Using System Entity Structure For Hierarchical Scheduling In An Intelligent Environmental Control System

Tag Gon Kim
Department of Electrical and Computer Engineering
University of Kansas
Lawrence, KS 66045

ABSTRACT

This paper describes an approach to specification of hierarchical schedules in a schedule subsystem with an intelligent environmental control system called the AI-Based, Distributed Environmental Control System (AIDECS). Our approach to schedule specification is based on a knowledge representation scheme called system entity structure. A system entity structure called the Timed Action Language (TAL) organizes a family of all possible long-/mid-/short-term schedules. The user specifies specific schedules by pruning the TAL. The specified schedules are translated into low-level schedule objects, which are converted into a list of activities for real-time execution.

1. INTRODUCTION

An intelligent environmental control system must be capable of managing environmental activities and conditions in an integrated fashion. Such activities range from planning of agricultural activities at the highest level to real-time control of environmental parameters at the lowest level. The activities at different levels give rise to hierarchical schedules which facilitate timing of events to occur at different time scales. Design of such an intelligent control system involves disciplines such as artificial intelligence/expert system, automatic control, operations research, discrete-event system theory, and others (Hoshi and Kozai, 1984; Saridis, 1987).

This paper describes a scheduling subsystem in such an intelligent environmental control system called the AI-Based, Distributed Environmental Control System (AIDECS) (Kim and Zeigler, 1990). The scheduling subsystem in the AIDECS supports specification of hierarchical schedules in a uniform manner. Our approach to schedules specification is based on the system entity structure – a knowledge representation scheme using a labeled tree with attached variable types (Zeigler, 1984).

The system entity structure, originally used to represent structure knowledge of a system with a family of variant models, can represent sub-components, decomposition, and taxonomic information about a system. In a hierarchical schedule specification, a system entity structure organizes a family of possible schedules from which a specific schedule can be selected by the user through the *pruning* operation. The specific schedule is translated into a low-level schedule object which eventually converted into a list of rule-like activities for real-time evaluation.

This paper is organized as follows. First, it presents the overall system architecture for the AIDECS and briefly explains its subsystems. It then gives a short review of the system entity structure formalism. The paper presents details of schedule specification and its translation into low-level objects that have been implemented in Scheme, a LISP dialect, for execution on PC/ATs or TI Explorers.

2. SYSTEM ARCHITECTURE OVERVIEW

The AIDECS (Figure 1) consists of Consultation Expert System (CES), User Interface System (UIS), Constraints Checker (CC), Schedule Manager (SM), Schedule Executor (SE), Schedule Decision Expert System (SDES), Control Data Base (CDB), and a Real-Time Control and Data Acquisition System (RTCDAS). The CES contains the Integrated Pest Management Expert System (IPMES) and the Diagnostic Expert System (DES).

The UIS enables the user to specify a short-/mid-/long-term schedule for environmental activities and parameters in hierarchical manner. The CC decides whether the schedule is acceptable or not by comparing it with a set of constraints contained within it, and sends it to the SM. The SM generates the specification of a hierarchical schedule, translates the specification into a schedule object, and sends the object to the SE. The SE transforms the object into a set of activities, each of which has slots for a condition/action pair, and others. The SE continually evaluates each activity in the set in conjunction with the SDES; if the condition of an activity is satisfied, the associated action in the activity is fired, which sends micro-level control signals to the RTCDAS. This signal has information on setpoints for environmental parameters such as temperature, humidity, and location where control actions are taking place. Data points that the RTCDAS collects by sensors are maintained in the CDB which the CES and the SDES might access as required.

3. THE SYSTEM ENTITY STRUCTURE

System entity structure (SES) is a representation scheme which contains the decomposition, coupling, and taxonomy information for a system (Zeigler, 1984). Formally, SES is a labeled tree with attached variable types that satisfies five axioms – alternating mode, uniformity, strict hierarchy,

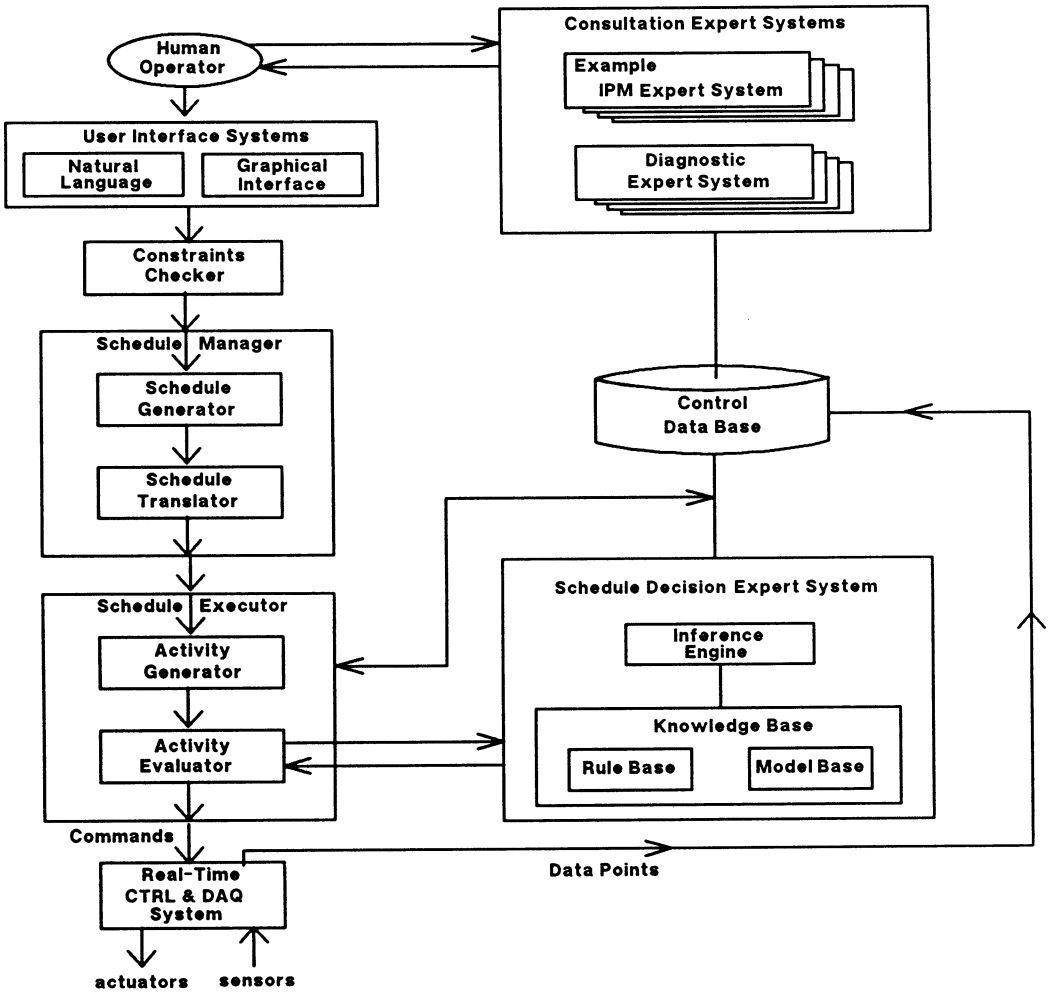

Figure 1. Overall System Architecture.

valid brothers, and attached variables. Detail description of the axioms is available in (Zeigler, 1984).

There are three types of nodes in SES – *entity, aspect,* and *specialization* – which represent three types of knowledge about systems. The *entity* node, having several *aspects* and/or *specializations*, corresponds to a model component that represents a real world object. The *aspect* node (a single vertical line in the labeled tree of Figure 2) represents one *decomposition*, out of many possible, of an entity.

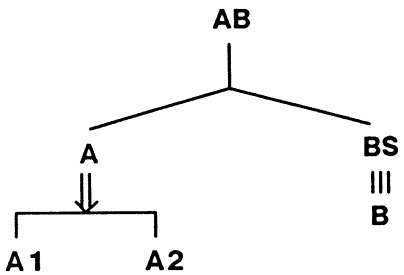

Figure 2. System Entity Structure.

Thus, the children of an aspect node are entities, distinct components of the decomposition. The *specialization* node (a double vertical arrows in the labeled tree of Figure 2) represents a way in which a *general* entity can be specialized into *special* entities.

A *multiple entity* is an entity that represents a collection of homogeneous components. We call such components a *multiple decomposition* of the multiple entity. The aspect of such a multiple entity is called *multiple aspect* (triple vertical lines in the labeled tree of Figure 2). Note that instead of presenting all BS's components, only one B is placed in the labeled tree.

Pruning extracts a sub-structure of the SES by selecting one aspect and/or one specialization for each entity in the SES. The *pruning* operation also expands multiple entities as well as assigning values of attributes attached to entities in the SES.

SES has been realized in Scheme environment. The realization called ESP-Scheme was described in (Kim and Zeigler, 1989). ESP-Scheme is originally used as a means of

representing structural knowledge for a simulation model in knowledge-based modeling/simulation research (Kim, 1988). Here, we use ESP-Scheme as a tool for specification of hierarchical schedules in the AIDECS.

4. SCHEDULING SUBSYSTEM

4.1 Timed Action Language

The schedule generator (SG), a subsystem of the schedule manager, generates hierarchical schedules which facilitate timing of events to occur at coarse- and fine-grained time and space units. An entity structure organizes a family of all possible schedules by its representation scheme; it can represent components, decompositions, and taxonomic knowledge of a schedule. The operation pruning defined on the entity structure allows the SG to generate a pruned entity structure representing a schedule specified by the user.

Using ESP-Scheme, we developed an entity structure called Timed Action Language (TAL) to organize a family of all possible schedules for the environmental control. As shown in Figure 3, entities, decompositions, and specializations with attached attributes in the TAL represent all necessary information for the SG to generate user-specified schedules. For example, *entities* WHERE, START-DATE, END-DATE, CYCLE, and ACT&TIMS along with their attached attributes designate region, start date, end date, period, and a sequence of control actions with associated time intervals, respectively. A *specialization* BIOME-SPEC specifies that a region (Biome) can be one of three types, namely, FOREST, SAVANNAH, or IPM-GREENHOUSE, each of which has its own plots, each of which has its own sectors. Such specialization allows the user to adjust sensitivity of regions in which control actions are taking place for the specified duration.

Note that the TAL has a multiple entity ACT&TIMS whose child entity ACT&TIM specifies a control action with an associated time interval. Since the pruning operation on a multiple entity can generate an unlimited number of multiple children entities, the user can specify any sequences of control actions and their associated time intervals.

A schedule is specified in a hierarchical manner. Such a hierarchical schedule is represented by a set of pruned entity structures organized in hierarchical fashion. An attribute *put-in-effect* attached to the entity ACTION is used to maintain an ordered list of entity structures for an entity structure, which represent sub-schedules to be executed in a sequence of specified intervals. Each sub-schedule, in turn, has its own sub-schedules, and so on. Such a capability allows the SG to generate hierarchical schedules on any level of hierarchy in time units. Putting into effect more than one schedule with the same interval results in a joint schedule, a set of schedules being executed together such as temperature and humidity schedules.

Specification of a hierarchical schedule can take advantage of information inheritance; an entity structure transmits all information specified for a schedule, except

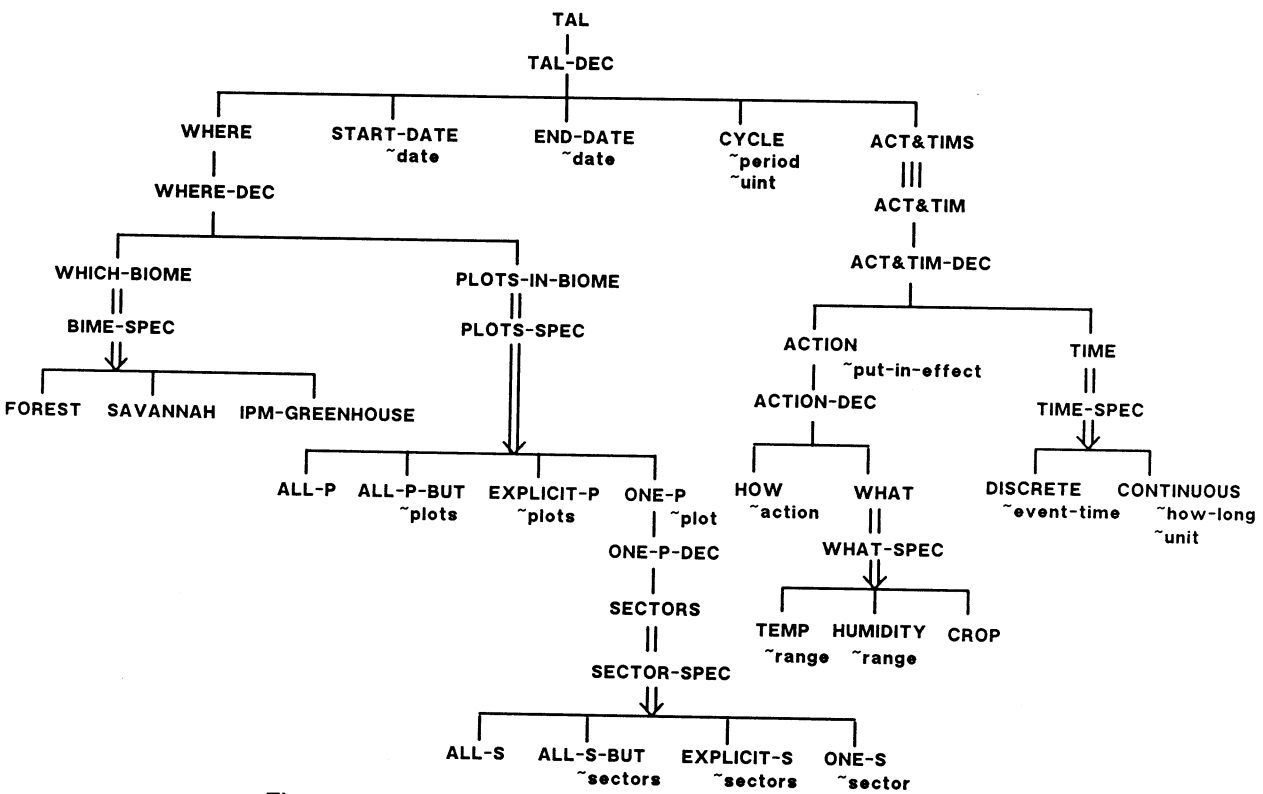

Figure 3. Entity Structure for Timed Action Language (TAL).

cycle and action and associate timing, to its sub-entity structures. For example, a start date specified in an entity structure is inherited to all its sub-structures as default. However, a new start date, if specified in a sub-structure, can override the inherited one.

4.2 Schedule Specification: Pruning the TAL

In a hierarchical structure, pruned entity structures with no sub-structures specify actual control actions, while those with sub-structures maintains a list of the sub-structures with their associated timing intervals. To explain how such actual control actions are specified by pruning the TAL, let us consider the schedule specified for both temperature and humidity as:

from 1991/1/1 to 1993/12/31
for all PLOTS in IPM-GREENHOUSE Biome
for every 24 hours period
maintain temperature at 55 degrees for first 4 hrs
at 60 degrees for next 4 hrs
at 70 degrees for next 6 hrs
at 65 degrees for next 6 hrs
at 60 degrees for last 4 hrs
maintain humidity at 70 % for first 12 hrs
at 75 % for last 12 hrs.

The UIS accepts information above and outputs a set of variables with their values assigned. Then the SG inputs output information of the user interface system and prunes the TAL. The SG generates two pruned entity structures for temperature and humidity schedules, respectively. Let us call the pruned entity structures for the temperature control p:temp and for the humidity p:humi. To obtain the p:temp involves the following procedures:

- assign values of attributes attached to the entities,
- select specialized entities under generalized ones,
- generate five ACT&TIMs for five control actions, and
- specify generated ACT&TIMs by assigning values of attributes and selecting entities under each such ACT&TIM entity.

More specifically, 1991/1/1 is assigned as the value of the attribute date attached to the entity START-DATE for the p:temp. Similarly, the above values of end-date, period, and period-unit are assigned to values of attribute date attached to the entity END-DATE, and attributes period and unit attached to the entity CYCLE, respectively. The values IPM-GREENHOUSE of biome-type and ALL of plot-type above are used to select entities IPM-GREENHOUSE and ALL-P for the p:temp under BIOME-SPEC and PLOTS-SPEC specializations in the TAL, respectively. Since the temperature schedule above has five temperature values with specified durations, the SG generates five ACT&TIMs for p:temp under the multiple entity ACT&TIMS.

4.3 Schedule Translation

High-level schedules specified by pruning the system entity structure TAL need to be translated into low-level schedule objects, which are eventually converted into an activity list for real-time execution. For the translation, we define a module called *SCHEDULE* as shown in Figure 4. The module is a template from which instant schedule objects can be created.

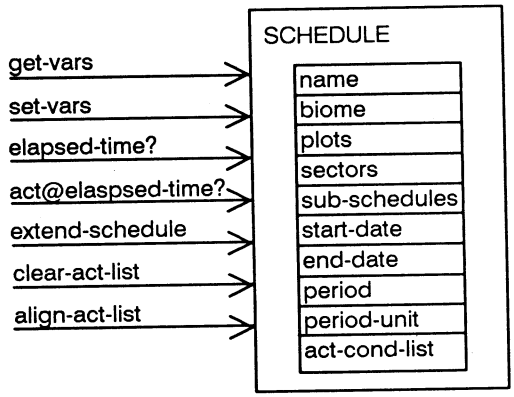

Figure 4. SCHDULE Module.

4.3.1 SCHEDULE Module

The *SCHEDULE* module is a package of local variables for information on a schedule and associated operations on the variables. Local variables within the *SCHEDULE* contain such information as name, locations, timing, actions, sub-schedules, and others for a given schedule. The variable *name* is used as a schedule identifier. Locations where a schedule is carried out is represented by three local variables, namely, *biome, plots,* and *sectors*. Variables *start-date, end-date, period,* and *period-unit* contains timing information for a schedule. For hierarchical scheduling, the variable *sub-schedules* is included which keeps a list of sub-schedules of the current schedule. The variable *act-cond-list* maintains a list of actual actions and associated conditions to carry out such actions.

Main operations on the *SCHEDULE* are accessing and assigning the values of all variables, which we represent *get-vars* and *set-vars* in Figure 4, respectively. The operation *extend-schedule* adds a set of condition-action pairs to the *act-cond-list*. The operation *clear-act-list* makes an empty list for *act-cond-list*. The operation *align-act-list* rotates the list of action-condition pairs such that the action-condition pair at a specified time (usually current time) is located at the first in the list. The query *elapsed-time?* returns the elapsed time since a current schedule has started. The query *act@elapsed-time?* returns a action-condition pair(s) at the elapsed time.

4.3.2 Translation Procedure

The procedure of schedule translation is as follows:
1. Create space object
2. Attach sector(s) to plot(s), and plot(s) to biome(s)
3. Create schedule object(s)
4. Attach the schedule object(s) to space object(s)

4.3.3 Space Objects

Locations where control actions take place are called compartments. Each compartment consists of biomes, each of which consists of plots, each of which, in turn, consists of sectors. The entity WHERE in the TAL specifies information on such locations. The WHERE consists of a biome and plots in the biome. If only plot in the biome is selected, a sector(s) in the plot needs to be specified.

In an object-oriented implementation, classes for location objects are defined as follows. The class *compartments* is defined as a general class for locations. It has a class variable *sub-compartments* and an instant variable *schedules*. Methods defined in the class *compartments* include *show-schedules*, *add-schedule*, *remove-schedule*, and others. A subclass of *compartments* is the class *biome*, the subclass of which is the class *plots*, which, in turn, has the subclass *sectors*. Creation of space objects begins with creation of biome objects, followed by plot objects, followed by sector objects.

Once objects for biomes, plots, and sectors are created, objects for plots and sectors need to know where they belong to. For this, plots objects are attached to a biome which plots belong to, and sector objects are attached to a plot to which they belong. By doing so, each compartment knows its sub-compartments, each of which, in turn, knows its sub-compartments, and so on. Dividing a compartment such a way allows the user to specify schedules for different locations in a distributed manner.

4.3.4 Access Functions

To facilitate the access to the values of attributes attached to entities in pruned entity structures, the entity structure manager (ESM) developed in ESP-Scheme (Kim and Zeigler, 1989) has been extended. Now, the ESM maintains a list of four-tuples called the *accessfn-tab* for each entity structure. A four-tuple in the *accessfn-tab* has access function name, unpruned entity structure name, entity name, and attribute name. The ESM has facilities to manipulate the *accessfn-tab* and to access value of an attribute for an entity in the *accessfn-tab*. For example, suppose that a four-tuple in the *accessfn-tab* is (*'period? e:tal 'cycle 'period*). Then the function call (*accessfn 'period? p:tal*) in ESP-Scheme returns the period of a schedule that is an attribute of an entity *period* specified in the pruned entity structure *p:tal*. Note that the pruned entity structure name *p:tal* is used to access period of a schedule even if the *accessfn-tab* has name of the unpruned entity structure, *e:tal*.

4.3.5 Schedule Objects

Creation of a schedule object starts with creating an instant object from the *SCHEDULE* module. Once a schedule object is created, values of local variables within the object needs to be assigned by using operations defined on such variables. Some values of the variables may be obtained from the values of attributes of an entity in a pruned entity structure specifying a schedule. For such case, access functions maintained in the *accessfn-tab* can be used to access various values of attributes for the pruned entity structure. However, for a more complex case, pre-processing information in the pruned TAL is required before assigning the values of variables in the schedule object. For example, start-date and end-date specified in the pruned TAL may be accessed by using respective access functions and copied into *start-date* and *end-date* in the schedule object, respectively. However, the assignment of a value for the variable *act-cond-list* requires making a list of action-condition pairs using information associated the entity *act&tims* in the pruned TAL.

After creating a schedule object and assigning all of its local variables, the object are attached to associated space objects. Each schedule is assigned as a value of instance variable for all associated biome objects, plots objects, and sector objects. Thus, each space object knows what schedule it will carry out.

4.4 Schedule Evaluation

The concept of the schedule evaluator (SE) evolved from the canonical architecture of rule-based expert systems as shown in Figure 5. The if-then rules are replaced by activities containing a special section for time management. The Inference Engine is called an Activity Evaluator (AE) since inferencing is now only a part of the effect of rule evaluation. The AE achieves its rule evaluation by referencing the clock and the timing section of activities, in addition to the usual rule conditions. The user is replaced by the real world process to be managed. The usual question and response interaction between the Inference Engine and

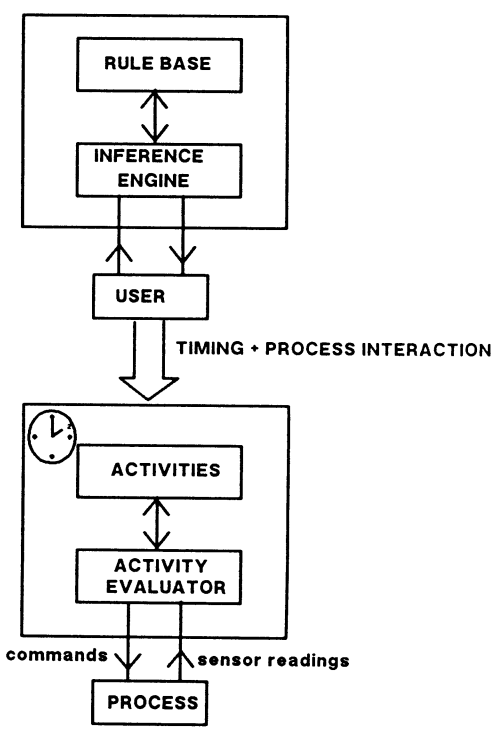

Figure 5. From Expert System To Schedule Evaluator.

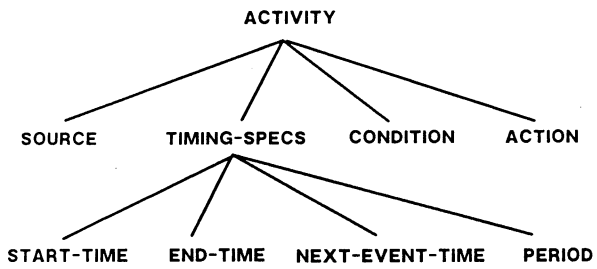

Figure 6. Activity Structure.

the user is replaced by a more complex interaction in which the AE exerts actions upon the process and receives sensorial feedback from it. The structure of an activity is as shown in Figure 6.

An activity will be fired, i.e., its action part will be executed by the AE, under the following circumstances:

1) its timing requirements are met
2) its condition is satisfied
3) the activities of its source schedule are not overridden.

In one cycle of rule evaluation, more than one activity can be fired. This is similar to the classifier systems approach (Holland et. at., 1986) and contrasts with the usual inference engine cycle in which a conflict resolution scheme selects only one rule to fire of those that are triggered. More details on the SE can be found in (Kim and Zeigler, 1990).

5. CONCLUSIONS

The concepts of system entity structure has been employed as a basis for hierarchical schedule specifications. By defining the system entity structure for the Timed Action Language, the user can specify his own schedules by pruning the Timed Action Language. A set of access functions defined for the TAL facilitates the translation of the pruned entity structures representing various schedules into low-level schedule objects. The scheduling subsystem within the AIDECS have been implemented and tested on the real-time basis using two PC/ATs coupled in a computer network.

REFERENCES

Hoshi, T. and T. Kozai (1984), "Knowledge-Based and Hierarchically Distributed Online Control System for Greenhouse Management," *Acta Horticulture* 148:301-303.

Holland, J.H., K.J. Holyoak, R.E. Nesbit and P.R. Thagard (1986), *Induction: Process of Inference, Learning, and Discovery,* The MIT Press, Cambridge MA.

Kim, Tag Gon (1988), "A Knowledge-Based Environment for Hierarchical Modelling and Simulation," Doctoral Dissertation, Electrical and Computer Engineering Dept., University of Arizona, Tucson, AZ.

Kim, Tag Gon and B.P. Zeigler (1989), "ESP-Scheme: A realization of System Entity Structure in a LISP Environment," *Proc in AI and Simulation of 1989 SCS Eastern Multiconference,* March, Tempa, Florida.

Kim, Tag Gon and B.P. Zeigler (1990), "An AI-Based, Distributed Environmental Control System for Self-Sustaining Habitats," *Artificial Intelligence in Engineering,* vol. 5, no. 1, pp. 33-42, Jan.

Saridis, George N. (1987), "Knowledge Implementation: Structures of Intelligent Control Systems," *Proc. IEEE International Symposium on Intelligent Control,* Jan., Phil, PA.

Zeigler, B.P. (1984), *Multifaceted Modelling and Discrete Event Simulation,* Academic Press, 1984.