Abstract

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 mapping a task plan hierarchy into an isomorphic tree of model abstractions that supports hierarchical task execution. The task plan hierarchy is formulated in a model-based planning approach. We also show how the hierarchical execution structure can be constructed. Then the endomorphism concept employed in the modelling of autonomous systems is illustrated. The DEVS-Scheme knowledge-based, discrete event simulation environment is used to test the models and tools in an autonomous laboratory application.

1 Introduction

A model-based autonomous system employs a multiplicity of models at various control layers to support the predefined system objectives [6]. Such models differ in level of abstraction and in formalism. These partial models, being oriented to specific objectives, are more computationally tractable, more understandable, and easier to develop than comprehensive multipurpose models [2, 9, 11, 12]. Also the multiplicity of abstracted models may provide an evolutionary path for the modelling process [10, 11, 12]. Much recent research has recognized the need for multiple levels of abstraction [13, 14, 15, 16, 17, 18], good representations and ability to change from one representation to another for efficient problem solving [19, 20, 21]. 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 [22] developed a means to support automation of simplification of discrete event models. The simplification tools use observation data from simulation runs of the original model to generate the lumped model that preserves validity in an experimental frame of interest. The simplification approach, based on weakened homomorphism, 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.

In objectives driven modelling methodology [2], 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 [1, 2, 8]. This is depicted in the commutative diagrams shown in Figure 1. In Figure 1a, 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 $S_0/H_s = H_s(S_0)$ 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 $S_f/H_s$, respectively. These states also correspond under $H_s$. In Figure 1b, when the base model is in state $S$ and the lumped model is in corresponding state $S_f = H_s(S)$, the output values observed in these states are $y$ and $y_f$, respectively; the value $y_f$ is also obtained by decoding $y$ under $H_s$, that is $y_f = H_s(y)$. The correctness of abstraction can be tested against the base model by simulation. 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.

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 [3, 4]. 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 [2, 4]. In DEVS-Scheme, the class forward-models, a sub-class of atomic-models, employs a rule-based modelling methodology [4]. It can be used to express the models of robot-operated instruments. Such a model is able to respond to operational commands and diagnostic probes. To further abstract such an external model for use as an internal model within an intelligent agent, table-models, another sub-class of atomic-models, was introduced. This class provides the transition, input, output, and timing information in a form of tuples in a relation, called the transition table. The relative simplicity of the model also facilitates planning, i.e., developing optimal trajectories from initial to goal states. The morphism class forw—table-morphisms [8] realizes the abstraction relationships from class forward-models to class table-models.

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.

This paper describes further efforts to develop abstraction mechanisms for systematic derivation of related models through the use of system morphisms. We develop morphism classes for mapping task plans to model abstractions for hierarchical execution. We also show how the hierarchical execution structure can be constructed. This paper is organized as follows. Section 2 provides an overview on model-based planning and the requirements for model abstraction. In section 3, we present the implementation of a morphism class from pruned entity structures to table-models. We demonstrate the use of the morphism class in a fluid handling example. In section 4, we discuss the endomorphism concept employed in the design of model-based autonomous systems. The testing of morphism correctness is discussed in section 5. Several simulation experiments are presented. Section 6 concludes this paper and considers some directions for future research.

### 2 Model-based Task Planning

Task planning is essentially a search process for selecting and sequencing actions to achieving desired goals [24]. Much AI research has focused on plan generation on simplified abstractions of complex systems in order to be at all practical [23]. However, once plans are derived they should be evaluated in more realistic scenarios before being accepted. An integrated “Planning, Scheduling, Simulation Triumvirate” [26] can combine the capability of AI techniques to generate plans together with the ability of simulation to evaluate them. Moreover, in order to bound the search space and to solve the low-level details of actions, task planning might be extended beyond operational level symbolic goals to include such domain constraints as resources, geometry, etc. and uncertainties [23, 25].

In an effort to develop a model-based task planning system for a space laboratory environment (Figure 2), a
A hierarchical planning approach using the system entity structure concepts has been proposed by Chi et al. [7] to support task planning autonomy at the operational level. The system entity structure knowledge representation scheme is employed to represent the structural knowledge of a system with a family of variant components, as well as to specify the task-related knowledge of action plans in a hierarchical fashion. The task structure hierarchy includes knowledge at three levels of abstraction: HIGH, MEDIUM, and LOW. A task specified at the HIGH level is decomposed into action-oriented subplans at the MEDIUM level, each in turn is refined into LOW level primitive actions. (In principle, such a task hierarchy can have more than one level of subplans.) The planner, called task formulator, generates plans by using the knowledge associated with the task structure such as the available action units and their preconditions and effects, etc. The resulting plans employ a kind of action-ordering representation [24] that directly specifies the relationships among actions. In contrast, the detailed state-based plan structures [24] corresponding to each action unit are specified in the dynamic models which underlie them as we will see later.

The model-based planning approach recognizes that task knowledge is encapsulated in the form of models that are employed at various control layers to support the predefined system objectives. Such models differ in level of abstraction and in formalism. A key requirement is the systematic development and integration of dynamic and symbolic models at different layers.

There are three levels of Model-Plan Units (MPUs) corresponding to the task hierarchy mentioned above. The HIGH level MPU (HMPU) supervises task planning and execution of MEDIUM level MPUs (MM-PUs), each in turn serves the same purpose with respect to the LOW level MPUs (LMPUs). Although these three types of MPUs are placed at the same level in Figure 2, conceptually they reflect the hierarchical decomposition of the task plan being employed.

The LOW level MPUs employ the event-based control paradigm [4] for operations and fault diagnosis. The controller of each unit is attached to a table model of the instrument or elementary action it is responsible for. Such a table model can be generated by an instance of form—table-morphisms which we have already implemented [8]. Through the use of system morphisms, a composition of models at one level can be abstracted into a next higher level model to represent macro state transitions. The MEDIUM level models are derived from compositions of LOW level models. Similarly, the HIGH level models contain abstractions of compositions of MEDIUM level models. The composite model can then be employed by a supervisor of the corresponding lower level controllers. As with the lowest level controllers, the higher level supervisors are to be designed in a generic fashion. Thus a task plan hierarchy will first be mapped recursively into an isomorphic tree of model abstractions. Then a corresponding execution structure will be formed with the assignment of controllers and supervisors, respectively to the leaves and upper level nodes of the model tree.

Figure 2: Entity structure base for an autonomous space-borne laboratory.
3 PES-to-Table Morphisms

Our approach to hierarchical model abstractions is similar to the layered design approach employed in the areas of communications networks such as ISO OSI reference model [27, 28] and computer operating systems [29]. Each layer in the design hierarchy uses the features of lower levels to create new features for use by higher levels. In our approach, the features are extracted from the operational abstractions of models.

The abstraction process is done by aggregating states, time advances, and time windows of lower level table models. Within each table model, the goal-table is prepared by a planner [4] to enable achieving a desired goal state from a given initial state. The morphism class, PES-table-morphisms is developed to do the abstraction process whereby a composition of table models at one level is mapped into a table model at the next higher level. PES stands for pruned entity structure, which is the result of pruning, or extracting, a hierarchical model specification from a system entity structure [4].

3.1 1-level PES-to-Table Mapping

To show how the PES-table-morphisms work, let's consider a FLUID-GETTING task in a space-borne chemical laboratory [5, 7]. To perform such a task, a robot should first get a syringe, then use it to sample some amount of liquid from a storage rack, and finally inject the liquid sampled into a desired container. Thus a FLUID-GETTING task can be decomposed into at least OBJ-GETTING, SAMPLING, and INJECTING subplans as shown in Figure 3.

![Figure 3: SES for fluid-getting task.](image)

Each subplan is refined into several primitive actions. For example, the SAMPLING subplan is composed of actions: move, push, fill, and pull. It says, to carry out the fluid sampling, a robot should act in the following sequences: 1) move itself to the storage site. 2) push the syringe to the storage. 3) fill the liquid into the syringe. 4) pull out the syringe. Note that we have made an assumption that the precondition, a syringe is already in the robot's hand, is satisfied. This action sequence can be abstracted into a global state transition from [at initial position with empty syringe] to [at some storage site with full syringe].

To generate the model structure, the pruning of the laboratory system entity structure is driven by the plan generated in the task formulation [7]. For example, the pruned entity structure P:SAMPLING (Figure 4) with root SAMPLING and children entities, MOVE, PLUG1, SYRIN and PLUG2 that refer to the models used to carry out the actions, move, push, fill and pull, respectively. We enlarge the table model concept to include models of actions such as MOVE, which represent the relevant time and state changes brought about in the agent and its environment by executing an action.

To derive a table model corresponding to a MEDIUM level entity such as SAMPLING, the pruned entity structure must have assigned initial and goal states for each subordinate entity as in Figure 4. As the abstraction proceeds, each LOW level table model is abstracted into a two-step transition starting from its given initial state, through an active state, to its goal state. The first transition is caused by an external event (command) that switches from its initial state to an active state for a period of time. Then as the active time is elapsed, the state is changed to its goal state.

![Figure 4: Mapping sampling PES to table models.](image)

1Indexing is used to distinguish the different observation times whenever a model is activated several times in a task order.
state. The active time is obtained by examining the
goal-table of the table model. A name for the ac-
tive phase is obtained by appending "ing" to the ini-
tial command given in the goal-table. As illustrated in
Figure 5, the move action is activated by external com-
mand 'move'. Its abstraction immediately switches its
state from 'passive' to 'moving'. After holding in state
'moving for an interval of travel-time, the move action
reaches the desired goal state, 'arrived and expects a
touch-sensor. Note that a zero time pseudo internal
transition (indicated by dotted arrow) is introduced
as a link between the final state of a transition and
the initial state of its successor. At the LOW level,
any missing table models can be derived through the
forw-table-morphisms [8]. The transition time, and
its window of the resulting table model are obtained by
summing the transition times and windows of individual
steps recorded in the goal-table of the table model.
An instance of the class PES-table-morphisms maps
the entities of P-SAMPLING into the corresponding
table models as shown in Figure 4.

3.2 N-level PES-to-Table Mapping

The class PES-table-morphisms derives the table
models for high level tasks such as FLUID-GETTING
as well. The abstraction mechanism will look for the
table models corresponding to the subtasks, e.g., OBJ-
GETTING, SAMPLING, and INJECTING, respectively. If all the table models of the subordinates exist,
then the PES-table mapping will be applied to inte-
grate the operational information extracted from each
one. Otherwise, subgoals will be set up to create miss-
ing table models from their subordinates. At the leaves,
any missing table models can be derived through the
forw-table-morphisms.

The derived table models are saved in the model base
for further reuse. The date of creation of the model
files is checked to see if the table model is still valid for
any model that has been redefined. If it is out of date then
the table model is regenerated. Once again this
contributes to model base coherence.

3.3 Model-Based Task Execution Hier-
archy

With the hierarchical abstraction, two classes of mor-
phisms have been employed. The first one is the
forw-table-morphisms at the lowest level. The second
one is PES-table-morphisms for mapping the entities
of a task hierarchy tree into their corresponding table
models. As illustrated in the FLUID-GETTING exam-
ple, an instance of PES-table-morphisms is generated
for 2-level PES to table mapping (Figure 6).

As just described, we can systematically generate ta-
ble models for the nodes of a task decomposition tree.
Also as indicated, to execute the task hierarchy, we as-
sign an appropriate generic engine to each table model in the resulting model hierarchy. There are two types of generic engines needed: a) for the leaves, these are the controllers that we have already developed, and b) the supervisors at the higher levels. To design the supervisors, two types of messages transmitted between levels are envisioned: goal (command) and completion (response) messages. Goal messages conveying the state that a subordinate is to attempt to reach are sent top down, whereas, completion messages are gathered in bottom-up manner. This hierarchical control structure is similar to that proposed by [30] except that the controllers and supervisors are generic interpreters of instrument models. They generate goal and completion messages by querying associated table model abstractions. For example, the table model provides the time window in which a completion message from a subordinate is expected.

Such a hierarchical control structure (Figure 7) provides local control when possible and global coordination as needed. Task execution is supervised at different levels of decomposition and abstraction, and such distributed control facilitates local faults diagnosis. Also task scenarios can be simulated and verified at certain levels without the flood of lower level details [7].

4 Endomorphism

A model always stands in relation to a thing it represents, the modellee or model counterpart. A high autonomy system may have models for concrete physical objects or for systems and subsystems whose boundaries are more arbitrary. The modelled objects or systems may also be included within the boundary defining the system itself.

The model base of a high autonomy system may contain models of:

- objects, external to it, such as obstacles it may encounter, pieces of equipment it may operate, etc., or internal to it, such as of its physical parts, needed for example, to execute repair replacement procedures.
- collections of objects, such as a world map containing the spatial locations of all objects
- systems (and subsystems), external, such as chemical processes it is trying to control, or internal subsystems such as its motion subsystem, whose time characteristics need to be known for navigation.

Zeigler [4] used the term endomorphy to refer to objects (systems, models, agents) in which some sub-objects use models of other sub-objects. An endomorphic agent incorporates, uses (and possibly constructs) models of (parts of) itself in its decision making.

An example of endomorphism, is the morphism between the MOTION system in Figure 2 and the table models, such as MOVE-O, PLUG-O, etc., employed in planning and control. Here a ROBOT's cognition system contains models of the ROBOT own MOTION subsystem.

Also the PES→table-morphisms maps a composition of table models at one level into a table model at the next higher level. Since the latter model is an abstraction of internal models (which are objects), this is another manifestation of endomorphy.

5 Testing the Abstraction Hierarchy

Task planning abstraction hierarchies allow task execution to skip over lower-level details and to be supervised at higher level of abstractions. However, in actual simulation, the status of the low-level dynamic models
must be updated to reflect actual execution. Thus, to employ higher level abstraction for faster simulation (rather than execution control) more information on low level state variable changes must be retained by the morphism.

In DEVS-Scheme atomic-models specification, the value of a state variable is changed by the set! operator, e.g., (set! (state-level s) 100). In addition, the state variables phase and sigma can be set by one of the three macros: hold-in, passivate-in, and passivate.

In forward-models specification, the action slot of an activity specifies a change in the state of the model. During the forw—table morphism mapping, the set! clauses in the action slot of a reachable activity are analyzed and processed into a list of attribute-value pairs, each with a state variable as its attribute. The result is held in the effects slot of a tuple in the transition table of the resulting table model. All of the effects are accumulated in goal planning. For multiply occurring state variables, the most recent effects always overwrite previous ones. For higher level abstraction, the underlying forward models in which the effects work are also specified so that the effects can be evaluated in appropriate environments. This is necessary since the values of state variables might be specified as procedures rather than explicitly.

The correctness of our abstraction morphisms is tested in several ways. Where possible we attempt formal proofs that a simplification procedure produces a model in a valid morphic relationship to the original. This follows the lines of [2]. Direct full-scale simulation is also used to empirically verify the correctness of the hierarchical abstraction.

5.1 The Test Environment

The simulation experiments focus on the interactions between the models, external and internal to a robot’s cognition system in the context of robot-managed fluid handling application. A full scale experiment is obtained by pruning the SES of Figure 2. Such experiments have been performed successfully.

However, a reduced test environment (Figure 8) was designed to permit less time consuming trials. The environment will enable testing a group of task-oriented table models supervised by a task orderer vis-a-vis a corresponding group of external models. The external models represent models of instruments and elementary actions and each internal model is generated by an instance of forw—table-morphism to interact with its corresponding external model. A single external—internal model pair can be simulated to test lowest level table models. In simulation, the table model controller will send to, and receive messages from, the corresponding external instrument or action model. If the operation completes normally this indicates that the homomorphic relation expressed in forw—table-morphism, was maintained between the table model and the controlled object expressed in forward-models specification. Such simulation results were reported in [8].

Within the test environment, the entities that refer to the models of instruments, such as SYRIN, and elementary actions, such as GET, MOVE, PLUG, are generalized and collected together as specializations in E:ACTION. Each entity in turn is specialized into -E and -O entities that refer to the underlying external and operational models, respectively. Note that the motion-related external models, GET-E, MOVE-E, and PLUG-E are parts of the robot’s MOTION system. The entity, TASK, represents the operational portion of the low level TASK MPU in Figure 2. By using context sensitive pruning associated with selection constraints [4], we can generate a coupled model containing EXTERNAL and INTERNAL configurations consisting of a sets of models in one-to-one correspondence. The INTERNAL configuration is a reduced version of a ROBOT’s BRAIN.
5.2 Simulation of the Task Hierarchy

Using such a test environment, we are able to test whether the PES→table-morphisms class produces correct table models. In simulation, the task orderer will cause the operational models be deployed in a sequence corresponding to the leaves of a task plan hierarchy tree. As each operational model is deployed, its controller will interact with the corresponding external instrument or action model. The final state reached and time needed to reach it, will be compared with the corresponding state and completion time window predicted by the root table model. The table model abstraction will survive a test if the states match and the actual completion time falls within the predicted time window.

Now consider testing the PES→table-morphisms for I-level PES such as SAMPLING (Figure 4). Before starting a simulation run, we save the initial settings of the state variables of each low-level dynamic model, for example, MOVE-E, PLUG-E, and SYRIN-E for SAMPLING. Then task sequence given in the task orderer is executed. When the simulation run ends, the final settings of the state variables of each dynamic model are compared with the saved initial settings. The differences are the state variable changes caused by the execution. On the other hand, we can employ the table model, SAMPLING-0 and directly evaluate the execution effects obtained from the goal-table in the environments of the underlying forward models. As an example, the robot initially at location #(0 0) samples some amount of liquid from storage located at #(30 40). The directly evaluated and simulated transition time, time window, and execution effects are listed in Figure 9. The simulated completion time for the sequence, 69.1, properly falls within the time window obtained by direct evaluation from the goal-table of the table model, FLUID-GETTING-0. The state variable changes also agreed with the result of directly evaluating the execution effects provided by the goal-table. Thus, the table model, FLUID-GETTING-0, is a valid operational abstraction of the FLUID-GETTING task by a 2-level PES→table homomorphic mapping marked M1 in Figure 10.

Now the table model, FLUID-GETTING-0, is abstracted from its subordinates, OBJ-GETTING-O, SAMPLING-O, and INJECTING-O by applying the 1-level PES→table mapping marked M2 in Figure 10. It is this mapping that we are actually testing. To see this, consider that we already know that OBJ-GETTING-O, SAMPLING-O, and INJECTING-O are correct 1-level PES→table morphism mapping images for OBJ-GETTING, SAMPLING, and INJECTING subtasks shown as M3a, M3b, M3c in Figure 10, respectively. Now morphism mapping M1 is a composition of morphism mappings M3a, M3b, M3c and M2. Since M1 and M3a, M3b, M3c are correct, this implies that M2 is correct also.

For arbitrary n-level abstraction hierarchy, we can directly test whether the n-level PES→table morphism mapping is correct. We also can test the n-1-level PES→table morphism mappings for the all sub PESs from level n to level 1 (the lowest level). If all the above morphism mappings are correct, then so is the 1-level PES→table morphism mapping for the top 1-level PES from root entity to entities at level n-1.
6 Conclusions and Further Research

In this paper, we have implemented abstraction mechanisms for systematic derivation of task abstraction hierarchies that support hierarchical task execution at different levels of abstraction. Such abstraction mechanisms employ the concepts of system morphisms. The morphism class, PES→table-morphisms, has been developed to do the abstraction process that systematically generates table models for the nodes of a task decomposition tree represented in pruned entity structure. 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.

To achieve a multi-formalism, multi-abstraction model-based systems design, we need to provide more morphisms for model abstractions in various formalisms. For example, space→map morphisms for abstracting spatial relations and device→diagnosis morphisms for deriving diagnostic models from device structures. Moreover, tools are needed to assess the validity of abstractions relative to the given objectives. Zeigler [1] presented an approach to this issue using experimental frame concepts. Further research is needed to create such tools.

Acknowledgements

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

References


