Model Base Management for Multifacetted Systems*

Bernard P. Zeigler, Cheng-Jye Luh, and Tag-Gon Kim†
AI–Simulation Research Group
Department of Electrical and Computer Engineering
University of Arizona, Tucson, AZ 85721
†Department of Electrical and Computer Engineering
University of Kansas, Lawrence, KS 66049

ABSTRACT

A multifacetted system needs not just one, but many, models on which to base control, management, design and other interventions. These models differ in level of abstraction and in formalism. Concepts and tools are need to organize the models into a coherent whole. This paper deals with the management of model bases using system entity structure concepts. We show how the pruning process supports reuse of previously pruned structures. Concepts of context-sensitive pruning and partitioned entity structure bases are introduced to promote model base coherence and evolvability.

1. INTRODUCTION

The *Systems Entity Structure/Model Base (SES/MB)* framework was proposed by Zeigler [12] as a step toward marrying the dynamics-based formalism of simulation with the symbolic formalisms of AI. It consists of two components: a system entity structure and a model base. The system entity structure, declarative in character [4][5][13], represents knowledge of decomposition, component taxonomies, and coupling specifications and constraints. The *model base* contains models which are procedural in character, expressed in dynamic and symbolic formalisms. The *entities* of the 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 *specializations*, each representing a classification of the possible variants of the entity.

One application of the SES/MB framework is to the design of systems [8][9]. 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 choices may be selected independently. Once some alternatives are chosen, some options are closed and others are enabled. Moreover, rules may be associated with the entity structure which further reduce the set of configurations that must be considered.

As shown in Figure 1, pruned entity structures are stored along with the SES in files forming the *entity structure base* (there may also be subsidiary SESs in the base as we shall show). Hierarchical simulation models may be constructed by applying the *transform* function to pruned entity structures 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.

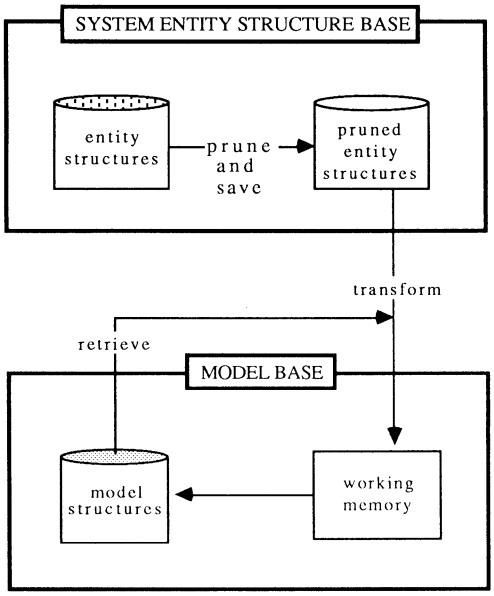


Figure 1. The System Entity Structure/Model Base (SES/MB) Environment.

*Research supported by NASA-Ames Co-operative Agreement No. NCC 2-525, "A Simulation Environment for Laboratory Management by Robot Organization".

The result of a transformation is a model expressed in an underlying simulation language such as DEVS-Scheme [13][15] which is ready to be simulated and evaluated relative to the modeler's objectives.

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 — as we shall see later.

As an example of a multifacetted system, let us consider a space-borne laboratory environment whose partitioned entity structure base shown in Figure 2 [16]. Such a laboratory environment is implemented as a centrally coordinated structure containing a SPACE model as coordinator, and OBJECTS as components. The SPACE model maintains the overall spatial relations of OBJECTS. Each OBJECT is specialized into ROBOT and INSTRUMENT. Each ROBOT is decomposed into MOTION-SYSTEM, SENSORY-SYSTEM, IMAGE-SYSTEM and COGNITION-SYSTEM. The MOTION-SYSTEM keeps track of a ROBOT's motion information such as position, direction, speed, etc. It accepts such motion commands as *move, change-speed,* etc. from the COGNITION-SYSTEM. When a robot changes its position, the MOTION-SYSTEM sends its new location and direction to the SPACE model. Basically, a ROBOT receives and sends messages via its SENSORY-SYSTEM. The IMAGE-SYSTEM is employed to generate an OBJECT's image in response to external visual interrogations.

The COGNITION-SYSTEM is also implemented as centrally coordinated structure consisting of a SELECTOR as coordinator, and MPUS (Model-Plan Units) as components. MPUs are task specialists that are designed by employing models of intended tasks and plans of action based on such models. MPUS is an example of a *multiple entity* (shown with three lines). Any number of instances may be generated for such an entity.

The MPUs comprising the robot's brain are of two kinds: those specialized for carrying out specific tasks and those that carry out more general tasks involving communication, motion, vision, co-operation, etc. TASK MPUs are specialists in operating specific INSTRUMENTS — each has models of its associated INSTRUMENT on which to base the operation of the INSTRUMENT and to diagnose faults if breakdowns occur. Note that there are several occurrences of the entity INSTRUMENT corresponding to the different contexts of its use. INSTRUMENT is said to be a *multiply occurring entity* as opposed to a multiple entity such as MPUS.

INSTRUMENT is a generic entity for laboratory instrument which is modeled much the same as a ROBOT. However, an INSTRUMENT has no COGNITION-SYSTEM. Among the specialized instruments available in the laboratory are BTL (bottle), MXR (mixer) and HTR (heater). An INSTRUMENT such as BTL has models used by the associated MPU as well as a model that represents the real instrument in a simulation. These models are expressed in different formalisms at different levels of abstraction. See Zeigler [15] for more details.

The set of interlocking entity structures, shown in Figure 2, specifies the family of robot configurations and objects that make up the simulated laboratory environment. We shall return to the example to illustrate the model base management concepts to be presented.

This paper describes the development of concepts and tools for *multifacetted* (multi-objective, multi-abstraction, multi-

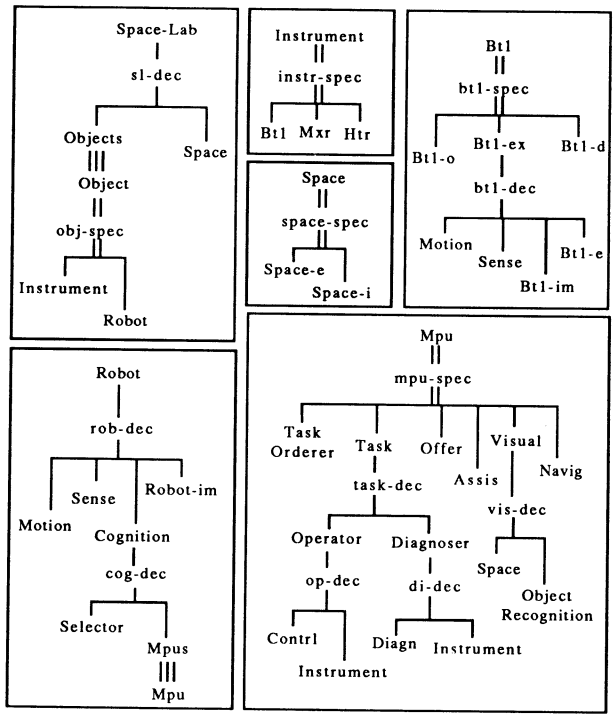


Figure 2. Entity structures for autonomous robot-managed space laboratory.

formalism) model/knowledge bases. Such systems support the maintenance of models in a variety of formalisms and levels of abstraction for a variety of operational, planning and diagnosis and other objectives.

First, we show how the pruning process supports reuse of previously developed pruned entity structures. Concepts of partitioned entity structure bases and context-sensitive pruning are then introduced to promote model base coherence and evolvability. The particular application context is the foregoing robot-managed space station laboratory simulation environment.

2. REUSE OF PRUNED ENTITY STRUCTURES

In DEVS-Scheme, a given SES can be pruned to create a number of *pruned entity structures (PES)* which are in turn transformed into simulation models [5][13][15]. The power to generate a large collection of models brings with it the need to archive the collection for *reuse*. Think of the SES as a powerful "printing press" and the collection of PES's the "books" to be archived. Properly cataloguing the books rolling of the press will help us find existing books that meet our current needs. The situation is described in Figure 3. In DEVS-Scheme, a PES uniquely represents a model in that the PES contains all the information needed by the transformation process to synthesize it. We assume that the model base is complete in the sense that it contains models for all the atomic entities referenced in the PES's.

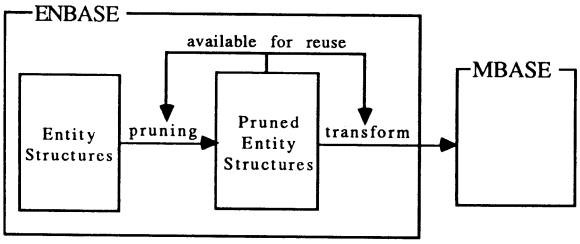


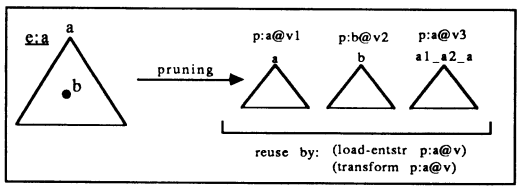
Figure 3. Reusability of pruned entity structures.

2.1 Requirement for PES Cataloguing

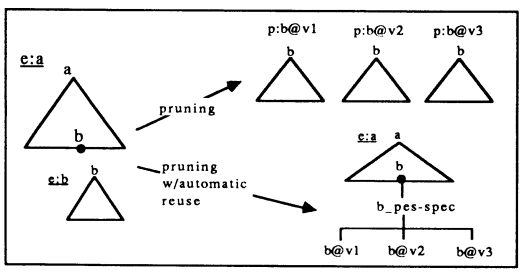
The basic requirement of cataloguing is simple to state: a PES should be catalogued in such a way as to facilitate subsequent recognition and retrieval.

Figure 4a depicts how this basic requirement is satisfied in DEVS-Scheme. Suppose that we have an entity structure E:A with root A. Pruning E:A we can start at any sub-entity B. This normally results in a PES with root B. If the user desires to store this PES, it will be saved under the name P:B@V, where *V* is a symbolic extension that the user supplies. Extensions can be used to identify different prunings all having the same root and therefore representing different models of the same entity. Employing a time-stamp extension would be reminiscent of software version control. However, in the present context, the extension should summarize the distinguishing characteristics of the PES.

For example, pruning E:MPU in Figure 2, the PES P:MPU@NAVIG suggests a navigator MPU; P:MPU@VISUAL might be another pruning of E:MPU characterized by its visual ability. Thus, we may save several prunings P:A@V1, P:A@V2, P:A@V3,...of E:A starting from the root A. To reconstruct a model from version P:A@Vi, we load it using (*load-entstr p:a@vi*) and then call the *transform* procedure with (*transform p:a@vi*).



(a)



(b)

Figure 4. Cataloguing of pruned entity structure for hierarchical reuse.

2.2 Hierarchical Reuse of PES Versions

Consider Figure 4b in which there is an SES E:A having a leaf entity, B. Also, an SES with B as root, E:B exists. We will return to see how this important situation might arise. As shown, suppose that several prunings, P:B@V1, P:B@V2,...of E:B have already been saved. Our next requirement for reusability is that when the leaf entity B is encountered in the pruning procedure, such versions automatically be available for user selection. This captures the hierarchical sense of model reuse, since existing versions of B can be "pulled of the shelf" and "plugged in" as components to a more encompassing model A.

To realize such hierarchical retrieval, the pruning procedure uses the following rules when encountering a leaf entity B:

1. if there is a model for B in the model base, skip the next steps,
2. if there is an SES, E:B and there are no PES versions of B, replace B in E:A by a copy of E:B and allow the user to continue pruning the new substructure copy of E:B,
3. if there is no SES, E:B and there are no PES versions of B, then add a new specialization B_PES-SPEC to B whose members are named in one-to-one correspondence with the versions of B. The entity selected by the user from B_PES-SPEC replaces B in the resulting PES of E:A. When transforming this PES, *transform* will encounter the leaf entity selected by the user, say B@Vi. At this point it invokes itself recursively to transform the corresponding PES P:B@Vi. The resulting model for B is employed as a component in the hierarchical model being synthesized, and
4. if both a system, and pruned, entity structures exist for B then allow the user to decide whether to prune the SES as in rule 2 or to select from existing versions of B as in rule 3.

As an example, suppose we have pruned the SES E:BTL in Figure 2 to obtain external, operational and diagnostic versions, viz., P:BTL@E, P:BTL@O, and P:BTL@D, respectively. In any SES containing BTL as a leaf entity, we can select from these existing versions of BTL, or prune E:BTL afresh to generate a new version needed in this particular situation.

3. PARTITIONED SYSTEM ENTITY STRUCTURES

As illustrated in Figure 2, more than one SES may exist in the entity structure base, each one representing a family of models for its root entity. In principle, every entity might have its own SES but this would lead to extreme fragmentation of the encoded knowledge. Reasons for giving an entity its own SES include: the large size of its family of possible prunings, its high likelyhood of being modified, and its occurrence in several places of existing SESs. Moreover, we shall see that partitioning is crucial to achieving model base coherence.

Figure 5 shows tools for creating and partitioning SESs. Given an SES, E:A with a leaf entity, B, we can use an operation, *cut-entstr* which:

- removes the substructure of B from E:A,
- reincarnates it in the form an SES, E:B, and then
- allows the user to prune E:B as many times as desired.

Of course, one may also create sub-SESs independently without extracting them from larger structures.

As discussed above, the pruning process is capable of "sewing together" pieces of SESs which fit. It does so only as it needs to, i.e., when it arrives at a leaf entity which has an associated SES. In this way, we avoid patching in SESs that might never be reached in the particular pruning being made. If desired, the user can piece together SESs with a command, *add-sub-entstr*.

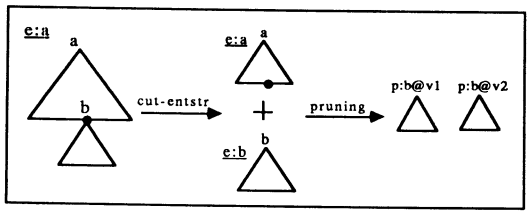


Figure 5. Partitioning of Entity Structures.

3.1 Context Sensitive Pruning

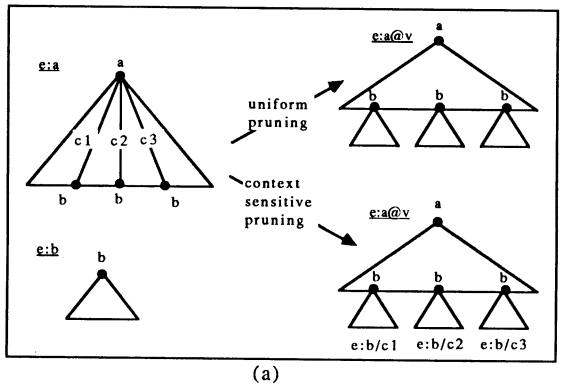
Due to the uniformity axiom of the SES [11][18] an entity subtends the same substructure where ever it occurs. Pruning however, must be able to break this uniformity so that different prunings can be made in the different occurrences. One way to understand the situation is to consider an SES E:A from which an SES E:B has been extracted (Figure 6a). This leaves an entity B which occurs at a number of places in the exterior of the structure (by uniformity, if B is a leaf entity in one occurrence it must be so for all occurrences). When pruning of E:A arrives at B, the user is given the choice of *context sensitive* or *insensitive* pruning. In the latter, E:B is pruned only once; by uniformity of E:A, B receives the selected substructure at each of its occurrences.

In *context sensitive* pruning, copies of E:B are made for each occurrence of B. Each copy is separately pruned, renamed and pasted back in the appropriate context. Due to the valid brothers axiom of the SES [11][18], each occurrence of B is reached by a uniquely labelled path from the root. This path (actually the minimal part needed for proper discrimination) provides the context needed by the user for pruning. Each item in a pruned copy of E:B is renamed to be distinct from occurrences in other copies of E:B.[1] Note that E:B may itself have entities that occur more than once. Thus context sensitive pruning is a recursive procedure.

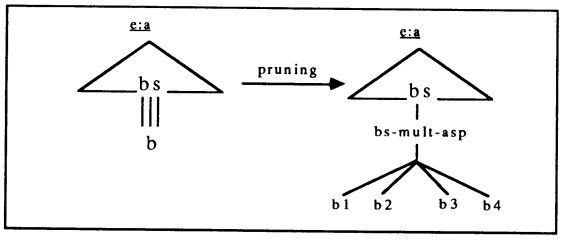
A multiple entity, as shown in Figure 6b, is pruned in a similar manner. The single entity B is replaced by a set B_0, $B_1, \ldots, B_n$ whose size is determined by the user.[2] The resulting SES clearly satisfies the valid brothers axiom. For each such brother B_i, a copy of E:B is pruned, renamed in the context of B_i (plus more of the path extending back to the root, if needed), and attached to B_i. Actually, for convenience, the user is asked to decide on a number of equivalence groups. Elements in the same group are given the same pruning (but different naming).

[1] In renaming, couplings and priority lists are also altered appropriately.
[2] This structure is transformed to a kernel model [4][15].

This obviates the need to specify each version separately. As an example, in pruning E:SPACE-LAB in Figure 2, we might select two equivalence groups for the multiple entity OBJECTS: the first group containing ROBOTS, the second containing INSTRUMENTS, all BTLs, say.



(a)



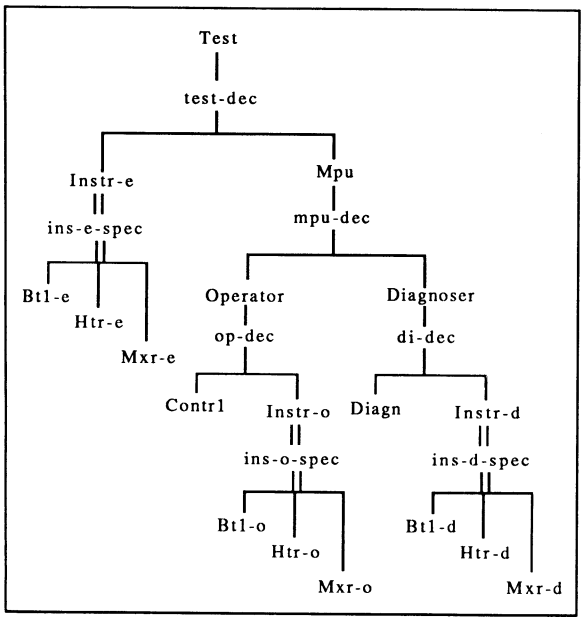
(b)

Figure 6. Pruning of multiply occurring entities (a) and of multiple entities (b).

4. MODEL COHERENCE AND CONTEXT SENSITIVE CONSTRAINT RULES

This section substantiates the claim made earlier that partitioning of SESs is crucial for supporting model coherence. First, consider an SES for testing a space adapted bottle shown in Figure 7. The specialized fluid handling MPU (Model-Plan Unit) [16] is decomposed into an operator for filling and emptying a bottle and a diagnoser for discovering the causes of any operational faults. A model of the bottle is referenced in three places but each is labelled differently: BTL-E (external model), BTL-O (operational model), BTL-D (diagnostic model). In this approach there is no formal way of recognizing that models of the same underlying entity are being referenced. The various models of BTL are dispersed throughout the structure. We can greatly improve coherence by using an SES in which the generic entity BTL replaces each of its special cases and by collecting them together as specializations in an SES, E:BTL, as in Figure 8. Organizing models by the entity they model, rather than the context they are used in, facilitates evolvability. Recall that models of an entity may be related by abstraction relationships so that when one is changes others must be amended to retain consistency. Indeed, this is the case for the models of BTL as of all INSTRUMENTS.

Choosing context-sensitive pruning, we can select the specializations, BTL-E, BTL-O, and BTL-D in the corresponding contexts, TEST-DEC, OP-DEC, and DI-DEC, respectively. This is a slight bit of extra work, but it need only be done once since the resulting PESs can be saved for hierarchical reuse (section 2.2).



Selection Rules:

If select Btl-e from Instr-e then select Btl-o from Instr-o
and select Btl-d from Instr-d

.....

Figure 7. Dispersed SES for testing an arbitrary instrument handling MPU with operational and diagnostic capabilities.

The advantage of such coherence is even more evident when we generalize the entity structure base so as to apply to an arbitrary instrument handling MPU. As shown in Figure 9, we can choose context insensitive pruning of E:INSTRUMENT to select a specialized entity such as BTL to uniformly replace the general entity INSTRUMENT in E:TEST. After this pruning step, the pruning of E:BTL proceeds as above. Note that the power in this approach is that the same SES E:TEST may be pruned to generate a family of models capable of testing MPUs for handling the various instruments organized in E:INSTRUMENT. Moreover the system is nicely evolvable: to add a new instrument, we place its name in E:INSTRUMENT and define a new SES for its models.

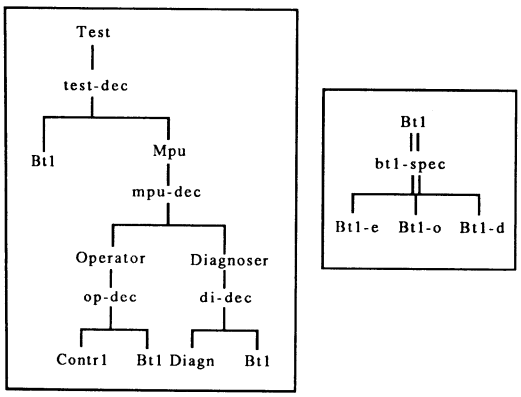
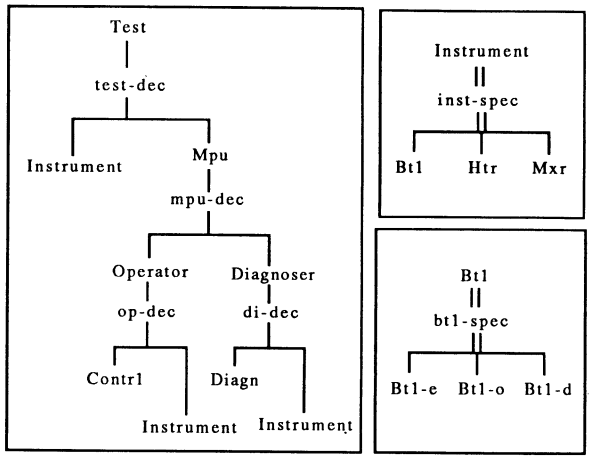


Figure 8. Entity structure base for testing a bottle handling MPU with operational and diagnostic capabilities.



Generalized context sensitive selection rules:

In the context test-dec, select x-e from x-spec
In the contex op-dec, select x-o from x-spec
In the context di-dec, select x-d from x-spec

Figure 9. Coherent Entity Structure Base for testing an arbitrary instrument handling MPU with operational and diagnostic capabilities.

Compare the above with an alternative SES in which the models of instruments are distributed into specializations according to context as in Figure 7. To add an instrument, we must add the proper specialized version in three different specializations. To generate a test model for a particular instrument, we must consistently select the appropriate representative in each of the specializations. As shown in Figure 7, constraint rules, of the form:

If select BTL-E from INSTR-E
then select BTL-O from INSTR-O
and select BTL-D from INSTR-D
....

can enforce the required consistency. A corresponding class of rules can be defined to assist context sensitive pruning of the form:

In the context TEST-DEC, select BTL-E from BTL-SPEC.
In the context OP-DEC, select BTL-O from BTL-SPEC.
In the context DI-DEC, select BTL-D from BTL-SPEC.

As shown in Figure 9, we can have the selection of an entity from a specialization be governed by context. Moreover, imposing syntactic patterns, we can state a generalized rule to replace an indefinite number of special cases of the same form.

5. EXAMPLE: SPACE-BORNE LABORATORY SIMULATION CONTINUED

The approach to entity structure base organization just outlined helps to organize the models employed in an autonomous system such as the robot architecture introduced earlier. The interesting, and complicating, fact of such autonomous systems is their use of internal models. Valid representations of such systems must also represent their internal models. We have argued that such models should be organized according to the entities they relate to rather than dispersed among the contexts they are

used in. Moreover, as we have seen, simulation models of such systems must incorporate external models of the same parts of reality in order to be able to test how well that modelled agents perform in their environment. Zeigler [15] shows how a base model of a system is abstracted into internal and external models.

As a concrete illustration, let us set up a fluid handling experiment in the space-borne laboratory environment. To perform such a task, a robot must first locate and identify a bottle required by external specification, then bring that bottle to the workspace, and finally perform the fluid handling. Thus, to set up such a particular laboratory environment, the SPACE-LAB SES shown in Figure 2 is pruned to generate one robot and one instrument (Figure 10). Upon reaching those entities with associated SESs, the pruning process will sew together the underlying SESs piece by piece. For example, the SES E:ROBOT will be plugged in to replace the entity ROBOT in the SPACE-LAB SES, and then E:MPU will replace the entity MPU within the robot. When the pruning proceeds further, the entity INSTRUMENT and the specialized instrument, BTL, will be replaced by E:INSTRUMENT and E:BTL, respectively. Here we might assume this is our first pruning, so there are no PESs available for BTL.

Note that the entity INSTRUMENT is a multiply occurring entity. We choose context insensitive pruning of E:INSTRUMENT to select a specialized entity, BTL to uniformly replace it. After this pruning step, we choose context sensitive pruning to select the specializations, BTL-EX, BTL-O, and BTL-D in the corresponding contexts, OBJECT-SPEC, OP-DEC, and DI-DEC, respectively.

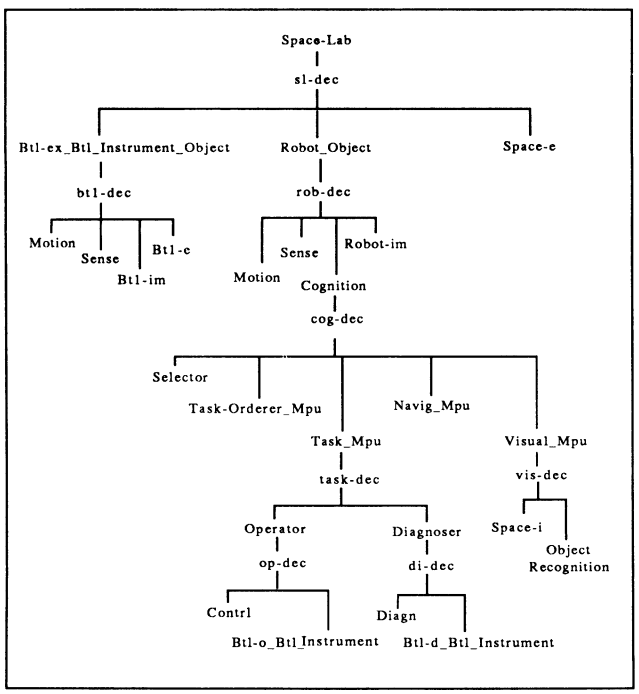


Figure 10. PES for robot-managed fluid handling experiment.

Within the robot's COGNITION-SYSTEM, MPUS is defined as a multiple entity. As stated earlier, we can prune such a multiple entity to generate any number of desired alternatives. In this example, the synthesized robot will contain one TASK-ORDERER, one VISUAL, one NAVIG, and one TASK, MPU. The multiply occurring entity, INSTRUMENT, within the TASK MPU is pruned as stated above. The resulting PES is shown in Figure 10.[3]

Among the specialized MPUs, the VISUAL MPU implements robot vision. Basically, the VISUAL MPU contains a world map and an object recognition model. The world map model, SPACE-I, maintains knowledge of locations and orientations of objects. It is updated whenever the robot successfully recognizes an object. When the robot starts the visualization, it checks the world map first to see whether there are objects located within the line of its sight. If so, the object that is closest to it is chosen as the expected target. Then whenever the image data come in, the VISUAL MPU applies the predefined classification system for visual identification of laboratory objects. The external model that corresponds to the robot's internal representation of an object is the image generator, e.g., BTL-IM. This external model responds with different images from various points of view. Such a vision modelling approach has been implemented by Luh and Zeigler [6].

After successfully identifying the bottle, the robot travels to the bottle's site and then brings the bottle to the workplace. Note that the robot consults its internal world map to determine the travel path and to avoid collisions. Moreover, the overall spatial relations of objects, which are maintained in the external model of space, SPACE-E, is updated whenever the robot changes its location. Note that the external model of space, SPACE-E, and its counterpart, the robot's internal world map model, SPACE-I, are selected by context sensitive pruning of E:SPACE.

Now let's consider the fluid handling experiment. The robot's internal fluid handling ability is realized in the TASK MPU as shown in Figure 10. The operational model, BTL-O, is implemented in the *table-models* class in DEVS-Scheme. Such a predictive model employs dynamic programming principles [7] to perform command sequence planning [15] by given goals. The diagnostic model of the bottle, BTL-D, is implemented in the *forward-models* class in DEVS-Scheme. Such a model provides rules to deduce the origin of a fault. The external model of bottle, BTL-E, representing the real bottle that the robot operates on, is able to respond to both operational commands and diagnostic probes.

To conduct such laboratory management, the robot's behavior is supervised by the TASK-ORDERER MPU. A more sophisticated liquid mixing experiment pruned from the same SES has been described by Chi and Zeigler [1].

6. CONCLUSIONS AND FURTHER RESEARCH

Building on the basis of the System Entity Structure/Model Base framework as implemented in DEVS-Scheme, we have extended the ability of our knowledge/model base tools to sup-

[3] As illustrated by ROBOT_OBJECT, the sequence of selections from specializations is reflected in the name automatically given a pruned entity; the sequence information is later employed to endow the entity with its inherited properties.

port multifacetted systems design through the ability to generate family of design alternatives, reusability of models and model base coherence and evolvability.

As described above, such an approach has been demonstrated in the design of a robot-managed space-borne laboratory simulation environment. However, much work remains to be done to extend the same organizational principles to achieve a multi-formalism, multi-abstraction simulation environment. First, such a system should provide tools for the systematic derivation of abstractions through the use of system morphisms [10][11] to integrate related models. The SES must be extended to represent knowledge about existing morphisms between models of an entity. Such knowledge should concern not only the definition of the morphism itself but also a specification of its domain of validity. Zeigler [10] presented an approach to this issue using experimental frame concepts. Moreover, to support multi-formalism modelling, a great amount of information should be associated with the entities in the SES. The kinds of knowledge that must be associated with a model include it's formalism, intended application, experimental frames that are applicable to it, etc. Such an approach might be implemented by embedding the system entity structure in a richer frame-based knowledge representation scheme called Frames and Rules Associated System Entity Structure (FRASES) [2][3].

REFERENCES

[1] Chi, S.D. and B.P. Zeigler (1990), "DEVS-based Intelligent Control of Space Adapted Mixing Process", (accepted for presentation on *Fifth Conference on Artificial Intelligence for Space Applications*, Huntsville, AL).

[2] Hu, J. (1989), *Towards A Knowledge-Based Design Support Environment For Design Automation and Performance Evaluation*, Doctoral Dissertation, University of Arizona, Tucson, AZ.

[3] Hu, J. and J.W. Rozenblit (1989), "Knowledge Acquisition Based on Representation for Design Model Development", In: *Knowledge-Based Simulation: Methodology and Applications* (eds.: P.A. Fishwick and R.D. Modjeski), Springer Verlag, Berlin.

[4] Kim, T., (1988), *A Knowledge-Based Environment for Hierarchical Modelling and Simulation*, Doctoral Dissertation, University of Arizona, Tucson, AZ.

[5] Kim, T., G. Zhang, B.P. Zeigler (1988), "Entity Structure Management of Continuous Simulation Models", *Proc. Summer Sim. Conf.*, Seattle.

[6] Luh, C.J. and B.P. Zeigler (1989), "Hierarchical Modelling of Mobile, Seeing Robots", *Proc. SPIE Conf. on Intelligent Robots and Computer Vision VIII: Systems and Applications*, Vol. 1193, pp. 141–150.

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

[8] Rozenblit, J.W., S. Sevinc, and B.P. Zeigler (1986), "Knowledge-based Design of LANs Using System Entity Structure Concepts", *Proc. Winter Simulation Conf.*, Washington, D.C., pp. 885–865.

[9] Sevinc, S. and B.P. Zeigler (1988), "Entity Structure Based Design Methodology: A LAN Protocol Example", *IEEE Transactions on Software Engineering*, Vol. 14, No. 3, March, pp. 375–383.

[10] Zeigler, B.P. (1976), *Theory of Modelling and Simulation*, Wiley, NY. (Reissued by Krieger Pub. Co., Malabar, FL. 1985).

[11] Zeigler, B.P. (1984), *Multifacetted Modelling and Discrete Event Simulation*, Academic Press, London and Orlando, FL.

[12] Zeigler, B.P. (1986), "System Knowledge: A Definition and its Implications", In: *Modelling and Simulation Methodology in the Artificial Intelligence Era* (eds.: M.S. Elzas, T.I. Ören, B.P. Zeigler), North Holland Pub. Co., Amsterdam, pp. 15–17.

[13] Zeigler, B.P. (1987), "Hierarchical, Modular Discrete Event Modelling in an Object Oriented Environment", *Simulation J.*, Vol. 49:5, pp. 219–230.

[14] Zeigler, B.P. (1989), "The DEVS Formalism: Event-based Control for Intelligent Systems", *Proceedings of IEEE*, Vol. 77, No. 1, pp. 27–80.

[15] Zeigler, B.P. (1990), *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*, Academic press, NY (in press).

[16] Zeigler, B.P., F.E. Cellier, and J.W. Rozenblit (1988), "Design of a Simulation Environment for Laboratory Management by Robot Organizations", *J. Intelligent and Robotic Systems*, Vol. 1, pp. 299–309.

[17] Zeigler, B.P. and T.G. Kim (1990), "The DEVS-Scheme Modelling and Simulation Environment", In: *Knowledge-Based Simulation: Methodology and Applications*, (eds.: P. Fishwick and R. Modjeski), Springer Verlag (to appear).

[18] Zhang, G. and B.P. Zeigler (1989), "The System Entity Structure: Knowledge Representation for Simulation Modeling and Design", , In: *Artificial Intelligence, Simulation and Modelling* (eds.: L.A. Widman, K.A. Loparo, and N. Nielsen), J. Wiley, NY, pp. 47–73.