response discrepancy, the 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 which information it can gather from auxiliary sensors, the local diagnoser tries to discover the fault that occurred. The diagnostic model is an inversion process going from external effects to underlying causes.

If the local diagnosis fails, the global (higher level) diagnoser is activated. It refers its possible diagnoses in the diagnostic model and verifies whether faults have actually occurred in the indicated lower models. The global diagnostic model has a cause-effect table which is obtained by symbolic simulation to generate all causal trajectories of component models by marking those that reach states exhibiting the detected symptom. More precisely, it first builds symbolic simulation environment, i.e. collect its children models and attach corresponding fault generators, and then simulate to find the actual faults. More details on deep diagnosis using symbolic simulation are in [19].

4. Model-based Shallow/Deep Reasoning

To manage and judge such multi-functional aspects, autonomous system should have shallow/deep reasoning capabilities. The reasoning methods can be built depending on functional aspects, levels of hierarchy, and system objectives/requirements. Thus, it is important to have coherent methodologies that can support shallow/deep reasoning capabilities, and systematic integration.

Reasoning can be viewed as generating a cause-effect table. In our model-based approach, the shallow reasoning is about atomic level whereas the deep reasoning is about coupled level. Figure 5 represents the shallow reasoning approach where the cause-effect table can be directly obtained by associating each model (depending on parameter range) with resultant output (symptom). In contrast, the deep reasoning approach shown in Figure 6 uses symbolic simulation to generate every possible trajectories which indicate the causal relationships between component models.

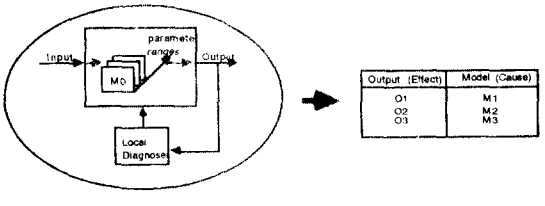

Figure 5. Model-based Shallow Reasoning

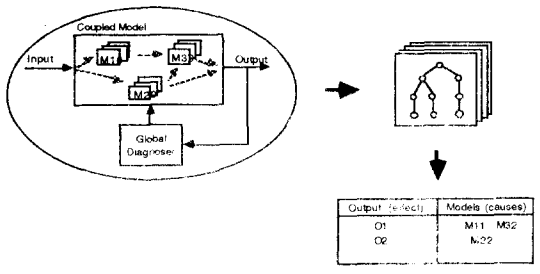

Figure 6. Model-based Deep Reasoning

5. Intelligent Unit Architecture

The functions described in a previous section have to be coupled within an unit in order to interact each other. We use term "Intelligent Unit" as a smallest unit for all functions (Figure 7). Each functional block is developed based on the endomorphism concept using engine-based modelling methodology as discussed next.

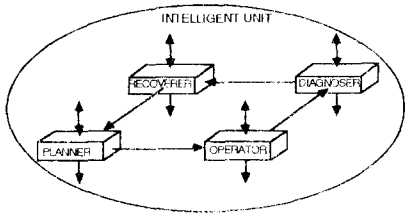

Figure 7. Structure of an Intelligent Unit.

5.1 Endomorphic Modelling 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 [14]. As illustrated in Figure 8, in order to control an object, an autonomy system needs a corresponding model of the object to determine the particular action to take. The internal model used by intelligent unit and external model are related by abstraction, i.e., some form of homomorphic relation. The inference engine asks to its internal model about the necessary information for activating real system (external model).

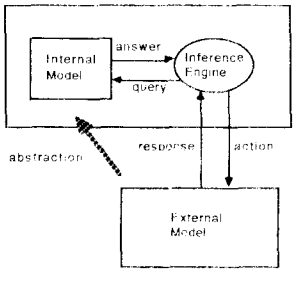

Figure 8. Endomorphic System Structure

5.2 Engine-based Design Methodologies

Typical expert systems comprise a domain-independent inference engine and domain-dependent knowledge base. The inference engine examines existing knowledge base decides the order in which inference made. The engine-based modelling approach provides a clear separation between domain-dependent model base and domain-independent inference engine so that it allows an automatic generation of model base preserving homomorphism. Figure 9 shows an engine-based modelling concept and a family of functional aspects realized by engine-based modelling concept.

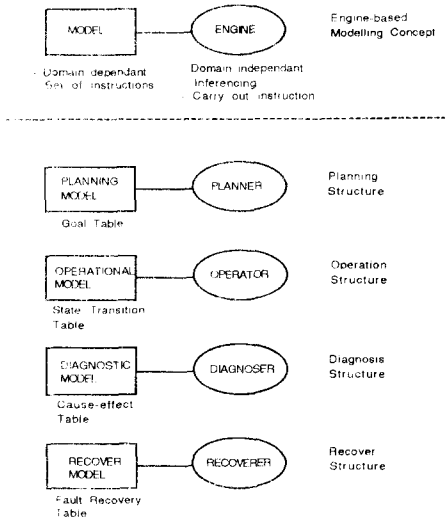

Figure 9. Engine-based Modelling Concept

6. Hierarchical Development of Intelligent Units

To cope with a complex problem, the high autonomy system requires multiple intelligent units that are coupled in hierarchical fashion, where each functional models of an intelligent unit in the hierarchy must have a valid abstraction relations each other. Figure 10 illustrates an autonomous system development based on hierarchical abstraction relationships which is discussed next.

Model base developing methods depend on levels of hierarchy: coupled level and atomic level. The former is about the structural knowledge whereas the latter is about the behavioral knowledge. The major constraints in constructing coupled level models are system objectives, requirements, and resource availabilities, etc. In contrast, atomic level models are constructed depending on dynamic constraints such as time. By integrating every functional aspects into an intelligent unit and by coupling every intelligent unit coherently, the model-based architecture of autonomous system can be built.

6.1. Hierarchical Abstraction Process

The abstraction is based on homomorphic preservation of the equipment input-output behavior where inputs are operation commands to the equipment and outputs are responses of finite-state sensors attached to the equipment to observe its state. Selection of controls and sensors must reflect the operation objectives. The discrete event model abstracts incremental micro-state transitions from the continuous model and replaces them by nominal times taken for macro-state transitions (which corresponding to crossing of sensor thresholds). And also discrete event model abstracts/composites its children models.

In our approach, we have multiple abstraction related models. Figure 11 represents an abstraction related models. M is a continuous state model of the system being controlled (the most refined model considered for it). Then the models are related by abstraction, i.e., some form of homomorphic relation. MB is a discrete event model derived from M, and PMB, OMB, DMB, and RMB are different abstraction

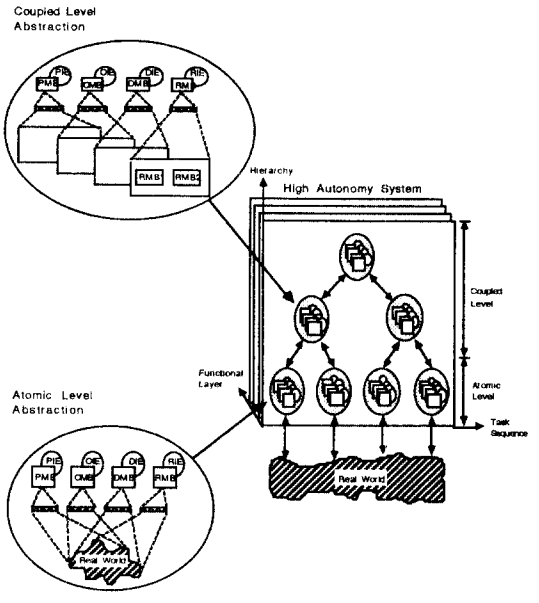

Figure 10. High Autonomy System Development Based on Hierarchical Abstraction

models of MB. Each abstraction is governed by an underlying morphism. MB serves as the external model of each equipment, whereas PMB, OMB, DMB, and RMB serve as the internal models of intelligent agent.

A group of PMB, OMB, DMB, and RMB can be again abstracted/ composited into higher coupled level models to represent more global state transitions (used by higher level agents). In this way, the higher level models use their own models, which are abstracted/composited from the lower level models, to control their sub-models.

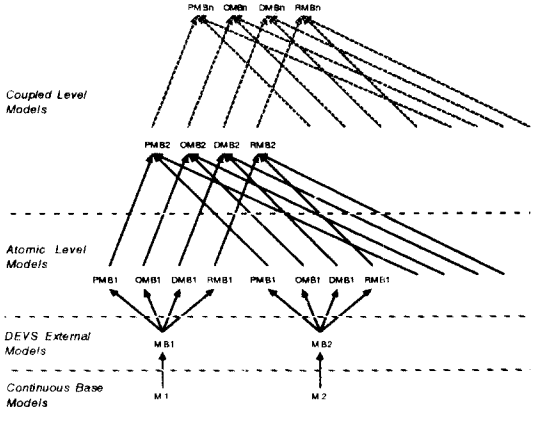

Figure 11. Abstraction Related Models

6.2. Autonomous System Generation

The overall methodology of the autonomous system generation in a model-based simulation environment is in Figure 12. The task formulation module receives an objective. It then retrieves SES from ENBASE and generates the plan structure by using planning part of SES or partitioned PESs. By pruning execution part of SES using plan structure, we have a planned execution structure and finally simulation structure by synthesizing corresponding models in MBASE, where models can be exist in advance or generated automatically.

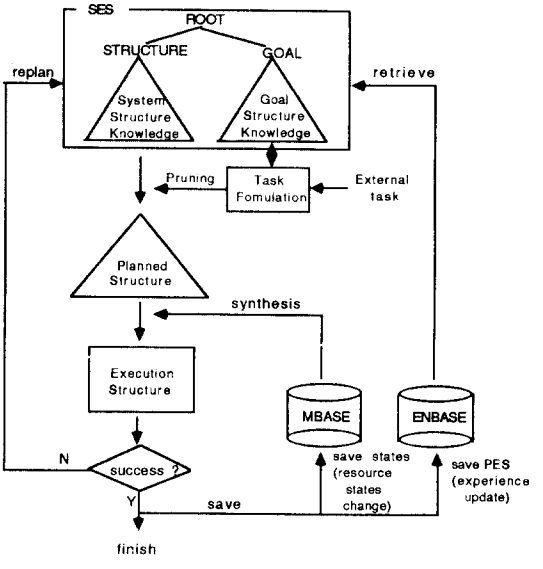

Figure 12. Autonomous System Generation Methodology

Once execution (simulation) is done, the resultant model states and/or planed structure are updated into MBASE and/or ENBASE for future reuse, respectively.

Note that the plan structure and execution structure can be implemented differently depending on the application domain, but conceptually it represent same structure. We now show the autonomous system generation procedure in more methodological way.

Phase I: Plan Generation

Once the basic environment is built, the next phase is the planning structure generation. When receiving given goal command, planning structure is generated in top-down fashion (task decomposition) as shown in Figure 13(a).

Phase II: Model Construction

Next phase is model base construction illustrated in Figure 13(b), where the necessary models can be retrieved from MBASE or automatically generated from the lower level models. This multi-layered hierarchical model base generation can be done in bottom-up fashion (model abstraction). The resultant structure represents the domain dependent knowledge base structure.

Phase III: Engine Attachment/Integration

By attaching domain independent engines such as planner, operator, diagnoser, and recoverer which is able to control corresponding models, we have multi-agent structure. Now by coupling those agents, we can obtain autonomous system architecture shown in Figure 13(c).

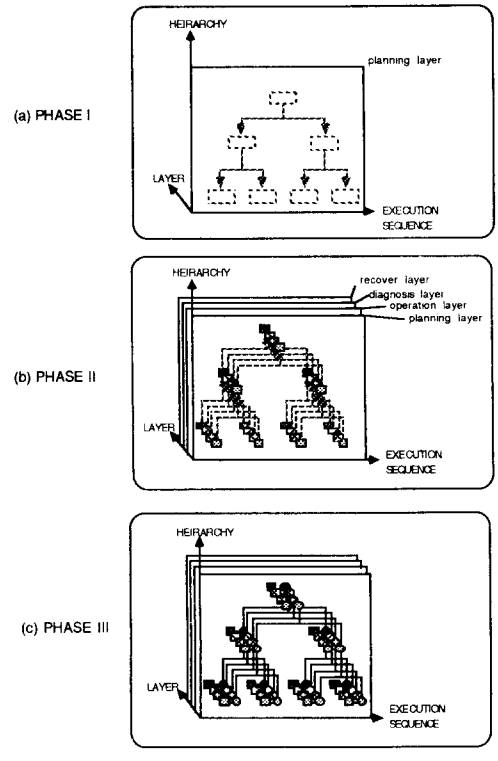

Figure 13. Autonomous System Generation Procedure

7. Conclusions

Building on the basis of the SES/MB framework as implemented in DEVS-Scheme, we have extended the ability of our knowledge/model base tools to support model-based design of high autonomy system through the ability to build multi-agent, multi-function, multi-abstraction, multi-level architecture coherently. The main characteristics of proposed architecture are as follows:

1. Time-based formalism (DEVS): provide coherent integration between symbolic and numeric models.
2. Hierarchical modularity: reusability, testability.
3. Atomic level and coupled level : clear distinction between behavioral and structural knowledge.
4. Deep reasoning as well as shallow reasoning: powerful reasoning capability.
5. Endomorphism: homomorphism preserving intelligent agent design methodology.
6. Engine-based design: domain independent system.
7. Event-based control logic: robustness, reduce sensor complexity, increase diagnostic capability (timing relation), integration of symbolic/numeric data.
8. Multi-functional capability: coherent development of various function to cope with complex problem.
9. Remembering: reusability of experienced structure.
10. Systematic abstraction process (morphism): automatic generation of knowledge base of each functional aspect.

References

[1] A. Meystel, "Intelligent Control: A Sketch of the Theory," *J. Intelligent and Robotic Systems*, vol.2, No.2&3, pp.97-107, 1989.

[2] P.J. Antsaklis, K.M. Passino and S.J. Wang, "Towards Intelligent Autonomous control Systems: Architecture and Fundamental Issues", *J. Intelligent and robotics systems*, Vol.1, pp.315-342, 1989.

[3] A. Newell, "Putting it All together," In: *Complex Information Processing: The Impact of Herbert A. Simon* (eds: D. Klahr, K. Kotovosky), Lawerence Erlbaum, Hillsdale, NJ, 1988.

[4] M.A. Fischler and O. Firshein, *Intelligence, The Eye, The Brain, and The Computer,* Addison-Wesley Pub. Co., Reading, MA, 1987.

[5] T. Kanade, A roundtable discussion: Present and Future directions: trends in Artificial Intelligence, OE Reports, SPIE, September, 1989.

[6] B. Chandrasekaran, A roundtable discussion: Present and Future directions: trends in Artificial Intelligence, OE Reports, SPIE, September, 1989.

[7] NASA, *The Space Station Program*, 1985.

[8] G.N. Saridis, "Intelligent Robotic controls", *IEEE Trans. on Auto. control.*, AC-28, No.5,1983.

[9] J.S. Albus, "The Role of World Modeling and Value Judgment in Perception", Proc. IEEE Conf. on Intelligent Control, September, Philadelphia, PA, 1990.

[10] J.S. Albus, "A Theory of Intelligent Systems", Proc. IEEE Conf. on Intelligent Control, September, Philadelphia, PA, 1990.

[11] J.S. Albus, "Hierarchical Interaction Between Sensory Processing and World Modeling in Intelligent Systems", Proc. IEEE Conf. on Intelligent Control, September, Philadelphia, PA, 1990.

[12] B.P. Zeigler, *Theory of Modelling and Simulation*, New York, NY : Wiley, 1976 (reissued by Krieger Pub. Co., Malabar, FL,1985).

[13] B.P. Zeigler, *Multifaceted Modelling and discrete Event simulation,* Academic press, 1984.

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

[15] B.P. Zeigler, C.J. Luh, and T.G. Kim, "Model Base Management for Multifacetted Systems", Proc. AI, Simulation and Planning in High Autonomy Systems, pp. 25-35, IEEE Press, Tucson, March, 1990.

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

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

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

[19] B.P. Zeigler and S.D. Chi, "Symbolic Discrete Event System Specification", *Conf. on AI, Simulation and Planning in High Autonomy Systems*, Florida, April, 1991 (revised version is submitted to *IEEE Trans on Systems, Man, and Cybernetics*).