McRoberts M., M.S. Fox, N. Husain, (1985), "Generating Model Abstraction Scenarios in KBS", Proceedings of Society for Computer Simulation MultiConference, Simulation Councils Inc: La Jolla, CA, pp. 29-33.

Generating model abstraction scenarios in KBS

Malcolm McRoberts
Mark Fox
Nizwer Husain

Intelligent Systems Laboratory
Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15217

Abstract

The purpose of this document is to describe a mechanism by which simulation models can be altered to execute at different levels of abstraction. These alterations can be performed automatically provided the model builder has created a knowledge framework for the task. Although most of the concepts used are fairly general, the examples used refer to models developed in the Knowledge Based Simulation (KBS) environment.

1. Introduction

All models are abstract in some sense because they stop at some level of detail and make generalizing assumptions. For instance in computing transportation time in a distribution model we may not have detailed knowledge about traffic, weather or vehicle capabilities all of which affect transportation time. Also there is always a boundary drawn between the system being simulated and that system's environment, since the whole universe cannot be simulated. Therefore anyone who builds simulation models must use some abstraction techniques.

The rationale for our investigation of model abstraction is somewhat different due to the style of simulation supported by the KBS system [1] [2]. KBS is a Lisp-based discrete simulation system in which models are constructed using SRL [3], a frame-based knowledge representation language. KBS is designed to be used interactively, enabling the user to examine the model and its behavior. We have found that the users of this system tend to construct very detailed models which take too long to run. Users are unwilling to sit through such sessions. We have also observed that users tend to focus on only a portion of the model. If it were possible to abstract the portion of the model not under investigation and thereby reduce the complexity of simulation, then better performance would be provided to the user. Our goal then is to allow the user to construct models which are very large and rich in detail and derive automatically a model which is a subset of the original but still contains the objects and behaviors which the user wishes to study.

Model simplification techniques fall into two main categories, static and dynamic. In static techniques both model structure and model parameters are altered; however, the event behaviors remain unchanged. Since only static aspects of the model are affected these are called static techniques. Dynamic techniques on the other hand alter a model's dynamic processes. This means redefining some of the event behaviors.

Previous research in automatic model abstraction have focused on dynamic behavior [4]. The concept of "coalescing micro-events into more aggregated macro-events", has been investigated. The difficulty in the techniques used lie in detecting which pattern of events are significant enough to abstract. Innis [5] takes a different approach to model abstraction. The primary motivations for simplification are to make models more understandable. A broad range of techniques are described including, replacement of slowly changing variables by constants, use of sensitivity analysis to suggest simpler models, re-coding or outright conversion to a different language, and replacement of parts of the model using analytic solutions. Most of these simplification methods cannot be automated easily.

The work described in this paper investigates both static and dynamic abstraction. A knowledge-based approach is utilized. Knowledge about schema models and how they may be abstracted is the basis of static abstraction. Dynamic abstraction focuses on single and multiple object abstraction through the observation and combining of object stimulus-response behavior.

After various concepts in model abstraction have been designed and implemented, these implementations must pass certain tests in order to be judged sucessful. These tests come in three levels and their sucess criteria are explained:

Correctness	Given a detailed model which is free from runtime errors during execution, and given sufficient knowledge from the model builder, selecting an abstraction will result in a model which is also free from runtime errors.
Accuracy	It follows from our definition of an abstract model that the detailed model may produce some results which are not available at the abstract level. Performance measures based on sufficient periods of time are not significantly affected by selecting an abstraction except where they are no longer meaningful.
Speed	For real world size problems selecting an abstraction improves the speed with which a simulation runs.

2. Problem Description and KBS

The examples in this paper refer to a corporate distribution model, currently under development [6]. The example corporation is concerned with manufacture and distribution of a variety of products. Individual components and complete systems are either manufactured in the corporation's own facilities or supplied by outside vendors and are stored in regional distribution centers. Actual sale of components and systems is performed by retailers or company owned stores whose inventory is replenished by the distribution centers under the control of a business unit. The corporate distribution group within the corporation is responsible for the design of the distribution network as well as operational planning which is responsive to the overall corporate goals. There are also references to a queueing model of a workshop consisting of machines which can perform operations on work-pieces.

The distribution model is implemented as schemata using SRL and KBS. KBS is a Knowledge-Based Simulation system, incorporating the discrete event simulation approach. All entities in KBS are represented as SRL schemata. SRL, an AI-based knowledge representation system for modelling, incorporates inheritance along relations. A model in KBS (eg. factory model) is a network of SRL schemata. Running the simulation then consists of dragging this network through time, event by event. KBS provides facilities for interactive model creation and alteration, simulation monitoring and control, graphics display, and selective instrumentation. It also allows the user to define and simulate a system at different levels of abstraction, and to check the completeness and consistency of a model, hence reducing model debugging time.

3. Static Abstraction

3.1. Equivalent Node Aggregation

The idea of equivalent node aggregation is to combine several nodes into a new node of the same type (type meaning the schema from which they are instantiated e.g. "store"). All the links need to be combined also. This new node must be in some sense the sum of the original nodes. This will result in information that is unique to the individual nodes being lost but if the new node's parameters are adjusted correctly then the capacity of the new node is the same as the old group. In this way the rest of the model should be affected very little. A general example of this (see figure 3-1) is a queueing system where one queue has 3 servers all of which flow into the same queue after finishing. It is possible to replace these 3 servers with one server that has 3 times the capacity without changing the amount of flow.

A fairly general schema definition follows:

```
{{equivalent-node-aggregation
    IS-A: model-abstraction
    TYPE:
        comment: "type of node to aggregate"
    SUPER-NODE:
        comment: "single node replaces sub-nodes when
            abstract"
    SUB-NODES:
        comment: "set of nodes present at detail level"
    REFERENCES:
        comment: "a reference is a schema-slot pair where
            references to sub-nodes are found"
    SUM-SLOTS:
        comment: "the value of these slots in the aggregate
            is the sum of the values in the sub-nodes"
    AVERAGE-SLOTS:
        comment: "the value of these slots in the aggregate
            is the average of the values in the sub-nodes"
    UNION-SLOTS:
        comment: "each value is a list of a slot in the aggregate,
            whose values will be the union of the sub-node
            values and the name of the type of aggregate
            schema to use to aggregate the meta values from
            the sub-nodes. If no aggregate type is given
            then no aggregation is performed on meta values."
    FUNCTION-SLOTS:
        comment: "each value is a list of a slot in the aggregate,
            and a function for filling that slot from the values
            of the same slot in the sub nodes."
    GENERATED-FROM:
        COMMENT: "if this aggregate was recursively generated
            as a result of another aggregation then this slot
            contains the name of the parent aggregation"
    SELECT: (select-node-aggregation)
    DESELECT: (deselect-node-aggregation)
}}
```

Consider a distribution system where stores are each serviced by a small regional distribution center called a stocking point. In such a system stores can be aggregated by stocking point. Since there are many stores and few stocking points, and since each store is only supplied from one stocking point, it makes sense to group together stores which share a common stocking point. This can be accomplished simply by creating these aggregate stores, generating aggregate demand and replacing the old stores with the aggregate during the scheduling of initial demand events.

3.2. Data Class Aggregation

In data class aggregation objects are grouped into classes and all references to the members are replaced by references to the class. This of course throws away information about variations among the class members but it should increase the overall efficiency and not greatly affect model execution.

An example of this concept as it can be applied to computer equipment distribution model would be grouping all items sold into classes. For instance all types of system boxes can be combined into one. The demand for all these should then be added together. Other types of equipment can be identified such as printers, terminals etc. This would greatly reduce the amount of data needed to track inventory levels in the model.

```
{{store-by-stocking-point
    IS-A: "equivalent-node-aggregation"
    TYPE: "store"
    AVERAGE-SLOTS: "lost-sale-percentage"
    UNION-SLOTS: ("order-weeks" "aggregate-weekly-demand")
            ("inventory" "aggregate-inventory")
            ("operating-days")
}}
```

```
{{aggregate-weekly-demand
    IS-A: "equivalent-node-aggregation"
    TYPE: "weekly-demand"
    UNION-SLOTS: ("order-days" "aggregate-daily-demand")
}}
```

```
{{aggregate-daily-demand
    IS-A: "equivalent-node-aggregation"
    TYPE: "daily-demand"
    UNION-SLOTS: ("bulk-order" "aggregate-bulk-order")
}}
```

```
{{aggregate-bulk-order
    IS-A: "equivalent-node-aggregation"
    TYPE: "bulk-order"
    UNION-SLOTS: ("orders")
}}
```

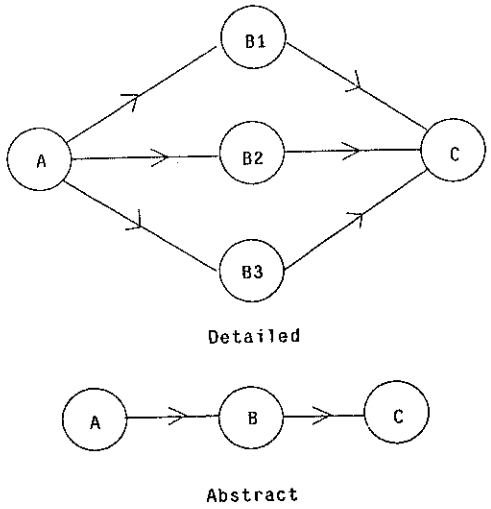

Figure 3-1: Equivalent Node Aggregation

4. Dynamic Abstraction

Dynamic abstraction attempts to simplify simulation models by analyzing the stimulus-response event behavior of one or more nodes, and constructing a single node with a stimulus-response behavior which is statistically similar. This technique can be used to combine nodes of the same or different classes.

In order for the abstraction of a node or group of nodes to work, that node or group of nodes must be *isolatable* from the rest of the model. Each node in a simulation has certain state variables within it. These can range from mode of operation to amount of material being stored, etc. A node or node group is isolatable if no information about its state variables is obtained by an event occurring outside the group except from its event parameters, and if no event occurring outside the group changes any of the

```
{{aggregate-inventory
    IS-A: "equivalent-node-aggregation"
    TYPE: "inventory"
    AVERAGE-SLOTS: "reorder-interval"
    SUM-SLOTS: "reorder-level" "initial-inventory"
}}
```

```
{{store-by-stocking-point-1
    INSTANCE: "store-by-stocking-point"
    SUPER-NODE: "aggr-store-1"
    SUB-NODES: "store-1" "store-2"
}}
```

group's state variables.

This section first describes how behavior is captured in KBS and follows with a discussion of issues around their abstraction.

4.1. Recording Dynamic Information (S-R-frames)

In order to perform dynamic abstraction it is first necessary to record the model's dynamic behavior over some time period. Although this information can be used in various ways, it can still be stored in a uniform way. The information that needs to be recorded is the relationship between a permanent object, the events which occur there and the new events those events schedule. This information is stored in stimulus-response-frames [7] (S-R-frames) and these S-R-frames are in turn stored in the "S-R-frames" facet of the object's event slot.

Using the example above, when the event "kbs-event-1" is executed it is seen to be a "load" event at "drill-1". At this time a new "S-R-frame" is created whose enabling event is "kbs-event-1" and it is stored in the "S-R-frames" facet of the "load" slot in "drill-1". When the contents of the "load" slot are evaluated an "unload" event is scheduled at "drill-1". The event scheduler creates the event schema "kbs-event-2" which is

```
{{S-R-frame
    ENABLING-EVENT:
        range: (type "instance" "event")
    CAUSED-EVENTS:
        range: (type "instance" "event")
}}
```

```
{{drill-1
    INSTANCE: "drill"
    .
    .
    .
    LOAD: "drill-load-rule"
        S-R-frames: "S-R-frame-1" "S-R-frame-2"
            range: (type "instance" "S-R-frame")
}}
```

```
{{S-R-frame-1
    INSTANCE: "S-R-frame"
    ENABLING-EVENT: "kbs-event-1"
    CAUSED-EVENTS: "kbs-event-2" "kbs-event-3"
}}
```

stored in the "caused-events" slot of the current "S-R-frame". If the only input and output to a node is taken to be the events that take place there and the new events those schedule, then these "S-R-frame"s provide all the information needed to abstract a node's behavior.

```
{{kbs-event-1
    INSTANCE: "kbs-event"
    EVENT-SCHEMA: "drill-1"
    EVENT-SLOT: "load"
    EVENT-TIME: 10:17am April 5, '79
    PARAMETERS: "part-1"
}}

{{kbs-event-2
    INSTANCE: "kbs-event"
    EVENT-SCHEMA: "drill-1"
    EVENT-SLOT: "unload"
    EVENT-TIME: 11:15am April 5, '79
    PARAMETERS:
}}
```

4.2. Abstracting Dependence

It has been suggested that the dependence between events can be modeled stochastically [8, 4]. That is the proportion of the times that an event occurs in which it causes another event is the likelihood of the former event causing the latter. This is illustrated in figure 4-1. Whenever event A occurs, either event B is caused, or both events B and C. Hence, for each occurence of event A, B occurs 100% of the time and C occurs 50% of the time. This information can then be used to create macro events that abstract an entire subsystem (see figure 4-2).

This abstract model fails to provide a representation for the distribution of the intra-event times, and the statistical relationship between the two event's parameters. That is, the modeling of transitions only, and not the parameter is insufficient. For example, in the distribution domain it is insufficient to know that an order at a retail store may result in the store ordering more parts from the distributor. Information about the parts and the size of the order may also be significant. We are not currently attempting to apply this technique to KBS models for these same reasons.

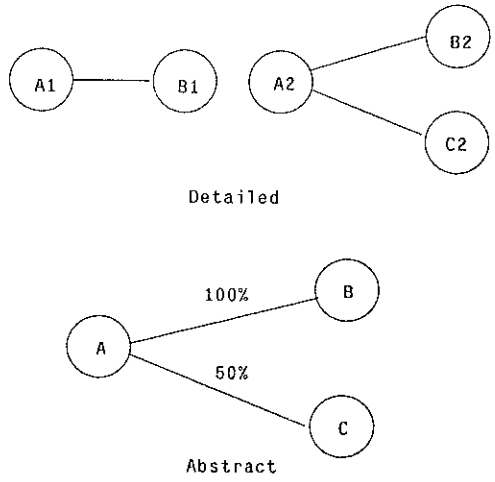

Figure 4-1: Stochastic Event Abstraction

4.3. Independence Assumption

In order to facilitate the task of abstracting the behavior of a node or a set of nodes it is convenient to assume that the output of this group is independent of the input to the group. Of course this statement is rarely valid in any simulation model; however if we can show that the input to the group will be constant over the different scenarios we wish to compare then we can still ignore input. What this means from a practical standpoint is that an event map will be created that maps all events coming from outside the group to nothing.

It is also true that if this independence assumption holds true,

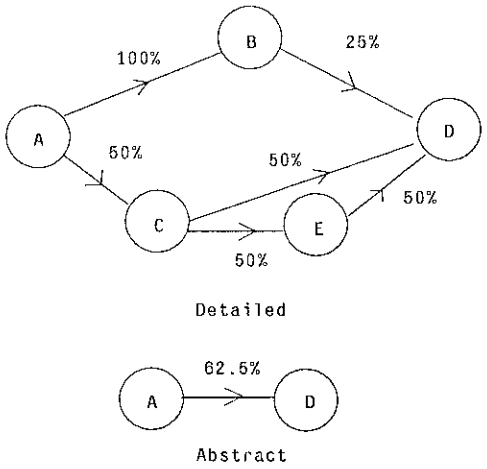

Figure 4-2: Abstracting a Subsystem

then the output of the group can be modeled as a distribution depending only on time. It follows that there must be a set of sufficient statistics from which an equivalent set of output events could be generated. All the information needed to compute these statistics can be obtained by analyzing the S-R-frames for the output events. In practice however it is very difficult to find these statistics since the event space is so large and complex.

4.4. Modeling by Flow

Instead of trying to model output by means of statistical generation it is possible to model output based on rate of flow. Each event represents some flow out of the group whether it be a message or physical material. If the rate of this flow can be captured then these events can be generated on a regular basis so that the total number of messages, parts etc is the same over a long period of time. In the illustration 4-3 the group consisting of C D E F, can be replaced by a sink at G and a generator at H so that the flow rates at A and B are unaffected.

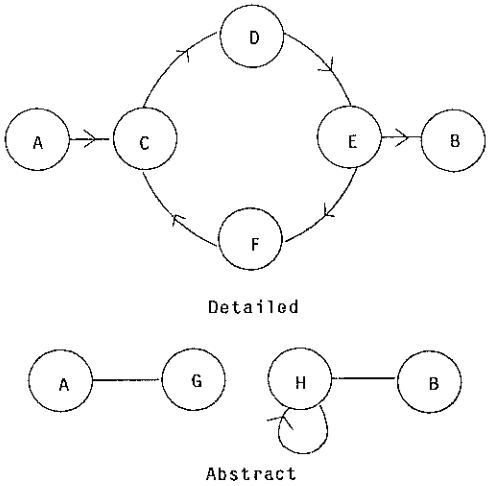

Figure 4-3: Modeling by Flow

The information about the rate of flow is usually contained in the output event's parameters. Of course the information can be buried deep within the parameters and it can be complicated to generate synthetic ones.

The data in the parameters must be summed over all the sources (nodes within the group) and over time. The data must also be differentiated by different destinations (nodes outside the group) and by different types of material or messages.

An example would be aggregating all the orders from resellers on a weekly basis. First the S-R-frames would be collected for many simulation weeks. Then all the "order-receipt-events" which were scheduled are summed in the following fashion. Since they all have the same focus (destination) there will only be one order-receipt-event per week. The parameter for the order-receipt-event is a bulk-order. A bulk order contains a list of orders each of which contains a list of line items. A line item consists of a part number and a quantity. For each part number the total quantity of that part from each order from each reseller over the entire time period must be found. This divided by the number of weeks in the data collection period yields the average weekly quantity ordered of that part. It is now possible to generate a new order-receipt-event each week which has as its parameter a bulk order which contains one order with a line item for each part number whose quantity is that part number's weekly average.

Of course such events may occur on any schedule hourly, daily, monthly or even yearly depending on the nature of the simulation.

4.5. Behavioral Abstraction

Behavioral abstraction is the most hazy of the various abstraction concepts. It involves making modifications in the actual event behaviors. It also may involve changing the format of the event parameters. The goal is to simplify the models micro behavior so that one can run more efficiently and still study macro problems. The number and type of the nodes does not change but the events and their parameters may be redefined.

An example of a behavioral abstraction in a distribution model would be to ignore the concept of individual orders. At present several orders of various sizes are placed at a reseller every day. In the current implementation these orders are filled individually. The event behaviors could be modified so that only the total number of each part matters not which order it is in.

5. Model Management

KBS provides the user with the ability to specify and select model abstractions. Schemata define the nature of the model abstraction to be done and those the user selects will be interpreted as a series of actions which alter the model. The general schema follows.

{{**model-abstraction**
STATUS:
range: (or 'abstract 'detailed)
comment: "indicates whether abstraction of this type is currently being used in model simulation"
SELECT:
comment: "this slot defines the selection process for the schema. Contents are slot-evaled at selection time"
DESELECT:
comment: "this slot defines the deselection process for the schema"
}}

Each subclass of model abstraction has its own slots, but they all have in common a status slot which indicates whether the aspect of the model which the schema governs is to be simulated at the abstract or detailed level.

In order to obtain a profile of the version of the model which is being run it is necessary to know which model abstractions have been selected. Therefore two slots are added to the "KBS-model" schemata: "abstractions-available" which lists all the abstraction concepts that the user can select; and "abstractions-selected" which lists those that actually have been selected.

Selecting a model abstraction does the following: changes the model abstraction's status from detailed to abstract; adds the selected abstraction to the "abstractions-selected" slot of the model; and evaluates the abstraction schema's "select" slot, which will contain the function which uses knowledge in the model abstraction schema to perform the model alterations needed to implement the abstraction concept.

```
{{KBS-model
    ABSTRACTIONS-AVAILABLE:
        range: (type "instance" "model-abstraction")
    ABSTRACTIONS-SELECTED:
        range: (type "instance" "model-abstraction")
    .
    .
    .
}}
```

Interpreting an abstraction schema's "select" slot may result in several things. The model's structure may be altered in some way, the definitions of some events may be changed and an event map may be set up. The effect of using an event map is to trap events as they are scheduled and to instead schedule the event that the original event maps into. Events may map to nothing which prevents their being scheduled or they may map into events that are the same except for their focus.

6. Results

As of this date the equivalent node aggregation concept has been implemented and is being tested with a small corporate distribution model. This model contains one outside vendor, two company owned manufacturers, one distribution center, one stocking point, two company owned retail stores, one other store, and 13 transportation lanes. This makes a total of 15 facilites connected by 13 lanes. The aggregation described in the examples above, in which retail stores are aggregated was performed resulting in a new model with one super store replacing the two company owned stores. This new model had 14 nodes instead of 15.

For this example equivalent node aggregation meets the testing criteria described in the introduction. The new model was run for two weeks of simulation time and no errors occured and the model statistics were the same for both models. The number of events executed in two weeks was reduced from 885 to 848, a reduction of 4% and the total process time was reduced from 62.4 min. to 60 min. also a 4% reduction. While these improvements are small they do suggest that this technique may be of real use on real world sized models with many stores.

As for dynamic abstraction, we are able to collect stimulus-response but methods for using the information they contain to abstract a system's dynamic behavior are still in the design stages.

References

1. Mark S. Fox and Y.V.Reddy, "Knowledge Representation in Organization Modeling and Simulation: Definition and Interpretation," *Proceedings of the Thirteenth Annual Pittsburgh Conference on Modeling and Simulation*, April 1982.

2. Y.V.Reddy and Mark S. Fox, "Knowledge Representation in Organization Modeling and Simulation: A detailed Example," *Proceedings of the Thirteenth Annual Pittsburgh Conference on Modeling and Simulation*, April 1982.

3. Wright,J.M. and Fox,M.S., "SRL/1.5 User Manual," Tech. report, Carnegie-Mellon University, 1983, Intelligent Systems Laboratory,The Robotics Institute, Pittsburgh,PA.

4. Sarah E. Goldin, Philip Klahr, "Learning and Abstraction in Simulation," *IJCAI*, 1981, pp. 212.

5. George Innis, Eric Rexstad, "Simulation model simplification techniques," *Simulation*, July 1983, pp. 7.

6. Y.V.Reddy, Mark S.Fox, Kevin Doyle, John Arnold, "INET: A Knowledge Based Simulation Model of a Corporate Distribution System," *Proceedings of IEEE Conference on Trends and Applications*, May 1983, .

7. Hayes-Roth F., V.R. Lesser, "Focus of attention in a distributed logic speech understanding system.," *Proceedings of the 1976 I.E.E.E. International Conference on Acoustics, Speach and Signal Processing*, 1976, pp. 416-420.

8. Mark S Fox and Frederick Hayes-Roth, "Approximation Techniques for the learning of sequential symbolic patterns," *Proceedings of the Third International Joint Conference on Pattern Recognition*, November 1976, pp. 616.