Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

ABSTRACTION MORPHISMS FOR WORLD MODELLING IN HIGH AUTONOMY SYSTEMS

Cheng-Jye Luh and Bernard P. Zeigler

AI-Simulation Research Group
Department of Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721

ABSTRACT

A model-based autonomous system employs a multiplicity of models at various control layers to support the predefined system objectives. These models differ in level of abstraction and in formalism. Concepts and tools are needed to organize the models into a coherent whole. We have developed abstraction mechanisms for mapping models of laboratory instruments into operational abstractions, and for mapping a task plan hierarchy into an isomorphic tree of model abstractions. This paper deals with further efforts to develop abstraction mechanisms for systematic derivation of related models through the use of system morphisms. We describe an abstraction mechanism for abstracting spatial relations between the model of external world and the internal world model employed in an autonomous agent. We also show how such a mechanism supports world modelling coherence. Then we illustrate the integration of the world modelling approach and abstraction mechanisms in an autonomous multi-agent laboratory application.

1 INTRODUCTION

High autonomy is an extended paradigm that subsumes both control and AI paradigms. Emerging from the control field, *intelligent control* is viewed as a new paradigm for solving control problems. Based on the concept of *intelligent control*, several autonomous control architecture have been proposed to achieve autonomy (Albus 1990a, Antsaklis et al. 1989, Saridis 1983). These architectures, from minimum three layers (management, coordination, and execution) up to arbitrary layers depending on particular applications, characterize the hierarchical use of control and information at various layers. Zeigler and Chi (1990) propose a *model-based architecture* in which knowledge is encapsulated in the form of models that are employed at the various control layers to support the predefined system objectives. The model-based approach 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.

The model-based autonomous system design is based on the multifacetted modelling methodology (Zeigler 1990) that employs a multiplicity of models oriented to specific objectives. These partial models are more computationally tractable, more understandable, and easier to develop than comprehensive multipurpose models (Zeigler 1984, McRoberts et al. 1985, Fishwick 1989a,b). Also the multiplicity of abstracted models may provide an evolutionary path for the modelling process (Fishwick 1988, 1989a,b). Much recent research has recognized the need for multiple levels of abstraction (Murthy and Addanki 1987, Murthy 1988, Jokowicz 1989, Unruh 1989) and good representations and ability to change from one representation to another for efficient problem solving (Benjamin et al. 1990, Keller et al. 1989a,b). However, the proposed approaches lack criteria for valid abstraction. Such criteria are imposed by an explicit statement of the kind of morphic relation to be preserved relative to the objectives at hand. Sevinc (1990) developed a means to support automation of simplification of discrete event models. The simplification approach is intended to provide faster running lumped models rather than more understandable models. However, abstraction can provide both. Also this empirical approach does not address the problem of maintaining consistency of related models.

Figure 1 depicts the objectives oriented approach to model development. Here, a relatively complex simulation base model is abstracted into simplified

models, oriented to planning, operation, diagnosis, and other objectives. Such abstractions are based on the homomorphic concept that a state correspondence between base and lumped models must be preserved under transitions and outputs (Zeigler 1976, 1984). Morphisms differ in such details as the nature of the state correspondence and the lengths of micro-state transition sequences corresponding to macro-state transitions. Choice of a particular morphism establishes the criterion of validity for abstraction. Generally, for an abstraction to be valid, it must provide as good an answer to the question of interest as does the base model. In the objectives driven methodology, modelling objectives lead to asking specific questions about the behavior of a real system which in turn require the selection of suitable variables. Ultimately such a choice of variables leads to formulating an experimental frame. Such an experimental frame characterizes the circumstances under which a model or its real system counterpart is to be observed or subjected to experimentation. Thus, the correctness of abstraction can be tested against the base model by simulation in an experimental frame of interest. The behavior preservation assesses the validity of abstraction relative to the objectives at hand. Multifacetted methodology leads to multiple models that need to be organized into a coherent whole. We employ the system morphism concepts just described for this purpose. Such morphisms connect models at different levels of abstraction so that they can be developed to be consistent with each other and can be consistently modified.

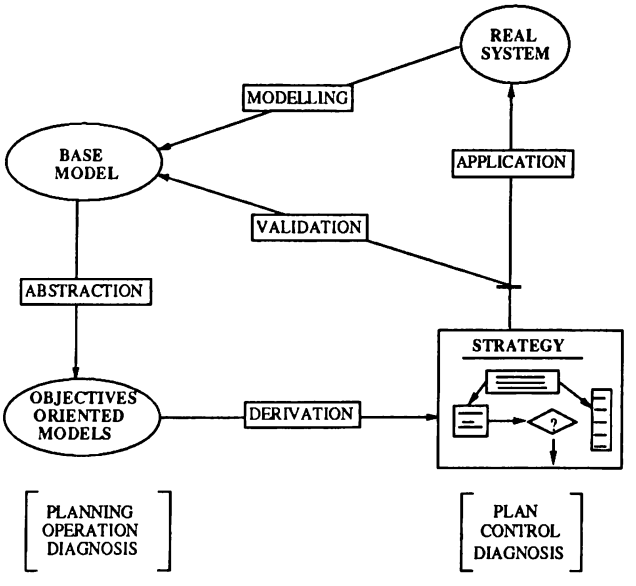


Figure 1: Objectives oriented model development.

We have developed a morphism class in DEVS-Scheme for mapping models of devices such as laboratory instruments to the operational abstractions needed for event-based control (Zeigler 1989, 1990). Such morphism abstraction supports model construction and model base consistency. DEVS-Scheme is based on the DEVS (Discrete Event System Specification), a system-theory based formalism supporting hierarchical, modular model construction and manipulation (Zeigler 1984, 1990). The morphism class *forw→table-morphisms* (Luh 1991) realizes the abstraction relationships from *forward-models* to *table-models* two classes in DEVS-Scheme.

An instance of *forw→table-morphisms* not only homomorphically relates existing models, but also has methods to actually construct a homomorphic table model from the given forward model. We have empirically verified correctness of the *forw→table-morphisms* mapping by simulation. The morphism instance is saved in the model base, and can be reactivated to automatically regenerate the corresponding table model whenever the forward model is modified. This ability greatly contributes to maintenance of model base consistency.

Moreover, we have implemented the morphism class, *PES→table-morphisms* (Luh and Zeigler 1991), for systematic generation of table models for the nodes of a task decomposition tree represented in pruned entity structure, which is the result of pruning, or extracting a hierarchical model specification from a system entity structure (Zeigler 1990). A corresponding execution structure is formed with the assignment of controllers and supervisors, respectively to the leaves and upper level nodes of the model tree. Such an abstraction approach has been demonstrated and tested in a model-based task-planning system for a robot-managed space-borne laboratory.

This paper describes further efforts to develop abstraction mechanisms for systematic derivation of related models through the use of system morphisms. We develop a morphism class for abstracting spatial relations between world models, external and internal to an autonomous agent. This paper is organized as follows. Section 2 provides an overview on endomorphic modelling of autonomous systems and model base management concepts. In section 3, we present the implementation of a morphism class for world modelling abstraction. The testing of morphism correctness is also discussed. In section 4, we illustrate the application of the morphism class to an autonomous multi-agent organization. Section 5 concludes this paper and considers some directions for future research.

2 MODEL BASES OF AUTONOMOUS SYSTEMS

To achieve realism, modelling of autonomous agents must be able to represent not only their decision making capabilities, but also the models on which such capabilities are based, – and the methodologies for building such models. Zeigler (1990) points out that the use of **internal** models plays a key role for valid representations of autonomous systems. Moreover, 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.

The model base of a high autonomy system may contain models, internal and external to the agent's cognition system:

- **internal models**: These models are used to model the agent's decision making capabilities for such interactions as operations, diagnosis, planning, etc. along with its modelling capabilities.
- **external models**: These models are employed to model the environment in which the agent operates, pieces of equipment the agent may operate, and its physical components such as motion subsystem, whose time characteristics need to be known for navigation.

Zeigler (1990) used the term *endomorphy* to refer to objects (systems, models, agents) in which some sub-objects use models of other sub-objects. An endomorphic simulation model might contain an agent and environment such that the agent has, and uses, a model of the environment and models of (parts of) itself in its decision making. Such self-embedding agents are termed *endomorphic agents.*

The approaches to model base management such as entity-based partitioning and context-sensitive pruning (Zeigler, Luh, and Kim 1990) and abstraction morphisms discussed here help to organize the models employed in an autonomous system such as the robot architecture in the space-borne laboratory (Figure 2). The entities possessing both internal and external models can be enumerated as follows:

- *objects*, including robots and instruments, have external models for perception interactions. Such a model represents how the object generates images in response to interrogations such as robot visual inspection requests. Internal model counterparts to these external models are organized into a classification system used by the robot to identify objects,
- *instruments*: each type of instrument is represented in internal models within an MPU in the brain related to operation and diagnosis; an external model of an instrument is used to respond to robot manipulations and to robot diagnostic probes, and
- *world map*: the locations and orientations of objects are maintained within an external model of space; correspondingly, each robot has an internal spatial map to help it determine the locations of objects it wishes to manipulate or avoid. Such a world model also helps to plan the actions needed to carry out an experimentation procedure and to navigate from place to place. The internal world map is a robot's internal representation of the external world (Albus 1990a,b,c, Zeigler 1990, Roth-Tabak and Jain 1989), i.e., an abstraction of the external world.

Each partitioned system entity structure (SES) represents a chunk of reusable knowledge. The pruning process will sew together the underlying SESs piece by piece upon reaching those entities with associated SESs. For example, to generate a laboratory robot, the SES E:ROBOT will be plugged in to replace the entity ROBOT in the STR SES, and then E:MPU will replace the entity MPU within the robot, and so on. As illustrated in Figure 2, 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. The robot consults its internal world map to locate expected objects, 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, are updated whenever the robot changes its location. Note that the external model of space, SPACE-E, and its counterpart, the robot's internal world model, SPACE-I, are selected by context sensitive pruning of E:SPACE. Organizing the models by the entities they concern rather than dispersed among the contexts they are used in facilitates model coherence and evolvability (Zeigler, Luh, and Kim 1990).

3 WORLD MODELLING

As pointed out in (Albus 1990a,b,c, Meystel 1988, Arkin 1989), the internal world model of an intelligent, autonomous system should consist of hierarchical decompositions on various levels of abstraction and resolution. World model knowledge at higher levels of abstraction includes symbolic representations

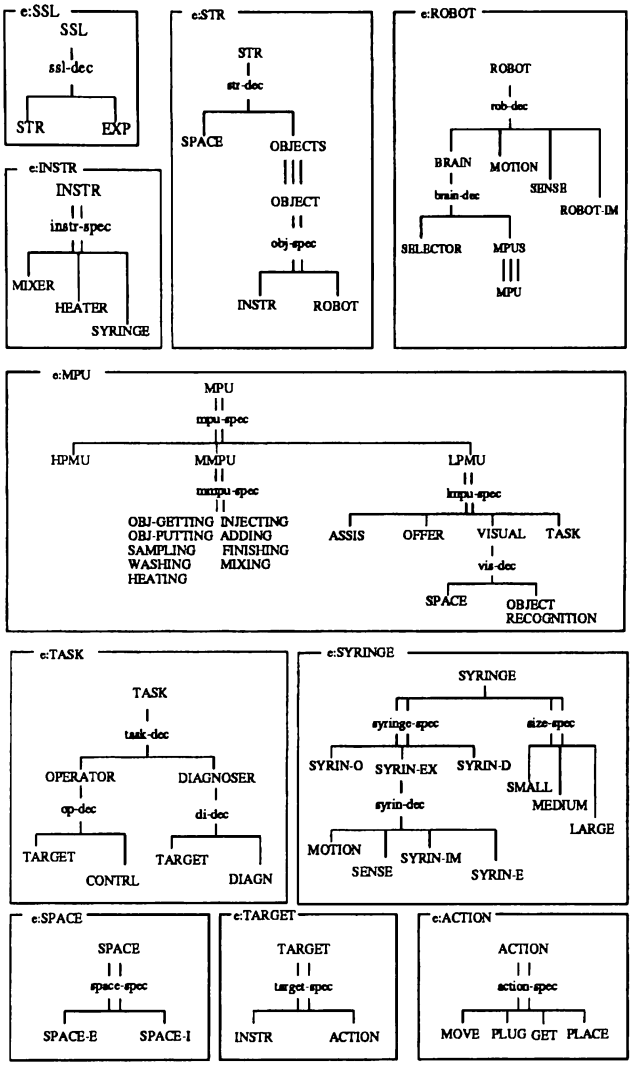


Figure 2: Entity structure base for an autonomous space-borne laboratory.

of the work place, objects, attributes of objects, relationships among them, events, etc. On the other hand, knowledge at lower levels of abstraction might be 3D cubic voxels (Roth-Tabak and Jain 1989 , Grant 1990) or 2D image pixels and image features (Albus 1990b) such as lines, edges, surfaces, etc. The latter knowledge is gained from perception of the environment by the autonomous system itself. In addition, some of the high level knowledge such as the attributes of objects is interpreted from sensory data at lower levels of abstraction.

In practice, the real world is rather complex. A model of the real world may contains knowledge about space, time, entities, events, temperature, humility, etc. In objectives driven modelling methodology (Zeigler 1984), the modeller's intended use of the model drives the process of model construction. Likewise, the agent's intended use of its internal model is a key consideration. A robot consults its internal world model for object recognition, global path planning, and collision avoidance in our design. Thus the spatial knowledge of objects need only be enough to serve such purposes. In our modelling approach, both the external model of space, SPACE-E, and the internal world model, SPACE-I, within a robot contain the spatial knowledge of objects. The external model of space represents the abstract real world in which robots operate. To represent the internal and external models of space, *space-models*, another sub-class of *atomic-models* in DEVS-Scheme was introduced.

3.1 Space-Models

The class *space-models* provides an object's identification, location, and orientation in a form of tuple in a relation, call the *map*. The entries are added to it using the method *assert* as shown in Figure 3. For stationary objects such as bottles which are put or stacked up on top of other objects, the symbolic location form (ON OBJECT) was introduced. For example in Figure 3, the bottle is located at (ON TABLE). When the entry is encountered, the bottle's location is inferred to be the table's location for numerical computation. This provides a means for relatively inexpensive mapping of all objects. For example, when a table moves, all objects on it retain their descriptions.

```
(make-pair space-models 'space-e)

(send space-e set-s (make-state 'phase 'passive
                                'sigma 'inf
                                'map (make-map)
 ))

;;; attributes: object location orientation

(send space-e assert-tuple
                  '(robot1 #(0 20) #(0 1)))
(send space-e assert-tuple
                  '(box #(0 30) #(1 0)))
(send space-e assert-tuple
                  '(table #(10 10) #(1 0)))
(send space-e assert-tuple
                  '(bottle (on table) #(.6 .8)))
(send space-e assert-tuple
                  '(robot2 #(100 100) #(1 0)))
(send space-e assert-tuple
                  '(robot3 #(30 30) #(1 1)))
```

Figure 3: The space model SPACE-E.

The underlying internal and external transition, and output functions of *space-models* implement a space engine. The space engine manipulates the map for internal uses within an agent such as detecting collision, checking neighborhood, and for external

communication purposes such as locating influencees, routing messages, computing relative orientation between objects, etc. The queries are serviced immediately after the currently activated event whereas the communication messages are delayed by the time defined in the instance variable, *delay-time*. The content of external input event is a structure defined by:

(define-structure content port value source channel echo-status)

In addition to the port-value pair, the slots, *source* and *channel* are employed for specifying message initiator and channel, respectively. The external input events that have non-null values in the *channel* slot are treated as communication messages. Different transmission media and sensory modalities are modeled such as light and vision, pressure and touch, etc. The user can apply the method *input?* to send queries such as *collide?, neighbors?* to the space model for detecting collision and checking neighborhood, respectively. The entries of the map can be displayed using the method *show-tuples*. For visualization, the method *get-nearest-tuple* finds the object that is closest to and is located within the line of sight of the seeing agent if one exists. The method *update-domain* is basically applied to update the internal world model's[1] map given the changes of external space model. Whenever an object moves around, a movement event is sent to the external space model. The latter records such events in a *motion-list*. The method *update-domain* proceeds in two steps. First, the internal world model goes through the morphism instance specified in its state variable, *pre-image* to find its counterpart external space model, and sends a *update* request to the external space model. The latter responds to the request by checking the *motion-list* and passing back the updated tuples, either moved within or out of the agent's working range. When such messages are received, the internal world model asserts the move-in tuples into, and retracts the move-out tuples from, the *map*. Moreover, the *update-domain* method also handles those objects that are put or stacked on top of other objects. For example, if a table is moved out of a robot's working range, the objects on the table are all out of range. Inversely, when the table is moved in, so are the objects on the table. The class *space-models* is shown in Figure 4.

[1] "Internal world model" is a space model employed within an agent. It is indeed an "internal space model". Both terms are used interchangeably without further distinction.

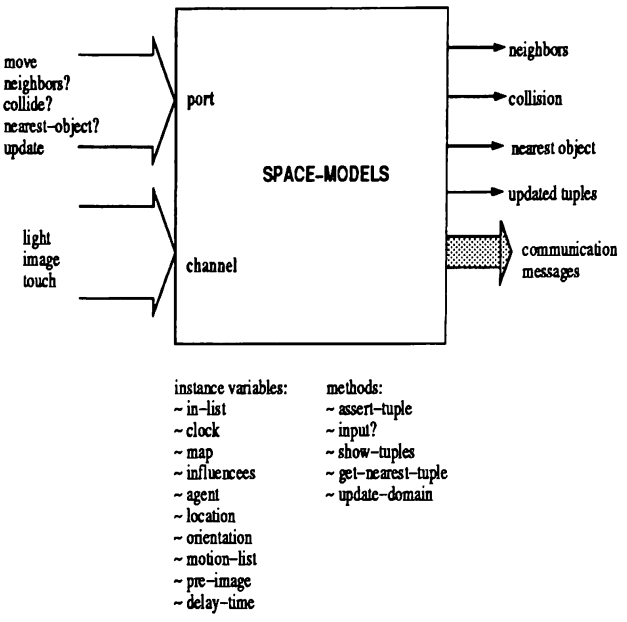


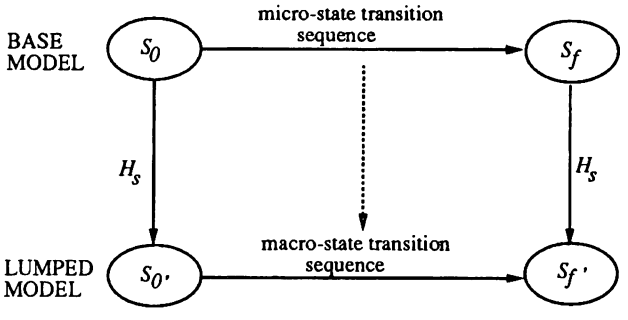
Figure 4: Class *space-models*.

3.2 World Models Coherence

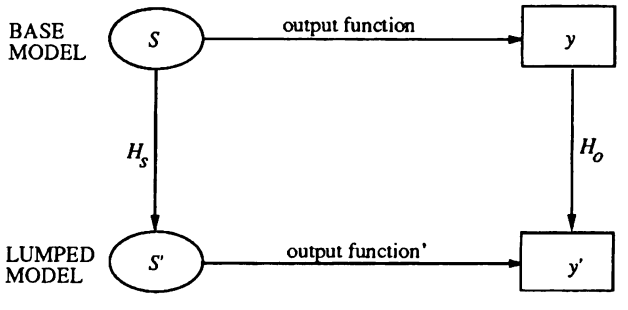
In an indoor environment such as a chemical laboratory, the overall spatial relations of objects can be predefined as a priori knowledge and stored in the external model of space (Albus 1990a, Arkin 1989). In our simulation experiments, every object sends its location and orientation with identification to the external model of space as part of the initialization process. This is a form of abstraction; only the spatial knowledge of objects is extracted. Thus the external model of space is homomorphic to the state of the real world initially. The external model of space is then updated by movements of robots and objects so as to remain consistent with the real world.

For an autonomous system to exhibit intelligent behavior, it must be able to perceive the dynamics of the environment. Albus (1990a) points out that perception is the establishment and maintenance of correspondence between the internal world model and the external real world. That is, an autonomous system's internal world model is updated from perception of the environment. An autonomous system must contain sensory processing capabilities for integrating and interpreting information from multiple sensors. Such capabilities, the focus of much current research on sensor-based autonomous systems, are treated at a high level of abstraction in our robot model. Due to the possible limitations of sensors and mobility, a robot may operate within a certain range. Thus a robot's internal world model can be

initialized from part of the spatial knowledge in the external model of space. In this way, the internal world model is homomorphic to the external model of space before the robot starts operation. If task targets are beyond a robot's working range, the robot can call for help from others via the ASSIS and OFFER MPUs (Zeigler 1990). As a side benefit, the approach to local focus reduces traffic in the laboratory environment. Assume all of the sensors and sensory processing facilities are satisfied, a robot should perceive all the spatial changes in its local environment. The updated world model is still homomorphic to the external model of space, although they are updated in different ways. Thus there is a *space→map* morphism relation that exists between the external model of space and the internal world model of a robot. Note that the morphism relation is valid in the experimental frame that the robot has well-functioned sensory processing capabilities to perceive all the spatial changes within its working domain.



(a) preservation of transition function



(b) preservation of output function

Figure 5: Commutative diagrams of homomorphism.

3.3 Morphism Implementation

The implementation of morphisms is based on the homomorphic concept that is depicted in the commutative diagrams shown in Figure 5. In Figure 5a, we start the base model in one of the states S_0 in a restricted set. The homomorphic mapping H_s yields a corresponding state $S_0\prime = H_s(S_0)$ in the lumped model. We then inject an input sequence into the base model and its corresponding version into the lumped model sending them to the states S_f and $S_f\prime$, respectively. These states also correspond under H_s. In Figure 5b, when the base model is in state S and the lumped model is in corresponding state $S\prime = H_s(S)$, the output values observed in these states are y and $y\prime$, respectively; the value $y\prime$ is also obtained by decoding y under output mapping H_o, that is $y\prime = H_o(y)$.

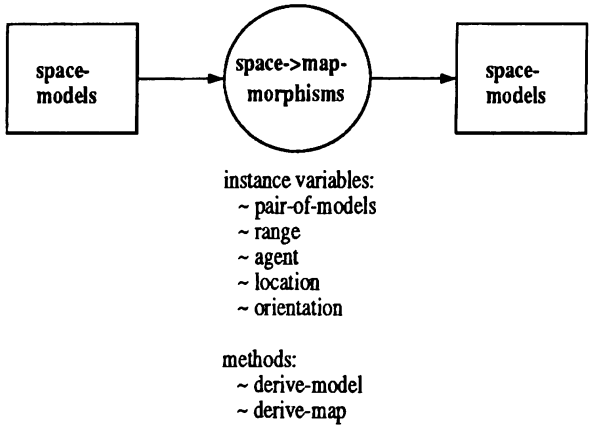


Figure 6: Space to map morphisms class.

Based on the object-oriented programming concepts, we implement the morphism class, *space→map-morphisms* (Figure 6). From such a morphism class we can generate various instances, each mapping one space model to another space model, e.g., from the external model of space to an internal world model within a robot. The homomorphic mappings H_s and H_o are implemented by essentially abstracting the spatial knowledge of the external model of space into a subset within the working range of the agent of the internal world model. The range is held in the instance variable, *range*. The instance variable, *pair-of-models* records the pair of models that are related by the morphism instance. The lumped space model is employed within the agent specified in the instance variable, *agent*. The location and orientation of the agent are held in the instance variables, *location* and *orientation*, respectively. For example, the instance, SPACE-SMM, is constructed in Figure 7. The space model, SPACE-I, is created for its agent located at #(0 20) from the given external model of space, SPACE-E, using the method *derive-model*.

The method *derive-model* not only implements the homomorphic mapping H_s in Figure 5a, but also

```
(mk-ent space-map-morphisms space-smm)
(send space-smm set-pair-of-models '(space-e space-i))
(send space-smm set-agent 'robot1)
(send space-smm set-location #(0 20))
(send space-smm set-orientation #(0 1))
(send space-smm set-range 20)
(send space-smm derive-model)
```

Figure 7: A *space→map-morphisms* specification for models of SPACE.

actually constructs a homomorphic internal space model from the given external space model. The other method, *derive-map* implements the homomorphic mapping H_s only, that is, compute domain within range for existing internal world model from its counterpart external space model.

3.4 Testing The Implementation of Space-models and Morphisms

To ascertain whether our implementation is correct we generate cases to test the preservation of transition and output functions. For this, we use a test macro:

(test ⟨string⟩
  (ap ⟨sequence of operations to external space model⟩) or ⟨Value-1⟩
  (ap ⟨sequence of operations to internal world model⟩) or ⟨Value-2⟩
)

This will print out

⟨string⟩: SATISFIED

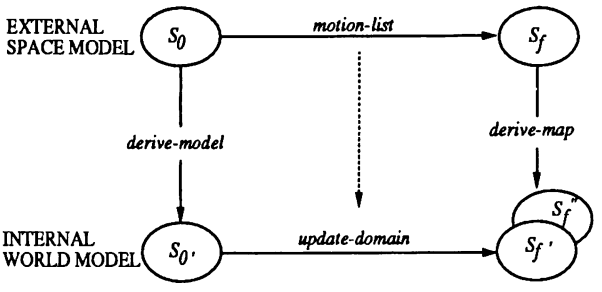
If the 2nd and 3rd arguments turn out to be equal, and

⟨string⟩: NOT SATISFIED

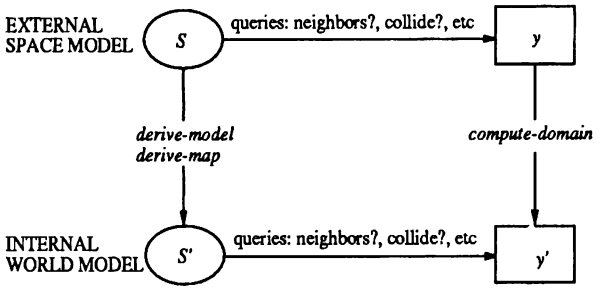
otherwise.

The test macro is essentially intended to test the set equivalence of map entries, thus a *same-set* procedure is employed if the 2nd and 3rd arguments are represented in the form of list; otherwise the primitive *equal?* procedure is used instead. The macro ap (for apply) is used to evaluate a sequence of expressions.

In our modelling approach, the movement of an object results in a message of the object's new location and orientation with identification passing to update the external model of space. The latter records these messages in a *motion-list* that represents the input sequence to the base model we mentioned above. Later on, when a robot consults its internal world model to avoid collisions and to determine neighborhood, the



(a) preservation of transition function



(b) preservation of output function

Figure 8: Implementation of the *space→map-morphisms* Mapping.

internal world model must first update itself, by looking through the *motion-list* of its counterpart external space model for those that are effective within its domain. The space-models method, *update-domain* performs this operation by evaluating the morphism instance specified in the state variable, *pre-image*. This depicts the way we model the corresponding input sequence that is gained from perception of the environment by the robot itself in practice.

Now let's associate the input sequence and methods with the commutative diagrams of Figure 4, the resulting diagrams are shown in Figure 8. In Figure 8, we start to apply the *derive-model* method to an instance of *space→map-morphisms* that maps the external space model in state S_0 to a newly generated lumped internal world model in the corresponding state $S_0\prime$. Then the input sequence *motion-list* brings the external space model from state S_0 to state S_f. On the other hand, the space-models method, *update-domain* derives the corresponding input sequence that brings the internal world model from state $S_0\prime$ to state $S_f\prime$. Meanwhile, we can apply the method *derive-map* to the *space→map-morphisms* instance that maps the state S_f of external space model into a lumped state $S_f\prime\prime$. To test the states $S_f\prime$ and $S_f\prime\prime$ are equivalent, we use the test macro:

```
(test "preservation of transition function"
      map-1
      map-2
)
```

Where map-1 represents the set of map entries of the internal world model in state $S_f\prime$ while map-2 denotes the set of map entries in state $S_f\prime\prime$. The agreement lends support to that $S_f\prime$ and $S_f\prime\prime$ are equivalent. This satisfies the preservation condition of commutative diagram that all paths with the same starting and ending nodes produce identical results.

To test the preservation of output function, we employ the *space-models* method *input?* to interrogate the external model of space and then apply the function *compute-domain* (i.e., *space→map* mapping H_o), the observed output is the same as sending the queries to the robot's internal world model in a corresponding state (Figure 8b). To perform the tests, we use the macro:

```
(test "preservation of output function: query"
      (ap (content-value
            (send space-i input? source query)))
      (ap (compute-domain
            (content-value
              (send space-e input? source query))
            location range))
)
```

where source is the agent of the internal world model and query can be either *neighbors?* or *collide?* for checking neighborhood and detecting collision, respectively.

The test results clearly indicate that our implementation satisfies the preservation conditions. Through the use of *space→map-morphisms*, we can derive a subset of the external model of space for use as an internal world model of an agent. Although the internal and external models of space are updated in different ways, the external model of space remains homomorphic to the internal model of space. This maintains consistency of related models of space.

4 APPLICATION TO MULTI-AGENT AUTONOMOUS ENVIRONMENT

This section shows how our modelling approach and morphisms support world model coherence in multi-agent environment such as the robot-managed space-borne laboratory (Zeigler, Cellier, and Rozenblit 1988). First , we consider the approaches to update the internal world models.

4.1 Update of Internal World Model

In our modelling approach, we have made a strong assumption that sufficient sensory information is available for the agents to identify objects and their movements in the agents' local environment. As we discussed above, an internal world model once generated can be updated in two different ways. One is applying the space-models method, *update-domain* directly to the internal world model; the other is applying the space→map-morphisms method, *derive-map* to the underlying morphism instance. Now the question is: which one is more efficient than the other in terms of the numbers of updated tuples and associated operations? And what are their applicability and limitations?

First, let's consider the case in which a number of objects move around and the agent of internal world model remains stationary. In this case, the *update-domain* method that extracts the spatial changes within certain domain from the external *motion-list* is more efficient because only part of the changes are taken into account, not the total number of objects in the environment. But what happens if the agent itself moves around? There might be some movements of objects before or when the agent moves. Moreover, as the agent travels, some stationary objects are relatively moved in or out of its domain from the viewpoint of the agent. Since these objects are globally stationary, their movements are not in the external *motion-list*. This excludes the use of *update-domain* method. Instead, the *derive-map* method allows the agent to compute its new domain at the origin of the new location.

In summary, to update the internal world model, when the agent is stationary, the space-models *update-domain* method is more efficient than the space→map-morphisms method *derive-map*. On the other hand, when the agent itself moves around, the latter is preferable to the former.

4.2 Multi-agent Organization

Now let's consider a multi-agent organization in which each agent employs an internal world model generated from a *space→map-morphisms* instance. Meanwhile, there is an external space model that maintains the spatial relations of the agents and objects that the agents may operate. (Actually the external space model can be viewed as an internal world model within a super-agent that employs the multi-agent organization. Once again this is an endomorphic agent.) As we mentioned above, when an agent or object moves around, it sends a movement message with its identification, new location and orien-

tation to update the external space model. The latter keeps these messages in a *motion-list*. Later on, other agents can go through their underlying morphism instance to access these information in order to update their own internal world models. From this point of view, the external space model acts as a blackboard for exchanging spatial information. The traveling agents post their movement messages on the blackboard, and then other agents that need spatial knowledge take whatever new and interesting to them. Since there are multiple accesses to the blackboard, extra efforts are needed to ensure the message version control. An additional lot, *version-counter* was introduced to each entry in the world map as well as in the *motion-list* in order to keep track of its update. By checking the version number, an agent can distinguish what messages are new to it. The blackboard approach to exchanging information was typically used for cooperative problem solving in distributed artificial intelligence research (Fennell and Lesser 1977, Lesser and Erman 1980).

For the traveling agent itself, when reaching its destination, the navigation component, i.e. the NAVIGATOR MPU (Zeigler 1990, Zeigler, Cellier, and Rozenblit 1988), must inform the internal world model of the agent's new location so that a new map is derived through the underlying *space→map-morphisms* instance. Thus, at each observation time, the multiple agents, either stationary or just traveled, can maintain coherent internal world models relative to the global spatial relations of the environment.

5 CONCLUSIONS AND FURTHER RESEARCH

Based on the concepts of system morphisms, we have implemented abstraction mechanisms for systematic derivation of related world models, external and internal to an autonomous agent. Such a modelling and abstraction approach has been demonstrated in multi-agent autonomous environment.

However, to enhance the model coherence, we must ensure the consistency of the motion events posted by agents and perceived by others. For example, we need to deal with the situation that there are some other agents in motion at the observation time while an agent is updating and consulting its internal world model. Also, for realistic applications, we need to extend the world model representation to multiple levels of abstraction, such as more local details for gross and fine motion planning.

To achieve a multi-formalism, multi-abstraction model-based systems design, we need to provide more morphisms for model abstractions in various formalisms. Moreover, tools are needed to assess the validity of abstractions relative to the given objectives. Zeigler (1976) presented an approach to this issue using experimental frame concepts. Further research is needed to create such tools.

ACKNOWLEDGMENTS

This research was supported by NASA-Ames Cooperative Agreement No. NCC 2-525, "A Simulation Environment for Laboratory Management by Robot Organization".

REFERENCES

Albus, J. S. 1990a. "A Theory of Intelligent Systems," *Proc. 1990 IEEE Conf. on Intelligent Control*, Philadelphia, PA, pp. 866–875.

Albus, J. S. 1990b. "Hierarchical Interaction Between Sensory Processing and World Modeling," *Proc. 1990 IEEE Conf. on Intelligent Control*, Philadelphia, PA, pp. 53–59.

Albus, J. S. 1990c. "The Role of World Modeling and Value Judgment in Perception," *Proc. 1990 IEEE Conf. on Intelligent Control*, Philadelphia, PA, pp. 154–163.

Antsaklis, P.J., K.M. Passino and S.J. Wang. 1989. "Towards Intelligent Autonomous Control Systems: Architecture and Fundamental Issues," *J. Intelligent and Robotic Systems*, Vol. 1, No. 4, pp. 315–342.

Arkin, R. C. 1989. "Navigation Path Planning for a Vision-based Mobile Robot," *Robotica*, vol. 7, pp. 49–63.

Benjamin, D.P., L. Dosrt, I. Mandyan and M. Rosar. 1990. "An Introduction to the Decomposition of Task Representations in Autonomous Systems," In: *Change of Representation and Inductive Bias*, (ed.: D. Paul Benjamin), Kluwer Academic Publishers, Boston, MA, pp. 125-146.

Fennell, R. D. and V. R. Lesser. 1977. "Parallelism in Artificial Intelligence Problem Solving: A Case Study of Hearsay-II," *IEEE Trans. on Computers*, C-26(2):98–111.

Fishwick, P. A. 1988. "The Role of Process Abstraction in Simulation," *IEEE Trans. Syst. Man and Cybern.*, vol. SMC-18, no. 1, pp. 18–39.

Fishwick, P. A. 1989a. "Abstraction Level Traversal in Hierarchical Modeling," In *Modelling and Simulation Methodology: Knowledge Systems Paradigms* (eds.: B. P. Zeigler, M. Elzas, and T. Ören), Elsevier, North Holland, 1989, pp. 393–429.

Fishwick, P. A. 1989b. "Process Abstraction in Simulation Modeling," In *Artificial Intelligence, Sim-*

ulation, and Modelling (eds.: L. E. Widman, K. Loparo, and N. Nielsen), John Wiley and Sons, 1989, pp. 93–131.

Grant, P. 1990. "Local Path Planning: A Brute Force Approach," *Proc. 1990 IEEE Conf. on Intelligent Control*, Philadelphia, PA, pp. 689–693.

Jokowicz, L. 1989. "Simplification and Abstraction of Kinematic Behaviors," In *Proceedings IJCAI-89*, Detroit, MI, pp. 1337–1342.

Keller, R. M., C. Baudin, Y. Iwasaki, R. Nayak, and K. Tanaka. 1989a. "Compiling Special-Purpose Rules from General-Purpose Device Models," Technical report KSL-89-49, Knowledge Systems Laboratory, Stanford University. June 1989.

Keller, R. M., C. Baudin, Y. Iwasaki, R. Nayak, and K. Tanaka. 1989b. "Compiling Diagnosis Rules and Redesign Plans from a Structure/Behavior Device Model: The details," Technical report KSL-89-50, Knowledge Systems Laboratory, Stanford University, June 1989.

Lesser, V. R. and L. D. Erman. 1980. "Distributed Interpretation: A Model and Experiment," *IEEE Trans. on Computers*, C-29(12):1144–1163.

Luh, C. J. 1991. "Abstraction Morphisms for High Autonomy Systems," PhD Dissertation, Dept. of Electrical and Computer Engineering, University of Arizona, Tucson, AZ, August, 1991.

Luh, C. J. and B. P. Zeigler. 1991. "Abstraction Morphisms for Task Planning and Execution," *Proc. 1991 Conf. on AI, Simulation and Planning in High Autonomy Systems*, Cocoa Beach, Florida, pp.50–59.

McRoberts, M., M. S. Fox, and N. Husain. 1985. "Generating Model Abstraction Scenarios in KBS," In *AI, Graphics, and Simulation: Proceedings of 1985 SCS Multiconference*, SCS Publications, San Diego, CA, pp. 29–33.

Meystel, A. 1988. "Intelligent Control in Robotics," *Journal of Robotic Systems*, vol. 5, no. 4, pp. 269–308.

Murthy, S. S. and S. Addanki. 1987. "PROMPT: An Innovative Design Tool," In *Proceedings AAAI-87*, Seattle, Washington, pp. 637–642.

Murthy, S. S. 1988. "Qualitative Reasoning at Multiple Resolutions," In *Proceedings AAAI-88*, St. Paul, MN, pp. 296–300.

Roth-Tabak, Y. and R. Jain. 1989. "Building an Environment Model Using Depth Information," *Computer*, vol. 22, no. 6, pp. 85–90.

Saridis, G.1983. "Intelligent Robotic Controls," *IEEE Trans. Auto. Control.*, AC-28, No. 5, pp. 547–556.

Sevinc, S. 1990. "Automation of Simplification in Discrete Event Modelling and Simulation," *Int. J. General Systems*, Vol. 18, pp. 125–142.

Unruh, A. and P. S. Rosenbloom. 1989. "Abstraction in Problem Solving and Learning," In *Proceedings IJCAI-89*, Detroit, MI, pp. 681–687.

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

Zeigler, B.P. 1984. *Multifacetted Modelling and Discrete Event Simulation*, London, UK and Orlando, FL, Academic Press.

Zeigler, B. P. 1989. "The DEVS Formalism: Event-based Control for Intelligent Systems," *Proceedings of IEEE*, vol. 77, no. 1, pp. 27–80.

Zeigler, B. P. 1990. *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*, Academic press, San Diego, CA.

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.

Zeigler, B. P., C. J. Luh, and T. G. Kim. 1990. "Model Base Management for Multifacetted Systems," *Proc. AI, Simulation, and Planning in High Autonomy Systems*, IEEE Press, pp. 25–33. (submitted to ACM Trans. on Modelling and Computer Simulation).

Zeigler B.P. and S. D. Chi. 1990. "Model-Based Architectures for Autonomous Systems," *Proc. 1990 IEEE Symp. on Intelligent Control*, Philadelphia, PA, pp. 27–32.

AUTHOR BIOGRAPHIES

CHENG-JYE LUH is a PhD candidate in the Department of Electrical and Computer Engineering at The University of Arizona. His research interests are knowledge-based system design and simulation and artificial intelligence. He is a student member of the IEEE Computer Society.

BERNARD P. ZEIGLER is a professor in the Department of Electrical and Computer Engineering at The University of Arizona. His research interests includes artificial intelligence., distributed simulation, and expert systems for simulation methodology. He is a senior member of IEEE.