Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

CHARACTERIZATIONS AND RELATIONSHIPS OF WORLD VIEWS

C. Michael Overstreet

Computer Science Department
Old Dominion University
Norfolk, VA 23529-0162 U.S.A.

Richard E. Nance

Systems Research Center
Orca Computer, Inc
Virginia Tech Corporate Research Park
Blacksburg, VA 24061 U.S.A.

ABSTRACT

We describe a characterization the three classical world views of event scheduling, activity scanning, and process interaction and discuss transformations among them. We believe that one advantage of each is to allow more concise model descriptions by allowing a model specifier to take advantage of contextual information. Automated transformation among world views is difficult due to a modeler's use of contextual information. We illustrate this by transforming and then simplifying a model representation creating a version, similar to what a programmer or modeler might generate.

1 INTRODUCTION

A fundamental sub-area of modeling methodology is the world view or conceptual framework adopted or imposed in the model development process. Recognition of the importance of a world view or "Weltansicht" is traced to the early days of discrete event simulation (DES) (Lackner 1962). Much of the early attention to world views emerges in the development of simulation programming languages (SPLs) and simulators (packages in a general purpose language (GPL)). The need for model specification independent of an implementation in an SPL is expressed in the early papers of Lackner (1962, 1964). Kiviat (1969) and Lackner are credited with the definition of the three "classical" world views: event scheduling, activity scanning, and process interaction. The characterization of a world view in terms of its embedding in an SPL is described by Kiviat (1969) as an inversion of theory and application that clearly merits the attention of the simulation community.

Zeigler (1972, 1976) uses systems theory as a foundation for explaining DES, and a "formal" representation of model behavior is given in a notation similar to state transition diagrams. Nance (1979) approaches model specification as an issue in the development of model documentation standards, drawing attention to research in program generators, the Conical Methodology, and the DELTA Project (Holbaek-Hanssen, Hfindlykken and Kygaard 1977). The significance of time and state relationships that form the core of any DES model specification is addressed by Nance (1981) within the context of contrasting world view perspectives.

The Condition Specification (CS), created by Overstreet (1982), seeks to enable an algorithmic (automatic) translation of a model represented in one world view to another. The specification focuses on the most basic or primitive characterization of time and state, thus making explicit causal dependencies among model objects more identifiable (Overstreet and Nance 1985). Graphical representations derived from the CS, akin to those offered by Schruben (1983) for the event scheduling world view, prove useful in model transformation and simplification (Overstreet and Nance 1986) and automated model diagnosis (Nance and Overstreet 1987).

Since each world view is based on a particular SPL that provided its own approach to time-advance, model implementations are usually closely tied to both a world view and the time-advance technique of the SPL used. These time-advance techniques can vary significantly in the run-time characteristics of executing models (depending on characteristics of the model). If a "world view independent" model specification could be created, then the choice of time-flow technique could be based on issues such as run-time efficiencies.

Note that several of the ideas in this paper appear in (Overstreet and Nance 1986) but due to the limited availability of this reference and continuing interest in the work, we describe that research to broaden its availability to the simulation community.

2 CONDITION SPECIFICATION OVERVIEW

The creation of the semantic forms embedded in a Condition Specifications (CS) is motivated by interest in better understanding the relationships among the three classical discrete event world views through enabling automated transformations from one world view to another. Our re-

search leads to the conviction that direct transformation is often made difficult or impossible because of implicit information (sometimes used by a modeler or programmer to simplify the implementation task) that may not occur in one world view but which must be explicit in another. For a transformation from one world view to any other to be completely automated, this implicit information must sometimes be represented explicitly in another world view representation and it may be difficult to determine. We provide examples of this in Section 5 with examples of world-view-based simplifications. The relevant point here is that the form and some of the semantic content required in a CS results from this transformation issue.

In a CS, a model is a collection of *Objects* and a collection of *Action Clusters* (ACs). Each object is composed of *attributes* and possibly additional objects. The attributes record part of the state of the object. Note that additional object state information may be used to improve run-time efficiency, support time management, or provide automated data collection, but this information may not explicitly appear in any AC. The additional objects, if any, complete the attribute definition for the composite object. The model itself is usually such a composite object. (We note that objects play no role in event scheduling or activity scanning representations and are used when generating process interaction representations.)

A collection of ACs defines the dynamic behavior of a model. Each AC consists of a boolean expression and a collection of actions (such as changing the value of an attribute or scheduling some future action). The actions of an AC are to occur whenever that AC's boolean expression, also called its condition, is TRUE. Thus the form of a CS is similar to the production rule approach to knowledge representation (Newell and Simon 1972). The CS approach has been described elsewhere (Overstreet and Nance 1985, Zeigler 1984, pp.353-358). While the treatment here is brief, it should be sufficient to follow the development. Some additional specifics of CSs are illustrated in the example in Section 3.

Zeigler discusses constructing a model specification by describing its "entity structure" (Zeigler 1984). The approach taken here is influenced by Zeigler's work, although exhibiting a significant departure. The objects in a CS do not provide a "partitioning" (in the mathematical sense) of the model like that of Zeigler's entity structure. We find the idea of objects providing a partition of the model into "disjoint submodels" (whatever that may mean) appealing and abandon this idea with reluctance. However, we find it necessary to do so when considering models with objects that participate in joint activities. To generate an acceptable specification in the process interaction world view automatically, the activities (and the state variables necessary to control the activities) may need to be associated with each of the objects participating in the activity. If this occurs, no partition is created.

2.1 Condition Specification Example

We use the classical Machine Repairman model as an example of a CS and later to illustrate world view transformations. This model is from (Palm 1947) and (Cox and Smith 1961).

Informal Model Description: A single repairman services a group of *n* identical semiautomatic machines. Each machine requires periodic service based on a negative exponential random variable with parameter "meanUptime." The repairman starts in an idle location and, when one or more machines require service, the repairman travels to the closest failed machine. Service time for a machine follows a negative exponential distribution with parameter "meanRepairtime." After servicing a machine, the repairman travels to the closest machine needing service or to the idle location to await the next service request. The closest machine is determined by minimal travel time. Travel time between any two machines or between the idle location and a machine is determined by a function evaluation.

An object specification for this model is provided in Figure 1. Each attribute is associated with one or more model objects and has a name and a type. The attribute types are typical of those available in many programming languages except for the addition of "time-based signal." A time-based signal is a boolean expressions, with its value changing from FALSE to TRUE due to the passage of simulation time.

Object	Attribute	Type
environment	systemTime	nonnegative real
	n	const pos integer
	maxRepairs	const pos integer
	meanUptime	const pos real
	meanRepairtime	const pos real
	initialization	time based signal
machine	n	const pos integer
	maxRepairs	const pos real
	meanUptime	const pos real
	meanRepairtime	const pos real
	mach[1..n]	const 1..n
	failure	time based signal
	arrMach	time based signal
	endRepair	time based signal
	numRepairs	nonnegative integer
	failed[1..n]	boolean
repairman	maxRepairs	const pos integer
	meanRepairtime	const pos real
	status	{avail,travel,busy}
	location	{idle,1..n}
	endRepair	time based signal
	arrMach	time based signal
	numRepairs	nonneg integer
	arrIdle	time based signal

Figure 1: Machine Repairman Object Specification

Figure 2 contains ACs for the model; each AC consists of a name (used in the graphic representations of the next section), a condition, and a set of actions. Conceptually

(implementations may be quite different), whenever the condition of an AC becomes true, its associated actions are to occur. If multiple conditions can become true at the same instant and the actions are order-dependent, then this is considered a specification error and the conditions should be extended to resolve this. (This problem could also be resolved by associating a priority to resolve "ties" but that is not the approach we have taken.

Action Cluster Id: Condition	Actions
initialization: initialization	INPUT n,maxRepairs, meanUptime,meanRepairtime CREATE(repairman) FOR i = 1 TO n CREATE(machine[i]) mach[i] = i mailed[i] = false SETALARM(failure, negExp(meanUptime),i) numRepairs = 0 location = idle status = avail
termination: numRepairs > maxRepairs	STOP
failure(i): failure	failed[i] = true
begin repair(i): arrMach	SETALARM(endRepair, negExp(meanRepairtime,i) status = busy location = mach[i]
end repair(i): endRepair	SETALARM(failure, negExp(meanUptime),i) failed[i] = false status = avail numPrepairs = numRepairs+1
travel to idle: (FOR ALL i, NOT failed[i]) AND status = avail AND location≠ idle	SETALARM(arrIdle, travelTime(location,idle)) staus = travel
arrive idle: arrIdle	status = avail location = idle
travel to mach: status = avail AND(FOR SOME i failed[i])	SETALARM(arrMach,travelTime(loction,mach[i]), closestFaileMach(failed, location)) status = travel

Figure 2: Machine Repairman Action Clusters

The SETALARM action has two or more parameters: the first names the time-based signal to be set, the second the time the alarm is to occur. Any subsequent parameters are used to pass values to any AC associated with that time-based signal.

To illustrate the semantics of Figure 2, the simulation stops when `numRepairs` is greater than `maxRepairs` (the termination AC); a "begin repair" occurs for some machine, whenever `arrMach` is true (that is, the repairman has arrived at a machine, an event scheduled by the "travel to mach" AC).

We omit the definitions of the functions `closestFailedMach` which identifies the closest failed machine to the repairman's current location and the function `travelTime` that determines the travel time from the repairman's current location to a particular machine or the idle location.

2.2 Action Cluster Interaction Graphs

One advantage of a CS is that it can be used to deduce interactions among components of a specification. A representation that supports this analysis is an *Action Cluster Interaction Graph (ACIG)* (Overstreet 1982, pp. 130-131). In an ACIG, nodes represent ACs and directed edges the ability of one AC to directly cause the occurrence of another AC, that is, in an ACIG, an edge leads from AC 1 to AC 2 if the actions of AC 1 can cause the condition of AC 2 to become true either at the same instant AC 1 is activated or at a future instant (through a SETALARM action).

Figure 3 is the ACIG for the Machine Repairman example. In this figure, an AC that can cause another AC to occur in the same instant (that is, with no change in simulation time) is connected to it by a solid edge. If an AC directly causes another AC to occur at a future time (through a SETALARM action), the ACs are connected by a dashed edge.

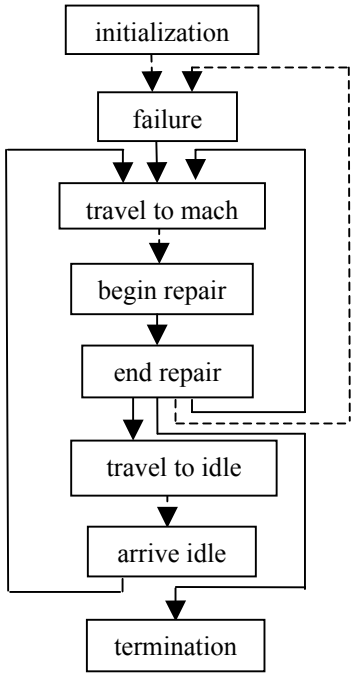

Figure 3: Machine Repairman Action Cluster Graph

See (Overstreet and Nance 1985) for a discussion of how this graph can be generated from a CS and some of the interesting problems involved in this process. The graph is introduced here since representations in each of the three world views are generated from it in Section 4.

3 INFORMAL CHARACTERIZATION OF WORLD VIEWS

We do not attempt a formal characterization of the three world views, see (Zeigler 1976, Chapter 9) for a more formal approach). The characterization of each world view is based on our interpretation of the concepts found in SIMSCRIPT (Dimsdale and Markowitz 1964) for the event scheduling world view, in CSL (Buxton and Laski 1963) for the activity scanning world view, and in SIMULA (Dahl and Nygaard 1966) for the process interaction world view.

We base our ideas on world view as providing different types of locality. Locality is defined by Weinberg as "that property when all relevant parts of a program are found in the same place" (Weinberg 1971). Locality is generally regarded as a positive attribute by the software engineering community since software is usually more easily understood, reused and maintained if all the parts of a program that provide certain behaviors are linked in a readily identifiable manner.

Unfortunately, what is "relevant" highly depends on the issue of interest. Thus it is impossible for one arrangement of the source text of a specification or program) to exhibit locality for all possible questions that the source text might be accessed to answer.

We assert that each world view attempts to capture a different kind of locality:

- *Event scheduling* provides *locality of time*: each event routine in a model specification describes related actions that should always all occur in one instant.
- *Activity scanning* provides *locality of state*: each activity routine in a model specification describes all actions that should occur due to the model assuming a particular state (that is, due to a particular condition becoming true.)
- *Process interaction* provides *locality of object*: each process routine in a model specification describes the action sequence of a particular model object.

These characterizations are illustrated by the transformations in the next section.

4 WORLD VIEW TRANSFORMATIONS

Transformations into each of the three world views can be treated as a two step process. First, appropriate subgraphs are generated from an ACIG, with different types of subgraphs for each world view. This can be done entirely automatically as described below. The second step simplifies each specification by use of precondition/postcondition analysis. The complexity of this step deserves the added discussion given below.

A CS contains three types of ACs:

1. *determined* if the attributes in the condition expression are all time-based signals,
2. *contingent* if the condition expression contains no attributes which are time-based signals, an
3. *mixed* if the conditions contains both time-based signal and non-time-based signal attributes. Thus the condition value for a determined AC depends only on the value of simulation time (at least, after the signal has been scheduled).

The simple transformation of a CS with mixed ACs into an equivalent specification with no mixed ACs is accomplished by the addition of attributes (Overstreet 1982). For the development that follows, it is convenient to assume that the specification contains no mixed ACs.

Each Condition Specification contains one special attribute, "initialization." The transformation algorithms discussed below are based, in part, on identifying those attributes that are time-based signals. For these algorithms, it is useful to treat the "initialization" attribute as both time-based (for event scheduling) and non-time-based (for activity scanning). This dual treatment is not unreasonable because it seems equally correct to regard initialization as occurring when system time is zero (it is a determined or time-based action) or as occurring because the model execution is initiated (a contingent action—the model should be initialized in every generation of sample behavior).

4.1 Event Scheduling

In an event scheduling world view, a modeler first identifies actions that are scheduled (i.e., time-based) and then for each of the scheduled actions, everything else that might happen as a consequence of that action both at the same instant of time and at some point in the future. Thus for each determined action, a modeler identifies both contingent actions (which can occur as a result of the determined action and in the same instant) and additional determined actions that occur as a direct result of the original determined action. Each event specification then consists of one determined AC and a collection of contingent ACs that may be caused by it.

Given an ACIG for a model specification, determined actions are identified by those nodes with dashed input edges (using the notation of Figure 3) plus the initialization AC. The contingent actions which can occur in the same instant as each determined action and which can be caused by the determined action are identified by the solid edges leading from the determined node. These subgraphs, called *event subgraphs*, are easily generated from an ACIG. Each subgraph represents the sequence of actions of one event specification. Figure 4 contains the five event subgraphs for the Machine Repairman model.

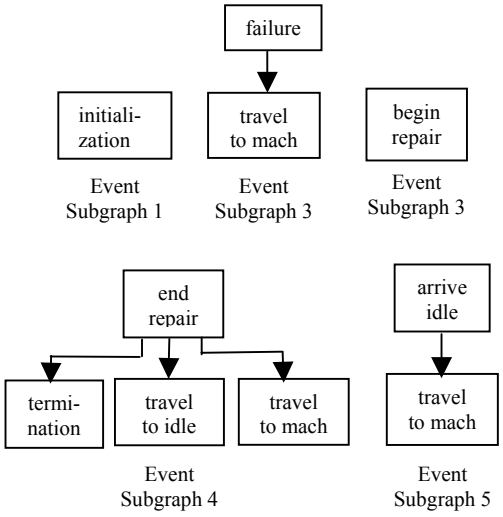

Figure 4: Machine Repairman Event Subgraphs

For the five event subgraphs, subgraph 1 contains the single AC "initialization." Subgraph 2 starts with the AC "failure," and, since a "failure" may cause a "travel to mach" to occur in the same instant (if the repairman is available), it also contains the AC "travel to mach." A "begin repair" can cause no other actions to occur in the same instant so the graph contains this single AC, but the "end repair" AC can cause either a "termination," a "travel to idle," or a "travel to mach" to occur in the same instant, so the subgraph contains all four ACs. As depicted in subgraph 5, the "arrive idle" AC can cause an instant "travel to mach."

4.2 Activity Scanning

We assert that the use of an activity scan world view requires the modeler first to identify all conditions to which the model must respond other than those dependent strictly on the passage of simulation time. After identifying these conditions, the modeler specifies for each condition all actions that should occur unconditionally including actions that should occur at a future time. Each activity specification consists of one condition and a collection of actions which must all occur whenever that condition is TRUE.

Given an ACIG for a model specification, contingent actions are identified by those nodes with solid input edges (using the notation of Figure 3). The determined actions that can occur as a direct result of each contingent action are identified by the dashed edges leading from the contingent node. Thus subgraphs, called *activity subgraphs*, are easily generated in which each subgraph represents the sequence of actions for one activity specification. Figure 5 contains the activity subgraphs for the Machine Repairman model.

Activity subgraph 1 indicates that, once an "initialization" has occurred, a "failure" is scheduled to occur. Like wise, activity subgraph 2 indicates that once a "travel to mach" has occurred, a "begin repair," an "end repair," and a "failure" must necessarily occur (unless the simulation is terminated). From subgraph 3, the "travel to idle" results in an "arrive idle," and a "termination" results in no other actions.

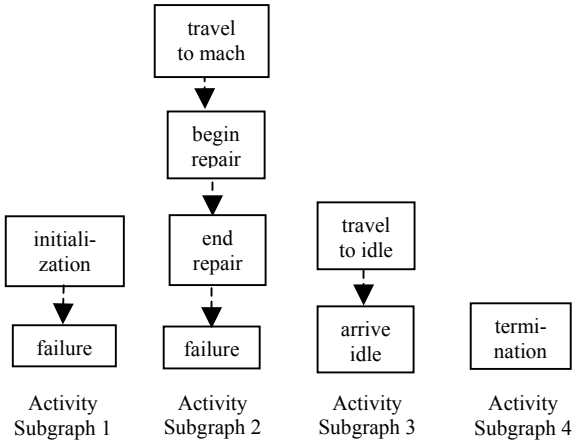

Figure 5: Machine Repairman Activity Subgraphs

4.3 Process Interaction

We assert that a process interaction world view requires the modeler first to identify all model objects whose action sequences must be defined. Subsequently, the modeler specifies the sequence of actions for each object.

The object specification, illustrated in Section 2, associates each attribute with one or more objects, from which each AC can be associated with one or more model objects. Each AC is associated with each model object if the object contains an attribute (1) that occurs in the AC condition, or (2) that the AC alters.

After this association of objects and ACs is complete, subgraphs are generated to represent the action sequences of each object. Each process subgraph contains all nodes that represent the ACs associated with that object and the edges of the ACIG connecting those nodes. These subgraphs, called *process subgraph*s, also provide sequencing of the object actions. Figure 6 contains the process subgraphs for the Machine Repairman model.

This model has three process subgraphs, one for each object. Although each process subgraph normally depicts the sequence of possible actions for some model object, complete sequencing of actions may not be provided. For example, for this model, the process subgraph for the environment, subgraph 1, consists of two ACs, "initialization" and "termination" and the graph is not connected. This is not unusual for an environment object (which often contains only initialization and termination ACs), but unconnected process subgraphs can also occur for other model objects. The other two subgraphs for the machine and repairman objects do depict action sequences for the objects and nicely reveal the cyclical behavior of each.

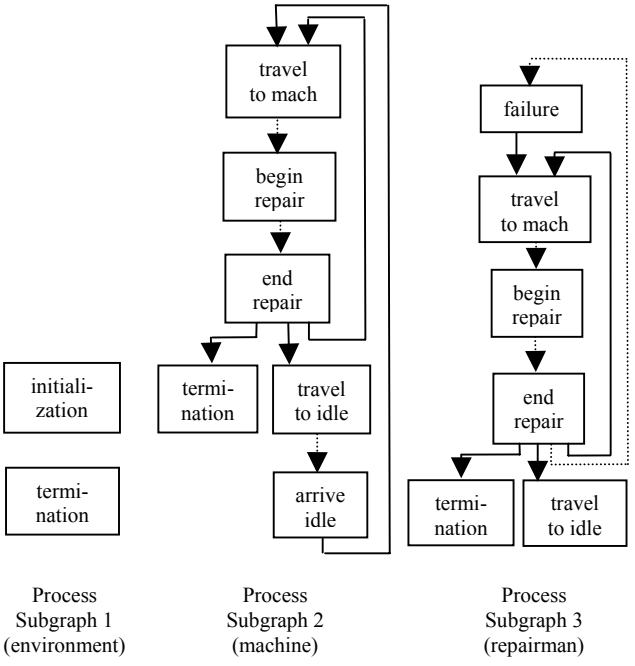

Figure 6: Machine Repairman Process Subgraphs

For each of these transformations, the union of the set of nodes in the subgraphs may not be the complete set of nodes in the original ACIG since some nodes may appear in no generated subgraph. Surprisingly, this is not a problem for generating transformations since for each of the three world view transformations, any node not included in any of the world view subgraphs cannot occur in any execution based on the CS (Overstreet 1982). The action cluster associated with any unincluded node likely indicates a specification error, though it can be deleted from the specification without affecting model behavior defined by the rest of the specification.

5 WORLD VIEW BASED SIMPLIFICATION

Each individual event, activity, and process specification resulting from the above graphs can potentially be simplified by removal of unnecessary tests or actions and simplification of some control structure. This is an important part of the transformation process, for it allows the resulting specifications to take additional advantage of the representational advantages of a target world view. Effective automation of this simplification is possible; Puthoff (1990) describes an expert system that performs this task for CSs. Nance, Overstreet and Page (1999) describe improved execution performance for several simulation models based on the improvements found by Puthoff's expert system.

5.1 Precondition/Postcondition Simplification

All simplifications illustrated here are the result of a pre/post condition analysis for sequences of model actions. The condition expressions for each AC should define a minimal precondition for the actions of that cluster. Each subgraph, whether event, activity, or process, defines a sequence of model actions. Analysis of the actions of each sequence allows formulation of postconditions for each AC. Comparison of the postcondition of an AC with the precondition of each of its successors (as identified by the appropriate subgraph) provides one basis for simplification of the specification.

Several types of the possible world-view-based simplifications are illustrated through an example. Consider a specification of the End Repair event following an event scheduling world view. A direct transformation (without any simplification) of the ACs into an event specification is presented in Figure 7. Lines 1 through 4 are from the End Repair AC, lines 5 and 6 from "termination," lines 7 through 9 from "travel to idle," and lines 10 through 13 from "travel to mach."

```
EVENT end repair(i:1..n)
1: SETALARM(failure,negExp(mean_uptime),i)
2: failed[i] = FALSE
3: status = avail
4: numRepairs = numRepairs+1
5: WHILE(numRepairs>maxRepairs)
6:   STOP
7: WHILE((FORALL i IN 1..n, NOT failed[i])
AND
     status=avail AND location≠idle){
8:   SETALARM(arrIdle,travelTime(location,
       idle))
9:   status = travel
     }
10:WHILE(status=avail AND
 (FOR SOME i IN 1..n,failed[i])){
11:  SETALARM(arrMach,traveltime(location,
   mach),closestFailedMach(failed,
   location))
12:  status = travel
 }
```

Figure 7: End Repair Event Specification–Unsimplified

Note that the condition of each contingent AC becomes the condition for a "while" construct. A "while" is required rather than an "if" since the actions of the AC should be repeated as long as its condition is satisfied.

Several types of simplification of this specification are possible. For this event specification, all WHILE constructs have been replaced with noniterative constructs (the simplified specification is presented in Figure 8). For the "travel to idle" and "travel to mach" ACs, replacement can be done since an action of each changes the value of the condition to FALSE. For example, in Figure 7, the condition of line 7 specifies that status must have the value "avail," but line 9 sets its value to "travel." In the case of the "termination" AC, lines 5 and 6 of Figure 7, the STOP action of line 6 implies that the condition of the AC need

not be reevaluated. Thus a postcondition of these ACs implies that their precondition is not satisfied without some further model action.

```
   EVENT end repair(i:1..n)
1:   SCHEDULE(failure,negExp(meanUptime),i)
2:   failed(i)=FALSE
3:   num_repairs = num_repairs+1
4:   IF num_repairs>max_repairs
5:     STOP
6:   IF FOR ALL i IN 1..n NOT failed[i]
7:     SCHEDULE(arrIdle,travelTime(
         locationIdle))
    ELSE
8:     SCHEDULE(arrMach,travelTime(location,
         mach),closestFailedMach(failed,
         location))
9:   status := travel
```

Figure 8: End Repair Event Specification–Simplified

Another obvious simplification is possible when an action of a preceding AC eliminates the necessity of testing the value of an attribute in the condition of a successor AC. This happens several times in this event specification. In line 3 of Figure 7, status is set to "avail" thus, its value need not be explicitly tested in the conditions of 7 and 10.

The validity of eliminating the test of the value of location of line 7 is more difficult to justify, but still results from a postcondition of the "end repair" AC. A partial analysis, to illustrate the process, follows. Unlike the simplifications of the test for status in lines 7 and 10, the value of location is not altered by the "end repair" AC. But a precondition for the occurrence of "end repair" is that location not have the value "idle." This condition is satisfied since (1) the value of location is only altered by the "begin repair" and "arrive idle" ACs, (2) an "end repair" can only occur after a "begin repair" (since this is the only place the "endRepair" alarm is set), (3) the value of location is not "idle" after the "begin repair" action, and (4) an "arrive idle" cannot occur between a "begin repair" and "end repair" (so that the value of location cannot be altered before the "end repair" occurs).

Two other simplifications are made in producing Figure 8. First, the two conditions in lines 7 and 10 of Figure 7 are such that if one is true, the other is false, so the two WHILEs become an IF-THEN-ELSE construct. Secondly, the action of line 3 of Figure 7 has been eliminated. This is valid since, if the model execution does not terminate, the value of status will be changed to "travel" either in lines 9 or 12 since these two alternatives are part of an IF-THEN-ELSE action. So setting it the value of status to "avail" is obviated.

The result of these simplifications is presented in Figure 8. SETALARMs are replaced with SCHEDULEs so that the syntax looks more like an event scheduling language. Because of space constraints, the additional simplifications to complete the event scheduling specification are omitted.

The process for each of the three world views transformations is similar and consists of deriving postconditions for sequences of ACs and comparing them with preconditions of other ACs. In general, the transformations involve reorganizing provided aggregates of model actions and tests into different groupings, followed by elimination of resultant redundancies in each new grouping. The simplifications are desirable if the resulting specification is to take full advantage of the target world view

While some simplifications are easily automated, others appear complex. Our experience in this area indicates that the automated discovery of some simplifications requires a robust theorem proving system; the simplification of line 7 of Figure 7 provides an example. The proof that it is not necessary to consider the value of "location" is not trivially derived. Identifying such properties of a specification automatically can be difficult or even impossible; no *a priori* bound for the complexity can be established.

5.2 Impact of Simplifications

We believe that modelers, in producing both specifications and implementations, intuitively use something like pre/postcondition analysis. They sometimes omit model actions and tests for parts of preconditions (and sometimes entire preconditions) when they know that the tests or actions are unnecessary. It seems likely that these omissions can be based on more than what we have illustrated in the above example (where all are based on analysis of the provided specification); modelers may also draw both on their deep knowledge of possible behaviors of a simulated system and on their understanding of implementation details of a supporting simulation tool.

Each eliminated test and action becomes implicit in the resulting representation. However in another world view, this implicit test or action may require explicit representation. A direct transformation from one world view to another can require discovery of what was omitted by a specifier or programmer. This poses a problem so formidable that we suspect no general algorithm for direct world view transformations can be constructed.

A converse issue, in a sense, is the challenge of the developer of the translator from the model specification to the model implementation (the executable program). The CS as a specification tool is focused correctly on *what* behavior is intended. The implementation describing *how* that behavior is achieved requires additional resolution of abstraction and considerably more detail as discussed in (Page and Nance 1999). Automatic translation from the CS, or any other specification language representation, forces the translator writer to make assumptions and decisions that should include the potential application domain(s), the users of the model development environment, and the likely costs incurred in experimental use.

6 SUMMARY

In other places (Overstreet and Nance 1984, 1987), we have discussed supporting the model specification process by providing analysis tools to evaluate a model specification during development. The traditional world views produce model specifications with embedded implicit knowledge that can simplify the specification process but at the price of inhibiting or preventing assistance through analysis. The benefits of model analysis are still being explored.

The characterization of each world view as providing a different type of locality reflects our view that no single world view is necessarily superior to any other for simulation modeling in general. This lack of superiority is due in part to the significant variation in model simplification possible with each world view that is due to properties of the model being specified. The choice of the "proper" world view for a particular model can impressively reduce the task of specifying model behavior due to the variation in implicit actions and conditions possible within each world view.

The fact that each world view encourages omission (at least in terms of how much must be included in the specifications) appears to make direct translation among the various world views impossible. Automated translation from a more basic form, a Condition Specification, into representations that take advantage of each of the three world views is instructive in identifying some benefits of each world view. The CS's ability to support multiple world views, it's enabling of significant model analysis for both error detection and identification of some revealing model characteristics, and the ability to create efficient run-time implementations directly from it makes the CS an useful tool for model specification.

REFERENCES

Buxton, J. N. and J. G. Laski. 1963. Control and simulation language. *The Computer Journal* 5: 194-199.

Cox, D. R. and W. L. Smith. 1961.*Queues*. Methuen and Company, Inc.

Dahl, O. Nygaard. 1966. SIMULA–an ALGOL - based simulation language. *Communications of the ACM*. 9 (9):349-395.

Dimsdale, B. and H. M. Markowitz 1964. A description of the SIMSCRIPT language, *IBM Systems Journal* 3(1): 57-67.

Holbaek-Hanssen, E., P. Hfindlykken, and K. Nygaard, 1977. System description and the DELTA language, DELTA project report no. 4. Second printing. Norwegian Computing Center.

Kiviat, P. J. 1969. Digital computer simulation: Computer programming languages. RAND Memo. RM-5883-PR, RAND Corporation, Santa Monica, California.

Lackner, M. R. 1962. Toward a general simulation capability. In *Proceedings of the SJCC*, 3. San Francisco, California.

Lackner, M. R. 1964. Digital simulation and system theory. System Development Corporation, SDC SP-12, Santa Monica, California.

Nance, R. E. 1979. Model representation in discrete event simulation: Prospects for developing documentation standards. In *Current issues in computer simulation*, ed. N. Adam and A. Dogramaci, New York, Academic Press.

Nance, R. E. 1981. The time and state relationships in simulation modeling. *Communications ACM* 24(4): 173-179.

Nance, R. E. 1996. A history of discrete event simulation programming languages. In *History of Programming Languages*, eds. T. J. Bergin and R. G. Gibson, 369-427. New York: Association for Computing Machinery Press and Addison-Wesley Publishing Company.

Nance, R. E. and C. M. Overstreet, 1987. Diagnostic assistance using digraph representations of discrete event simulation model specifications, *Transactions of the Society for Computer Simulation* 4 (1): 33-57.

Nance, R. E., C. M. Overstreet, and E. H. Page, 1999. Redundancy in model specifications for discrete event simulation, *ACM Transaction on Modeling and Computer Simulation*, 9(3): 254-281.

Newell, A. and H. A. Simon. 1972. *Human Problem Solving,* Prentice-Hall, Inc., Inglewood Cliffs, NY, 22-33.

Overstreet, C. M. 1982. Model specification and analysis for discrete event simulation. Doctoral dissertation, Virginia Polytechnic Institute & State University, Blacksburg, Virginia.

Overstreet, C. M. and R. E. Nance, 1985. A specification language to assist in analysis of discrete event simulation models. *Communications AC,* 28(2): 190-201.

Overstreet, C. M. and R. E. Nance, 1986. World View Based Discrete Event Model Simplification, *in Modelling and Simulation Methodology in the Artificial Intelligence Era*, North-Holland Publishing Co., Amsterdam, The Netherlands, ed. M. Elzas, T. Oren, B. Zeigler, 165-179.

Page, E. H. and R. E. Nance, 1999, Incorporating support for model execution within the condition specification, *Transactions of the Society for Computer Simulation International,* 16(2)*, 47-62.*

Palm, D. C., 1947. The distribution of repairmen in servicing automatic machines*, Industritidningen Norden*, 175(75). (Swedish).

Puthoff, F. A., 1990. The model analyzer: prototyping the diagnosis of discrete event model specification, M. S. Thesis, Department of Computer Science, Virginia Tech, Blacksburg, VA.

Schruben, L. M. 1983. Simulation modeling with event graphs. *Communications ACM,* 26(11): 957-963.

Weinberg, G. M. 1971. *The Psychology of Computer Programming*, Van Nostrand Reinhold, New York. 229.

Zeigler, B. P. 1972. Towards a formal theory of modeling and simulation: Structure preserving morphisms. *Journal of the ACM,* 19 (4): 742-764.

Zeigler, B. P. 1976. *Theory of Modelling and Simulation,* John Wiley & Sons, New York.

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

AUTHOR BIOGRAPHIES

C. MICHAEL OVERSTREET is an Associate Professor of Computer Science at Old Dominion University. A member of ACM and IEEE/CS, he is a former chair of SIGSIM, and has authored or co-authored over 80 refereed journal and conference articles. He received a B.S. from the University of Tennessee, an M.S. from Idaho State University and an M.S. and Ph.D. from Virginia Tech. He has held visiting appointments at the Kyushu Institute of Technology in Iizuka, Japan, and at the Fachhochschule fũr Technik und Wirtschaft in Berlin, Germany. His current research interests include model specification and analysis, static code analysis and support of interactive distance instruction. Dr. Overstreet can be reached by e-mail at <cmo@cs.odu.edu>; his home page is <www.cs.odu.edu/~cmo>.

RICHARD E. NANCE is the Chief Scientist of Orca Computer, Inc. and Emeritus Professor of Computer Science at Virginia Tech. He served on the faculties of Southern Methodist University and Virginia Tech, where he was department head of Computer Science, 1973-1979 and held the John Adolphus Dahlgren Chair in Computer Science, 1988-2004. He held a distinguished visiting honors professorship at the University of Central Florida for the spring semester, 1997. Dr. Nance has held research appointments at the Naval Surface Weapons Center and at the Imperial College of Science and Technology (UK). He has held a number of editorial positions and was the founding Editor-in-Chief of the ACM *Transactions on Modeling and Computer Simulatio*n, 1990-1995. He served as Program Chair for the 1990 Winter Simulation Conference. Dr. Nance received a Distinguished Service Award from the TIMS College on Simulation in 1987. In 1995 he was honored by an award for "Distinguished Service to SIGSIM and the Simulation Community" by the ACM Special Interest Group on Simulation. He was named an ACM Fellow in 1996. He is a member of Sigma Xi, Alpha Pi Mu, Upsilon Pi Epsilon, ACM, IIE, INCOSE and INFORMS. His email address is <nance@vt.edu>