TR-86-8

DIAGNOSTIC ASSISTANCE USING DIGRAPH REPRESENTATIONS OF DISCRETE EVENT SIMULATION MODEL SPECIFICATIONS*

C. Michael Overstreet†
and
Richard E. Nance‡

26 March 1986

*Research partially supported by the Office of Naval Research and the Naval Sea Systems Command through the Systems Research Center at Virginia Tech.
†Department of Computer Science, Old Dominion University, Norfolk, Virginia 23508.
‡Systems Research Center and Department of Computer Science, Virginia Tech, Blacksburg, Virginia 24061
Also cross-referenced as Technical Report Number SRC-86-001, Systems Research Center, Virginia Tech.

SRC-86-001

DIAGNOSTIC ASSISTANCE USING DIGRAPH REPRESENTATIONS OF DISCRETE EVENT SIMULATION MODEL SPECIFICATIONS*

C. Michael Overstreet†
and
Richard E. Nance‡

26 March 1986

*Research partially supported by the Office of Naval Research and the Naval Sea Systems Command through the Systems Research Center at Virginia Tech.
†Department of Computer Science, Old Dominion University, Norfolk, Virginia 23508.
‡Systems Research Center and Department of Computer Science, Virginia Tech, Blacksburg, Virginia 24061
Also cross-referenced as Technical Report Number TR-86-8, Department of Computer Science, Virginia Tech.

ABSTRACT

Automated diagnosis of digraph representations of discrete event simulation models is illustrated as an effective model verification technique, applicable prior to coding the model in an executable language. The Condition Specification is shown to provide an effective representation, from which automated analysis can initiate with a digraph extraction. Subsequent diagnostic simplification techniques are applied to the digraph, either automatically or in concert with the modeler. Three categories of diagnostic assistance are defined: analytical, comparative, and informative. The categories represent diagnostic techniques that are progressively more difficult to automate. Examples of techniques assigned to each category are included, and a concluding caution is made that diagnostic techniques requiring modeler interaction should not be ignored in the preoccupation with complete automation.

CR Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Testing and Debugging — *Diagnostics*; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

Additional Keywords: model development, model specification

1. INTRODUCTION

Software development technology is recognized as an area in which major advances are needed. Various systems and methodologies have been advocated for reducing software costs (see, for example, [14, 36]). The work reported here focuses on *model development*, which encompasses more than "program development" [25, p. 326]. By restricting attention to the modeling domain of discrete event simulation, we hope to identify techniques which can reduce the cost and improve the quality of simulation models. This work represents an extension of earlier research in the representation of simulation models [29], and the reader is advised to consult the earlier paper for details omitted here for brevity.

1.1. The Role of Diagnosis in a Model Development Environment

The complexity and scope of simulation models has forced reliance on automated assistance. While different tools are required in different phases of the model life cycle, we concentrate on diagnostic procedures intended to support the model development phases (see Nance [25, pp, 326-329]). These tools assist the development process by providing early diagnosis of several types of real or potential problems and by providing alternative forms of documentation.

Several researchers have reported work in the area of simulation support; among the more interesting are the program generators [5, 8, 19, 35]. Other authorities have asserted that, if significant improvements are to be realized, the problems of quality and productivity must be addressed at a global level through the creation of integrated systems designed to provide an environment for simulation model development and experimentation [23, 27, 39]. In this paper we discuss some tools which are intended to be part of a model development environment (MDE), following an architecture described by Balci [1].

1.1.1. The Context

A model development environment (MDE) consists of a set of hardware and software tools, collectively referred to as the model development system, and the physical surroundings in which the development tasks are accomplished. Typically, the term "environment" is used; however, emphasis is clearly placed on the tool set: a configuration of hardware, languages, data bases, and utilities residing within an operating system enabling interactive use.

Through experience with rapid prototyping of several utilities, we are attempting to define the requirements for a "production" MDE. The Conical Methodology [22] forms the underlying conceptual framework for the prototypes by:

(1) guiding in the partitioning of model development functions among the tools (see [1]),

(2) clarifying the relationships among the tools to achieve an integrated environment, and

(3) assisting in the definition of tool functionality through the emphasis on specific principles.

As an example of the principles governing tool functionality, the methodology requires that the modeling team employ a top-down model definition followed by bottom-up model specification, producing model documentation during the construction process. Additionally, project documentation is an indispensable requirement of the MDE. Important in the Conical Methodology is the clear distinction between a "model specification" (which describes how the model is to behave) and a "model implementation" (which describes how the behavior is to be achieved).

1.1.2. Model Diagnosis

Within a model development environment based on the Conical Methodology the purposes of model diagnosis are:

(1) to assist in the identification of conceptual errors (misperceptions) or descriptive errors (misrepresentations) as early as possible in the modeling effort,

(2) to suggest alternatives that might be less prone to errors or might offer more efficient model representation and experimentation, and

(3) to provide guidance and checks on the modeling effort (for example, the reminder to update required documentation as changes are made in the model representation).

Diagnosis begins with the first communicative model and continues throughout the model life cycle [22, pp. 12-17].

Three categories of diagnostic assistance are identified within the MDE. The most common is *analytical diagnosis*: determination of the existence of a property such as attribute utilization or revision consistency (both discussed below). The second category is *comparative diagnosis*, which requires the definition of measures intended to depict differences among multiple representations of a single model or between representations of different models. *Informative diagnosis* constitutes the third category, which includes assistance in the form of characteristics extracted or derived from model representations. Graphical techniques are prominent in this last category.

1.2. Related Work

Graphical depictions of system behavior represent a longstanding simulation modeling technique. Among the program generators, for example CAPS [5] and DRAFT [19], the entity cycle diagram provides a graphical language for system description that originated with the "wheel chart" of K.D. Tocher [34]. The GPSS block diagram can be traced to the

beginnings of the language [11]. Zeigler's influence diagram [38, p.92], for example, characterizes the interactions among cells that can potentially effect a state transition. More generically, the influence diagram depicts the causality in state transitions among system components.

The references cited above employ graphical techniques as a modeling form, characterizing the behavior among system components. The diagnostic techniques that follow are applied to model representations in order to detect modeling errors or provide information to the modeler. In this respect, the intent is similar to that of Burns and Winstead [3], who utilize a signed digraph to capture causality assumptions in a systems dynamics model. The authors, commenting on the application to discrete event models, restrict their consideration to determined events (strictly time dependent in contrast with state dependent or a combination).

DeCarvalho and Crookes [9] use a graphical representation of a model specification to identify independent "cells" in order to improve the efficiency of an activity scanning time flow mechanism. The graphical form also serves to identify components with output that can be revised in replications of the simulation run.

Schruben's event graphs offer the closest parallel to the graph-based diagnostics to be described [33]. The primary distinctions between event graphs and those that follow lie in the semantic level of description. Event graphs inherently represent the event scheduling world view (see [16, pp. 20-21]), limiting the representations of relationships to those permitted among events (scheduling, conditional occurrence, and cancellation). We utilize graphs for diagnostic purposes that depict relationships among objects, attributes of objects, states of objects (activities), and events (changes in the states of objects). (See [24, p. 176] for a precise definition of "event" and "activity.")

Schruben's objectives are similar to ours: use graphical analysis to simplify model representation and recognize potential logic errors. However, by working at a more primitive semantic level, stripping away the implicit knowledge embodied in the event scheduling world view, we can capture broader ("richer"), but less precise, relationships.

An explanation of the implicit knowledge embodied in world views is beyond the scope of this paper. The interested reader is referred to [16], [28], [29], and [30].

The Conical Methodology can be accurately classified as an entity-relationship (E-R) (also referred to as entity-relation-attribute (ERA)) modeling technique generally attributed to Peter Chen [4]. However, the conceptual underpinnings are influenced more by the simulation programming languages SIMULA and SIMSCRIPT and the ideas of George Mealy [21] and P.J. Kiviat [15,16] than by the much later E-R concepts. The CM is object-oriented, while the entity-attribute-set (EAS) conceptual view of system description [17,18] combines an "object influenced" static view with an "event causality" dynamic view. With respect to the description of interactions among permanent entities (in SIMSCRIPT terminology) and accommodation of a top-down modeling approach, the Conical Methodology draws more heavily on concepts from the SIMULA implementation of process interaction.

2. MODEL REPRESENTATION AND MODEL DIAGNOSIS

Accomplishing the purpose of early detection and correction of modeling errors through diagnostic assistance requires a representation formalism prior to that achieved through an executable programming language. That is, a "formal" specification must precede the "formal" implementation (a program). Such a specification formalism is provided by the condition specification (CS) introduced in an earlier paper [29] and briefly reviewed below.

Unlike the representation provided by a simulation program, the condition specification contains none of the syntactic or semantic elements imposed for execution purposes.

Furthermore, the implicit knowledge permitted by the adoption of a world view is removed; the model behavior is made explicit in terms of the objects and relations among the objects comprising the model. Implicit knowledge in a specification impedes the application of diagnostic analysis; therefore, the explicit representation is a precursor to effective diagnosis [30].

2.1. The Condition Specification

A model specification is defined as a quintuple:

< input specifications,
output specifications,
object definition set,
indexing attribute,
transition specification >.

The input and output specifications jointly define the interface between the model and its environment. The indexing attribute, commonly called "system time," provides the means for depicting temporal relationships so fundamental in discrete event simulation.

Objects are defined in terms of attributes, which must be typed by the modeler. The enumeration of values for all attributes of an object at a particular value of system time defines the state of that object at the particular instant (value of system time). The transition specification for a model defines: (1) an initial state, (2) termination conditions, and (3) the dynamic structure — how each attribute value (and object) affects every other attribute value (and object).

The transition specification could take a form such as that used by Zeigler with the transition function, which prescribes a mapping from one model state to another [38, pp. 141-142]. However, the generality achieved by Zeigler's abstract, formal description of the dynamics of component interactions is purchased with a loss in the instructive characterization that promotes convergence to an eventually correct and

executable model representation. The condition specification, summarized in Table 1, is an attempt to strike a balance between descriptive generality and an instructive formalism.

TABLE 1: Syntax and Function of Condition Specification Primitives (from [29, p. 197]).

Name	Syntax	Function
Value change description	Not specified	Assign attribute values
Set Alarm	SET ALARM(< alarm name > [(< argument list >)] , < time delay >)	Schedule an alarm
When Alarm	WHEN ALARM(< alarm name exp > [(< parameter list >)])	Time sequencing condition
After Alarm	AFTER ALARM(< alarm name > & < Boolean exp > [(< parameter list >)])	Time sequencing condition
Cancel Alarm	CANCEL ALARM(< alarm name > [, < alarm id >])	Cancel scheduled alarm
Create	CREATE(< object type > [, < object id >])	Generate new model object
Destroy	DESTROY(< object type > [, < object id >])	Eliminate a model object
Output	Not specified	Produce output
Stop	Not specified	Terminate simulation experiment
Comment	{ < any text not including a "}" > }	Comment

2.2. An Example Revisited: The Machine Repairman Model

Illustration of graph-based diagnostic techniques requires an example, and for brevity we utilize the machine repairman model introduced in [29, pp. 198-199]. The model is a variation of the classical machine interference problem which prescribes ser-

vice of facility failures according to the closest failed facility [31,6].

Model Abstract:

> A single repairman services a group of n identical semiautomatic facilities. Each facility requires service randomly based on a time between failure which is a negative exponential random variable with parameter "mean_uptime." The repairman starts in an idle location and, when one or more facilities requires service, the repairman travels to the closest facility needing service. Service time for a facility follows a negative exponential distribution with parameter "mean_repairtime." After completing service for a facility, the repairman travels to the next service request. The closest facility is determined by shortest travel time. Travel time between any two facilities or between the idle location and a facility is determined by a function evaluation.

Model Objective:

> Estimate total and percentage downtime for each facility.

Model development following the Conical Methodology begins with a definition stage, which concludes with the interface and object specifications shown in Figures 1 and 2. (A Pascal notation is used for comments in all figures and tables.) The model definition produces an object-oriented, static description in terms of strongly-typed attributes. Attribute association with objects can lead to multiple uses of a single attribute in describing objects. This apparent redundancy promotes "correctness" by emphasizing the relationships among objects in the ensuing model specification stage. The model specification stage, requires dynamic description to be imparted through an attribute-oriented construct called a condition action pair (CAP).

A CAP is composed of a condition, a boolean expression or time-based signal, and an associated set of actions, which are taken when the condition is evaluated as "true." Specification via the CAP construct requires the modeler to prescribe the condition(s) under which defined attributes change value and the expression effecting the change.

```
Input:
    n                       { Number of facilities                  } : positive integer;
    mean_uptime             { Mean time between facility failures   } : positive real;
    mean_repairtime         { Mean repairtime                       } : positive real;
    max_repairs             { Number of repairs for termination     } : positive integer;

Output:
    total_downtime [1..n]   { Total downtime for each facility      } : nonnegative real;
    percent_downtime [1..n] { Percentage downtime for each facility } : nonnegative real;
```

FIGURE 1: Machine Repairman Interface Specification

{ Object	Attribute	: Type }
environment	system_time	: positive real;
	n	: positive integer constant;
	mean_uptime	: positive real constant;
	mean_repairtime	: positive real constant;
	max_repairs	: positive integer constant;
facilities	n	: positive integer constant;
	max_repairs	: positive integer constant;
	mean_uptime	: positive real constant;
	mean_repairtime	: positive real constant;
	i	: 1..n
	failure(1..n)	: time-based signal;
	failed[1..n]	: Boolean;
	begin_downtime[1..n]	: positive real ;
	total_downtime[1..n]	: nonnegative real ;
	percent_downtime [1..n]	: positive real;
	end_repair(1..n)	: time-based signal;
	arr_facility(1..n)	: time-based signal;
	num_repairs	: nonnegative integer;
repairman	max_repairs	: positive integer constant;
	mean_repairtime	: positive real constant;
	status	: (avail, travel, busy);
	location	: (idle, i:1..n);
	end_repair(1..n)	: time-based signal;
	arr_facility(1..n)	: time-based signal;
	num_repairs	: nonnegative integer;
idle position	arr_idle	: time-based signal;

FIGURE 2: Machine Repairman Object Specifications

CAPs with identical conditions are grouped into an action cluster (AC). Figure 3 explains and illustrates the relationships among the object specification (definition stage), the CAP (specification stage), and the action cluster.

Object Specification Extract

repairman:	status	:(avail, travel, busy)
	location	:(idle, i:1..n)

Condition Action Semantics

(<condition> , <action set>)

Condition Action Pair Examples {and Explanations}

(WHEN ALARM arr_facility (i:1..n), status :=busy)
{When the repariman arrives at a facility, he is immediately busy repairing the facility.}

(WHEN ALARM arr_idle, status :=avail)
{When the repairman arrives at the idle location, he is available to repair failed facilities.}

(WHEN ALARM arr_facility (i:1..n), location :=i)
{When the repairman arrives at a facility, his location is that facility.}

Action Cluster Formation

```
WHEN ALARM arr_facility (i:1..n);
    SET ALARM (end_repair (i), neg_exp(mean_repairtime));
    status :=busy;
    location :=i;
    END WHEN
```

FIGURE 3: Illustration of the Relationships in Attribute Definition and Specification.

Figure 3 illustrates the strong similarity of the CS representation to production rule organization in knowledge bases [26, p. 23-33]. Achieving this resemblance was not an underlying motivation; however, the implications of the structural similarities are not being ignored. Consequently, the utility of logic programming within model development is a currently active area of investigation.

The action clusters for the machine repairman model are shown in Figure 4.

```
{ Initialization }
    WHEN Initialization
        INPUT ( n, max_repairs, mean_uptime, mean_repairtime );
        CREATE repairman AN object WITH
            location := idle;
            status := avail;
            END WITH
        CREATE facility A p_set WITH INDEX i :1..n ALSO
            failed[ i ] := false;
            total_downtime[ i ] :=0;
            SET ALARM( failure( i ), neg_exp( mean_uptime ) );
            END WITH;
        DEFINE down A d_set OF facility WITH failed[ i ] = true;
        num_repairs := 0;
        END WHEN
{ Termination }
    WHEN num_repairs ≥ max_repairs
        FOR i := 1 TO n DO
            percent_downtime[i] := total_downtime[i]/system_time;
            OUTPUT(i, total_downtime[i],);
            END FOR
        STOP
        END WHEN
{ Failure }
    WHEN ALARM( failure ( i : 1 .. n ) )
        failed[ i ] := true;
        begin_downtime[ i ] := system_time;
        END WHEN
{ Begin repair }
    WHEN ALARM arr_facility ( i : 1 .. n )
        SET ALARM( end_repair( i ), neg_exp( mean_repairtime ) );
        status := busy;
        location := i;
        END WHEN
{ End repair }
    WHEN ALARM end_repair ( i : 1 .. n )
        SET ALARM( failure( i ), neg_exp( mean_uptime ) );
        failed[ i ] := false;
        total_downtime[ i ] := total_downtime[ i ] +
            ( system_time - begin_downtime[ i ] );
        status := avail;
        num_repairs := num_repairs + 1;
        END WHEN
{ Travel to idle }
    WHEN down = empty & status = avail & location ≠ idle:
        SET ALARM( arr_idle, traveltime( location, idle ) );
        status := travel;
        END WHEN
{ Arrive idle }
    WHEN ALARM( arr_idle );
        status := avail;
        location := idle;
        END WHEN
{ Travel to facility }
    WHEN status = avail & down = NOT empty;
        i := closest_failed_fac( down, location );
        SET ALARM( arr_facility( i ), traveltime
            ( location, i ) );
        status := travel;
        END WHEN
```

FIGURE 4: Machine Repairman Action Clusters

The initialization and termination ACs are required in every model: the first to prescribe initial state definition, the second to stipulate the conclusion of a single simulation experiment. Examination of the remaining ACs reveals a formal specification of the actions taken when:

(1) A facility failure occurs: the "failed" condition of the facility is noted and the downtime recording is begun.

(2) The repairman arrives at a facility to begin repair: an end of repair is scheduled, the repairman is "busy," and his location is at the facility.

(3) The repairman completes the repair of a facility: the next failure of that facility is scheduled, the facility is "operational," the repairman is available, and the number of repairs and total downtime are updated.

(4) No facility requires repair as the repairman completes a repair and becomes available: the repairman travels to the idle position (status is "travel").

(5) The repairman arrives at the idle position: his status is available for repairs and his location is at the idle position.

(6) The repairman completes a repair and finds at least one facility failed: the closest failed facility is determined and the repairman begins travel to that location.

Note that the fourth and sixth ACs described above conform to the definition of contingent events, while the others are determined events (see Nance [24, p. 176]). The event scheduling world view accommodates the latter and the activity scan, the former. The condition specification accommodates both.

A skeleton function specification for the model is shown in Figure 5 to complete the model specification. Note that a report specification [29, p. 197] is omitted for brevity; the use of OUTPUT from the interface specification serves to convey the required information on model behavior. Neither INPUT nor OUTPUT are totally specified.

{ Function	Arguments	Type }
closest_failed_fac	(down : i:1 .. n, location : idle, 1..n)	: 1 .. n
traveltime	(origination : idle, 1 .. n, destination : idle, i .. n)	: positive real
neg_exp	(mean : real)	: positive real

FIGURE 5: Machine Repairman Function Specifications

2.3. The Basis for Model Diagnosis

Our contention is that model diagnosis should not require a significant investment of time and effort beyond that demanded of the modeler for specification and documentation of the model. Consequently, graph-based representations suitable for diagnosis are automatically derivable from a condition specification. The following terms are defined to support that derivation:

> An attribute x is a control attribute of an action cluster if x appears in a condition expression of the action cluster.
>
> An attribute x is an output attribute of an action cluster if the actions can change the value of attribute x.
>
> An attribute x is an input attribute of an action cluster if the value of x affects the output attributes of the action cluster.

The control, input, and output attributes for the machine repairman model are shown in Table 2.

For a condition specification, let

$T = [t_1, t_2, \ldots, t_k]$ be the set of all time-based signals in the CS;

$A = [a_1, a_2, \ldots, a_m]$ be the set of all other attributes in the CS; and

TABLE 2: Attribute Classification for the Machine Repairman Model

Action Cluster (Abbreviation)	Control Attributes	Input Attributes	Output Attributes
INITIALIZATION (in)	initialization	n max_repairs mean_uptime mean_repairtime	i failed failure(i) total_downtime num_repairs location status
TERMINATION (te)	num_repairs max_repairs	system_time total_downtime	total_downtime percent_downtime
FAILURE (fa)	failure(i)	system_time	begin_downtime failed
BEGIN REPAIR (br)	arr_facility(i)	mean_repairtime i	end_repair(i) status location
END REPAIR (er)	end_repair(i)	mean_uptime system_time total_downtime num_repairs begin_downtime	failure(i) failed total_downtime num_repairs status
TRAVEL TO IDLE (ti)	failed status location	location	arr_idle status
ARRIVE IDLE (ai)	arr_idle		status location
TRAVEL TO FAC (tf)	status failed	i arr_facility(i)	i location status

$AC = [ac_1, ac_2, \ldots, ac_n]$ be the set of all action clusters in the CS.

With these definitions, the basis for graph derivations is established.

3. Graph-Based Diagnosis

The preceding development provides the foundation for the graph-based diagnosis, which requires the derivation of two digraphs: (1) an action cluster attribute graph,

and (2)an action cluster incidence graphs. A definition and description of the utility of each is described in the following sections. The diagnostic information derivable from each graph, the emphasis of this paper, is described in some detail.

3.1. Action Cluster Attribute Graphs

The *Action Cluster Attribute Graph* (ACAG) for a CS is defined as follows. Assume CS to be a condition specification with k time-based signal attributes, m other attributes, and n action clusters. Then G, a directed graph with k + m + n nodes, is constructed as follows:

G has a directed edge from node i to node j if

(1) node i is a control or input attribute for node j, an AC, or

(2) node j is an output attribute for node i, an AC.

Thus G is bipartite with one set of nodes representing ACs and the other, attributes. The ACAG for the machine repairman model is omitted for brevity since it represents the information in Table 2 in a different format.

The ACAG depicts the interaction between the ACs and attributes in a model specification, showing both the potential for actions of an AC to change the value of an attribute, and the reverse (an attribute's influence on the actions of an AC). In order to distinguish interactions that occur instantly from those involving a time delay, edges from an AC to a time-based signal (attribute) are depicted with a dashed line; other edges with a solid line.

While the graph is a potential documentation tool, any complex model is likely to have a graph much too large to be directly helpful to a modeler. Examination of Table 2 reveals the tendency of graphs to become cluttered and incomprehensible. The graph, in the form of an appropriate data structure, is primarily useful as input for some of

the analyses discussed below.

3.1.1. Analysis of Action Cluster Attribute Graphs

The ACAG can be used to support several types of model analysis:

(1) Attribute Utilization: if the node representing an attribute has an out-degree of 0, then that attribute cannot influence model behavior.

Note that having an unutilized attribute is not necessarily an error since the attribute may be serving strictly a reporting ("statistical") function, in which case it should appear in some output specification (and this is easily verified).

(2) Attribute Classification: analysis of the graph can identify the function (e.g. control, output, or input) of each attribute in a model specification.

For example, attributes can be classified as "control," if their value change can cause the occurrence of some actions, or as "statistical," if they serve only reporting functions.

(3) Attribute Initialization: if the node representing an attribute has in-degree of 0, then the attribute must be uninitialized.

The model may contain uninitialized attributes other than those with an in-degree of 0. The absence of attribute initialization is a sufficient but not a necessary condition for the existence of uninitialized attributes.

(4) Action Cluster Completeness: if the condition expression for an AC contains attributes which are not time-based signals, then at least one control attribute of the AC must also be an output attribute of the same AC.

To avoid one type of "infinite loop" some action of the AC must eventually make the condition for that AC false.

(5) Revision Consistency: development of a model specification is stages permits the intended use of attributes, described in an earlier incomplete specification, to be checked against the actual usage in a later, more

complete revision.

3.1.2. Matrix Representations

Matrix representations can be constructed from an ACAG, and the manipulation of the matrix data structure has intuitive appeal. The operations described here are similar to those for producing term-document matrices in information storage and retrieval [32, p. 50].

The information contained in an ACAG can be represented by a pair of boolean matrices. Let a CS have m ACs, $ac_1, ac_2, \cdots, ac_m$, and n attributes, $a_1, a_2, \cdots, a_n$. Then the *attribute action cluster matrix* for a CS is an n by m Boolean matrix defined as follows:

$$b(i,j) = \begin{cases} 1 & \text{if edge } (a_i, ac_j) \text{ exists in the ACAG} \\ 0 & \text{otherwise.} \end{cases}$$

The *action cluster attribute matrix* for a CS is an m by n Boolean matrix defined as follows:

$$b(i,j) = \begin{cases} 1 & \text{if edge } (ac_i, a_j) \text{ exists in the ACAG} \\ 0 & \text{otherwise.} \end{cases}$$

3.1.3. Operations on Matrix Representations

If A_1 is an attribute action cluster matrix for a CS and A_2 is an action cluster attribute matrix for the same CS constructed so that the ordering of attributes and ACs corresponds in both matrices, then we can define the *attribute interaction matrix* A as

$$A = A_1 \times A_2.$$

A contains a 1 in position (i,j) if and only if attribute a_i is an input or control attribute for an AC for which a_j is an output attribute. Thus the matrix A indicates the potential for one attribute to directly influence the value of another.

Powers of the matrix A can be used to measure the strength of relationships among attributes and perhaps assist the modeler in decomposing the CS by identifying groups of related attributes. The term "attribute cohesion" is given to the measure of this strength of relationships.

If the order of the matrices in the above multiplication is reversed, then an *action cluster interaction matrix* is formed. A "1" is formed in position (i,j) if and only if ac_i has an output attribute which is an input attribute of ac_j. Thus an action cluster interaction matrix indicates the ability of one AC to directly influence another.

Powers of an action cluster interaction matrix can be used to measure strength of relationships among ACs and perhaps guide the modeler in the decomposition of the CS. The term "action cluster cohesion" is used. Bauer, et. al. [2] demonstrates the utility of this matrix in a recent paper describing the application of factor analysis to effect model decomposition.

3.2. Action Cluster Incidence Graphs

The Action Cluster Incidence Graph (ACIG), is useful in the analysis of model specifications, in depicting a particular type of interaction among ACs. The idea is simple: if an action of AC_i has the potential to cause AC_j to occur (by changing the value of a control attribute), then a directed edge leads from AC_i to AC_j. As before, a solid edge indicates that AC_i can cause AC_j to occur in the same instant of simulated time and a dashed edge indicates the potential occurrence only after a time delay (through a time-based signal).

The automatic construction of an ACIG from a CS is easily accomplished following the procedure given below. However, this procedure produces a graph which may include many potential edges. That is, an edge from AC_i to AC_j might be constructed although the application of additional knowledge would reveal that AC_i could never cause AC_j to occur.

Potential edges removed by the application of additional knowledge are termed infeasible. Overstreet [28, p. 271] shows that no algorithm can exist to produce from a model specification an ACIG which never includes infeasible edges. Even so, in many cases the number of infeasible edges is not excessive.

An ACIG for a CS consisting of a set of ACs $\{ac_1, ac_2, \cdots, ac_n\}$, can be constructed as follows:

(1) For each $1 \leq i \leq n$, let node i represent ac_i

(2) For each ac_i, partition the attributes into three sets:
$T_i^i = \{$control attributes which are time-based signals$\}$
$C_i^i = \{$all other control attributes$\}$
$O_i^i = \{$output attributes$\}$

(3) For each $1 \leq i \leq n$,
For each $1 \leq j \leq n$,
Construct a solid edge from node i to node j if $O_i \bigcap C_j \neq 0$
Construct a dashed edge from node i to node j if $O_i \bigcap C_j \neq 0$.
END for each $1 \leq j \leq n$
END for each $1 \leq i \leq n$.

For the machine repairman model, the ACIG produced by the above algorithm is show in Figure 6. Note the prevalence of immediate change edges in the graph. Some of these potential edges can be classed infeasible by employing knowledge-based analysis in a manner described in Section 3.2.2.

3.2.1. Analysis of Action Cluster Incidence Graphs

In addition to providing documentation of possible component interactions, the ACIG can assist in several interesting analyses. The graph itself can be analyzed for the following properties:

(1) Connectedness: if, for a given condition specification, no ACIG exists which is not connected, then the CS is said to be connected.

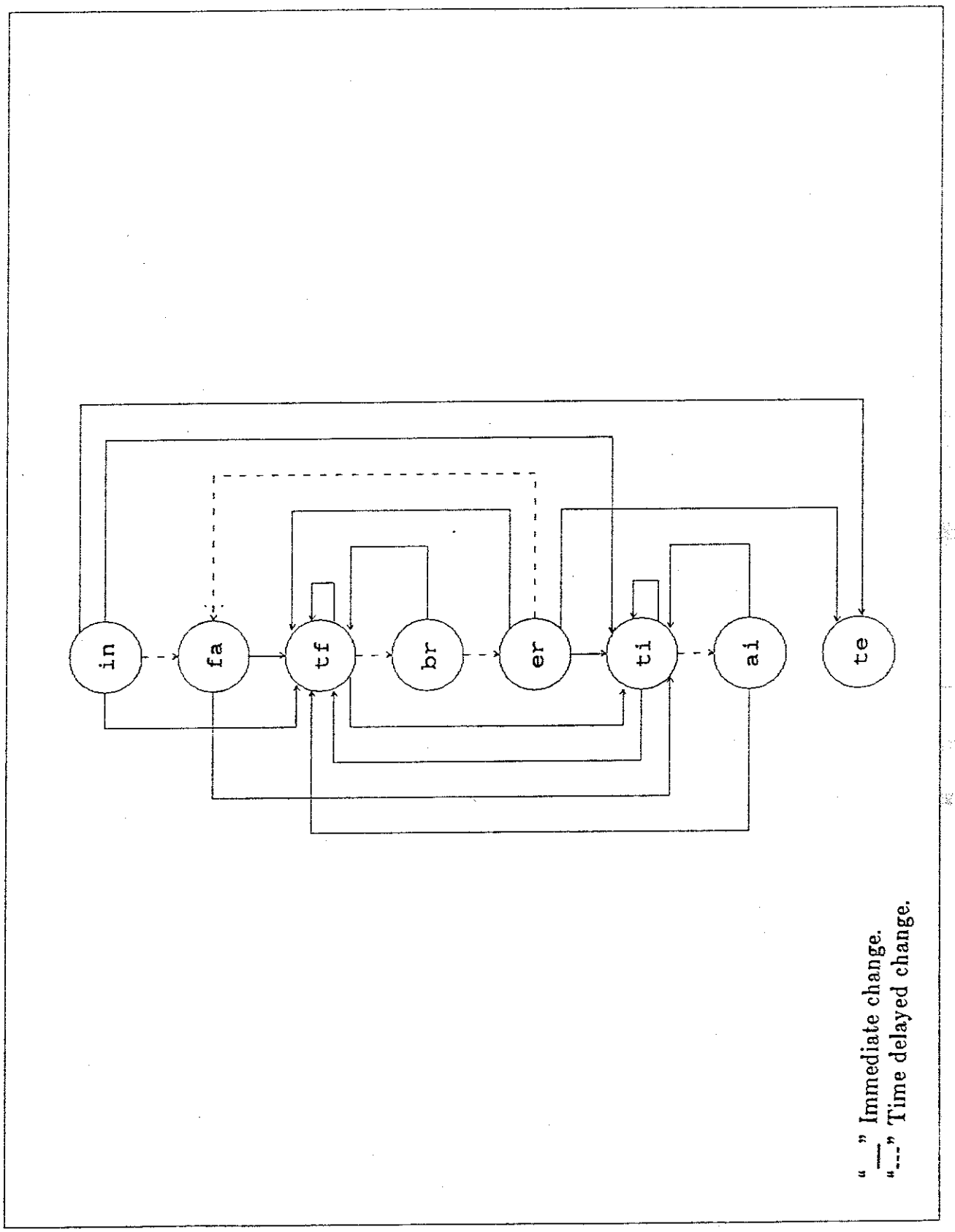

FIGURE 6: Machine Repairman Action Cluster Incidence Graph

If a CS has an ACIG which is not connected, then the model specification contains components which cannot influence the actions of other components. Consequently, the specification consists of two or more independent submodels. The property can be extended in a useful fashion. If the initialization AC for a CS is the only node connecting components, then the specification contains non-interacting components. While lack of connectedness is not necessarily an error, knowledge that a CS has this property may be useful to the modeling team.

(2) Accessibility: if, for a given condition specification, no ACIG contains nodes with an in-degree of zero (other than the Initialization node) then the CS is said to be accessible.

If a CS is not accessible, then it contains ACs which can never be activated in any subsequent execution. These ACs can be deleted from the model specification with no effect, a result unlikely to be intentional.

(3) Out-complete: if, for a given condition specification, no ACIG contains nodes with an out-degree of zero (other than the termination node), then the CS is said to be out-complete.

If a CS is not out-complete, then it contains ACs which can be deleted from the model specification without affecting model behavior. The inclusion of ACs with an out-degree of zero is not necessarily an error. The purpose of such an AC might be to report model behavior. In this case the AC should contain some kind of output operation, which is readily recognizable.

3.2.2. Extended Uses of Action Cluster Incidence Graphs.

The analyses described above require only direct operations on ACIGs. More extensive uses require additional tools, such as simplification techniques, defined metrics, and interface utilities. The knowledge-based simplification of an ACIG, alluded to above, illustrates the capability for modeler assistance.

3.2.2.1. Graph Simplification

Examination of the ACIG for the machine repairman model reveals several infeasible edges among the potential edges shown in Figure 6. For example, the edge from INITIALIZATION (in) to TERMINATION (te) can be deleted as infeasible since model termination immediately following initialization is not feasible (the comparison of the value of "num_repairs" with that of "max_repairs" must permit some non-zero simulated time period).

Coupling expert knowledge of simulation analysis with that of the application domain enables further recognition of infeasible edges. Table 3 provides a summary of the results of the simplification procedure, which at this juncture is applied through interaction with the modeler.

Each action cluster in the model is typed as "contingent" (the condition is not strictly a time-based signal) or "determined." Each potential successor is examined to determine its feasibility. Infeasible edges are deleted, based on arguments briefly summarized in Table 3, and the resulting graph is presented in Figure 7. A more detailed explanation of the simplification procedure is given in [28, pp. 199-214].

Three significant points arise from the examination of Table 3 and Figures 6 and 7:

(1) While automatic graph construction is easy, the simplification necessary to provide useful diagnostic assistance can be involved. Representing the knowledge required to perform edge deletion can range from the almost trivial (termination can never immediately follow initialization) to the more challenging (a travel to the idle position can only follow an end of repair and if the set "down" is empty).

(2) The knowledge required for simplification is both generic to simulation (the first example in (1)) and specific to the application domain (the second example).

(3) The level of descriptive detail furnished by the specification language affects the difficulty in acquiring and representing the required knowledge. For example, the specification {down = NOT empty} is equivalent to

TABLE 3: Summary of the Graph Simplification for the Machine Repair Model

Action Cluster (Abbreviation)	Type of Edge[1]	Potential Successors: Output Attributes	Basis for Deletion of Edge	Feasible Edges d	Feasible Edges c
INITIALIZATION (in)	c	te: num_repairs max_repairs	Termination immediately following initiation indicates an error.		
	c	tf: failed status	Violates assumption of all facilities initially operating		
	c	ti: location failed status	Violates assumption that initially repairman is at idle position		
	d	fa: failure(i)	Not deleted	fa	
TERMINATION (te)	.	None			
FAILURE (fa)	c	ti: down=empty (NOT failed[i]Vi:1..n)	Repairman does not go to idle position except after ending a repair and finding no facilities failed		
	c	tf: failed	Not deleted		tf
BEGIN REPAIR (br)	c	tf: status failed	Repairman must end a repair		
	c	ti: status location	Repairman must end a repair		
	d	er: end_repair(i)	Not deleted	er	
END REPAIR (er)	c	ti: down=empty (NOT failed[i]Vi:1..n) status	Not deleted		ti
	d	fa: failure(i)	Not deleted	fa	
	c	te: num_repairs	Not deleted		te
	c	tf: down=NOT empty (failed[i] for some i:1..n) status	Not deleted		tf
TRAVEL TO IDLE (ti)	c	ti: status	Repairman does not travel to idle immediately following a travel to idle		
	c	tf: status	Repairman must arrive at the idle position before identifying a failed facility		
	d	ai: arr_idle	Not deleted	ai	
ARRIVE IDLE (ai)	c	ti: status location	Repairman leaves idle position only to repair a failed facility		
	c	tf: location	Not deleted		tf
TRAVEL TO FAC (tf)	c	ti: status	Repairman travels to facility only to begin a repair		
	d	br: arr_facility(i)	Not deleted	br	

[1] Edge types: d = determined (strictly a time-based signal); successor occurs following some delay
c = contingent (state or state and time-based); successor could occur immediately
. = none (no edge).

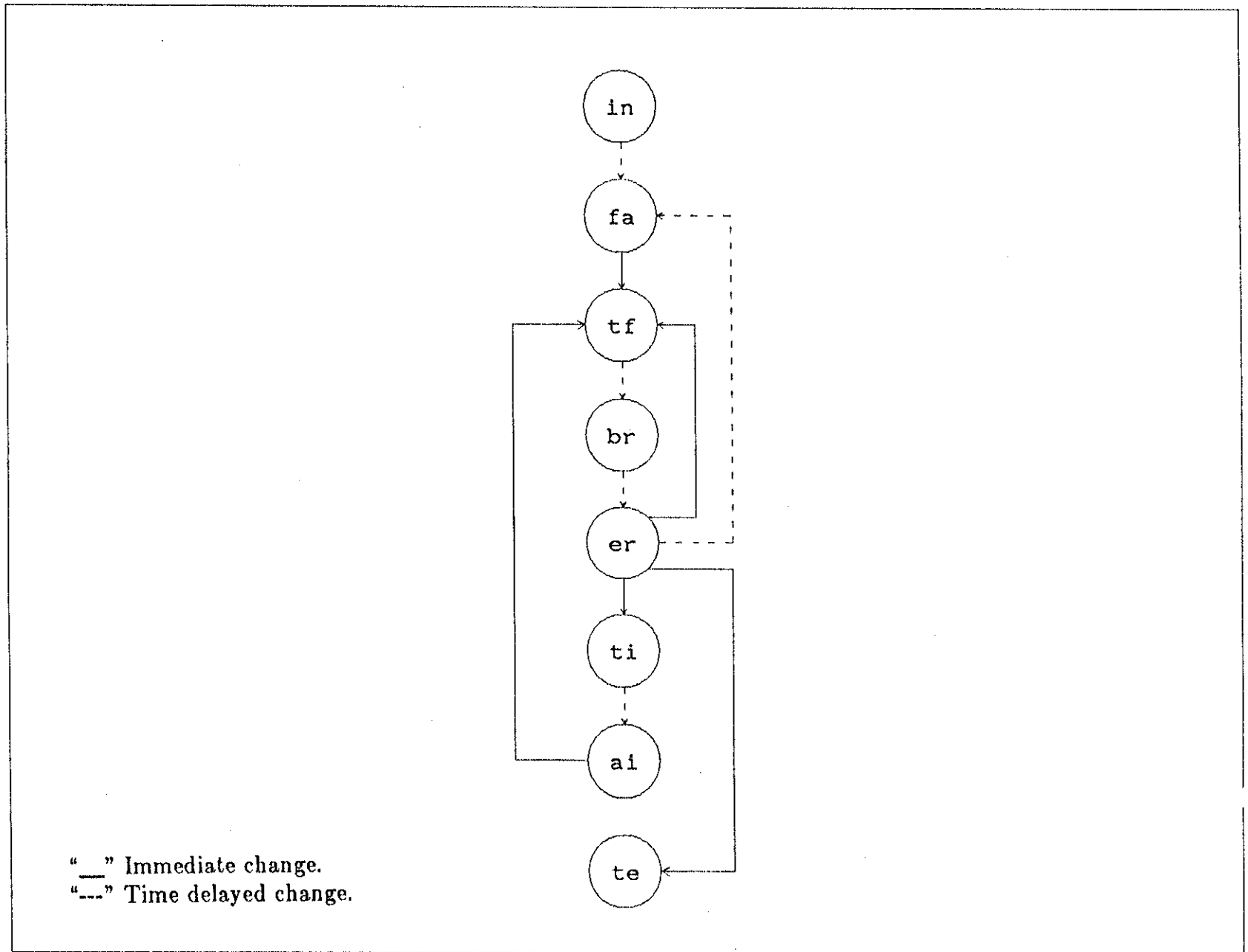

FIGURE 7: Simplified Repairman Action Cluster Incidence Graph

{failed [i] for some i:1..n}: but, while the latter reflects a direct relationship between the END REPAIR and TRAVEL TO FACILITY action clusters, the former is indirect, forcing a derivation.

The introduction of expert systems into model development environments to provide diagnostic assistance is clearly accommodated by the condition specification and the semantic structure of condition action pairs. However, the exploitation of the modeler's intelligence through diagnostic assistance should not be overlooked in the rush toward totally automated support.

3.2.2.2. Structural Evaluation

The structure of an ACIG can be diagnosed for other characteristics, three of which are briefly described in this section.

(1) Complexity: no definition of complexity is offered here; it is a property not easily defined. However, we propose that an ACIG for a model specification can be analyzed to suggest, at least to a relative if not an absolute degree, both anticipated run-time overhead and difficulty in implementation, verification/validation, and maintenance.

Several graph-based metrics can be proposed which, at least at an intuitive level, relate to the complexity of the underlying model. Several authors have proposed graphical related metrics for the study of software complexity. See [20, 37, 12] for some proposed metrics and [10, 7, 13] for evaluations of these metrics.

(2) Precedence structure: An ACIG, by the nature of its construction, conveys something about the sequencing of model actions. For example, if a node has in-degree one, then its occurrence in any execution based on the CS must always be preceded by the AC with an edge to that node.

For each AC, the ACIG provides both a set of predecessor ACs and a set of successor ACs. Although no algorithm is likely to detect errors in these sets, the predecessor and successor sets are interesting diagnostic tools. Such sets presented to one who understands the behavior of the simulated system, could enable that individual to recognize that anticipated

successors are missing or unexpected successors are included.

(3) Decomposition: While the ACIG depicts interaction among ACs, the approach can be used for components other than ACs. An incidence graph for decompositions of a model into structures other than ACs can be utilized.

For example in [29], this approach is used to analyze alternate world view representations of a single CS. Utilizing the action cluster interaction matrix, derived from the ACIG, as the basis for a factor analytic decomposition technique is cited above [2].

4. Diagnostic Assistance in the Model Development Environment

In Section 1 three categories of diagnostic assistance are defined. Collectively, the categories span a continuum from manual to automated techniques. Analytical diagnosis includes the most readily automated; while the informative category represents the most knowledge intensive. Table 4 utilizes this classification scheme to provide a summary of the graph-based techniques described herein.

The summary provided in Table 4 assists in emphasizing three points with regard to this research:

(1) The categorization of diagnostic assistance clarifies the extent to which the responsibility for detecting problems in a model specification is shared between diagnostic tools and the modeler.

(2) While analytical diagnosis produces the most definitive statements about the model specification, informative diagnosis may present the highest utility/cost tradeoff, especially in the short term.

(3) Within the categories of comparative or informative diagnosis, precise measures and completely defined techniques cannot be recommended. Only by empirical investigation can proposed measures and techniques be evaluated.

5. Summary

Graph-based diagnostic analysis offers considerable promise for the expansion of

TABLE 4. Categorized Summary of Diagnostic Assistance

Category of Diagnostic Assistance	Properties, Measures, or Techniques Applied to the Condition Specification (CS)	Basis for Diagnosis
1) Analytical: Determination of the existence of a property of a model representation.	a) Attribute Utilization: No attribute is defined that does not affect the value of another unless it serves a statistical (reporting) function.	Action Cluster Attribute Graph (ACAG)
	b) Attribute Initialization: All requirements for initial value assignment to attributes are met.	ACAG
	c) Action Cluster Completeness: Required state changes within an action cluster are possible	ACAG
	d) Attribute consistency: Attribute Typing during model definition is consistent with attribute usage in model specification.	ACAG
	e) Connectedness: No action cluster is isolated.	Action Cluster Incidence Graph (ACIG)
	f) Accessibility: Only the initialization action cluster is unaffected by other action clusters.	ACIG
	g) Out-complete: Only the termination action cluster exerts no influence on other action clusters.	ACIG
	h) Revision Consistency: Refinements of a model specification are consistent with the previous version.	ACAG
2) Comparative: Measures of differences among multiple model representations.	i) Attribute cohesion: The degree to which attribute values are mutually influenced.	Attribute Interaction Matrix (originates with the ACAG)
	j) Action Cluster cohesion: The degree to which action clusters are mutually influenced.	Action Cluster Interaction Matrix (originates with the ACAG)
	k) Complexity: a relative measure for the comparison of a CS to reveal differences in specification (clarity, maintainability, etc.) or implementation (run-time overhead) criteria.	ACIG
3) Informative: Characteristics extracted or derived from model representations.	l) Attribute Classification: Identification of the function of each attribute (e.g. input, output, control, etc.)	ACAG
	m) Precedence Structure: Recognition of sequential relationships among action clusters.	ACIG
	n) Decomposition: Depiction of subordinate relationships among components of a CS.	ACIG

human problem-solving ability to deal with large, complex simulation models. The ACAG and ACIG, and their graphical and matrix derivations, represent a source of much needed automated and partially automated assistance in the model development task. The graph construction and simplification procedures provide concrete evidence of the potential for expert systems applications in simulation modeling. However, the need to recognize the significant progress to be achieved through assistance in extending the modeler's intelligence is

asserted and reinforced by the taxonomy of diagnostic assistance presented. In the short- to intermediate-term the high payoffs may be found in the category of informative diagnosis.

ACKNOWLEDGEMENTS

The authors acknowledge the many stimulating discussions with Osman Balci, whose influence is no doubt reflected in this work. Also, the extensive helpful guidance of the referees in the revision of an earlier version is acknowledged. The careful, laborious typing of manuscript versions by Karen Kaster is gratefully appreciated.

REFERENCES

[1] Balci, O. Requirements for model development environments. *Computers & Operations Research 13,* 1 (Jan.-Feb. 1986).

[2] Bauer, K.W., Kochar, B, and Talavage, J.J. "Simulation model decomposition by factor analysis." In *Proceedings Winter Simulation Conference*, (San Francisco, CA, December 1985), pp. 185-188.

[3] Burns, J.R. and Winstead, W.H. M-Labeled digraphs: An aid to the use of structural and simulation models. *Management Science 31,* 3 (March 1985), pp. 343-357.

[4] Chen, P.P. The entity-relationship model--Toward a unified view of data. *ACM Transactions on Database Systems 1,* 1 (March 1976), pp. 9-36.

[5] Clementson, A.T. *Extended Control and Simulation Language/Computer Aided Programming System, Detailed Reference Manual*, The University of Birmingham, Birmingham, England, April 1978.

[6] Cox, D.R. and Smith, W.L. *Queues*, Methuen and Company, Ltd., 1961.

[7] Curtis, B., Sheppard, S.B, Milliman, P., Borst, M.A., and Love, T., Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. *IEEE Transactions on Software Engineering SE-5*, 2 (March, 1979), pp. 96-104.

[8] Davies, N.R. "A modular interactive system for discrete event simulation modeling." In *Proceedings Ninth Hawaii International Conference in System Sciences*, Western Periodical Company, (January 1976), pp. 296-299.

[9] DeCarvalho, R.S. and Crookes, J.G. Cellular simulation. *Operational Research Quarterly 29*, 1 (1976), pp. 31-40.

[10] Elshoff, J.L., and Marcotty, M. On the use of the cyclomatic number to measure program complexity. *SIGPLAN Notices 13*, 12 (December 1978), pp. 29-40.

[11] Gordon, G. The development of the general purpose simulation system (GPSS). In

History of Programming Languages. Richard L. Wexelblat ed., Academic Press, 1981, pp. 403-426.

[12] Henry, S. and Kafura, D. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering SE-9*, 5 (September 1981), pp. 510-518.

[13] Henry, S. and Kafura, D. "On the Relationships Among Three Software Metrics." Proceedings 1980 Symposium on Software Quality Assurance, in *Performance Evaluation Review 10*, 1 (1981), pp. 81-88.

[14] Howden, William E. Contemporary software development environments. *Communications of the ACM 25*, 5 (May 1982), pp. 318-329.

[15] Kiviat, P.J. Digital computer simulation: Modeling concepts. RAND Memorandum RM-5378-PR, August 1967.

[16] Kiviat, P.J. Digital computer simulation: Computer programming languages. RAND Memorandum RM-5883-PR, January 1969.

[17] Markowitz, H.M. SIMSCRIPT: Past, present, and some thoughts about the future. In *Current Issues in Computer Simulation,* N.R. Adam and A. Dogramaci, eds., Academic, New York, 1979, pp. 27-60.

[18] Markowitz, H.M., Malhotra A., and Pazel D.P. The EAS-E application development system: Principles and language summary. *Communications of the ACM 27*, 8 (August 1984).

[19] Matthewson, S.C. "Interactive simulation program generators." In *Proceedings of the European Computing Conference on Interactive Systems*, Brunel University, 1975.

[20] McCabe, T.J. A complexity measure. *IEEE Transactions on Software Engineering SE-2*, 4 (December 1976), pp. 308-320.

[21] Mealy, G.H. "Another look at data." In *Proceedings Fall Joint Computer Conference*, 1967, pp. 525-534.

[22] Nance, R.E. "Model representation in discrete event simulation: The conical methodology". Technical Report CS81003-R, Department of Computer Science, Virginia Tech, Blacksburg, March 15, 1981.

[23] Nance, R.E., Mazaache, A.L., and Overstreet, C.M. "Simulation model management: Resolving the technological gaps." In *Proceedings Winter Simulation Conference*, (Atlanta, GA), December 1981, pp. 173-179.

[24] Nance, R.E. The time and state relationships in simulation modeling. *Communications of the ACM 24*, 4 (April 1981), pp. 173-179.

[25] Nance, R.E., "A tutorial view of simulation model development." In *Proceedings Winter Simulation Conference*, (Arlington, VA), December 1983, pp. 325-331.

[26] Newell, A. and Simon, H.A. *Human Problem Solving*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1972.

[27] Oren, T.I. "Computer aided modelling systems." In *Progress in Modelling and Simulation*, F.E. Cellier, ed., Academic Press, London, 1982.

[28] Overstreet, C.M. "Model specification and analysis for discrete event simulation." Ph.D. Dissertation, Department of Computer Science, Virginia Tech, Blacksburg, VA, December 1982.

[29] Overstreet, C. M. and Nance, R.E. A specification language to assist in analysis of discrete event simulation models. *Communications of the ACM 28*, 2 (February 1985), pp. 190-201.

[30] Overstreet, C.M. and Nance, R.E. World view based discrete event model simplification. In *Modeling and Simulation Methodology in the Artificial Intelligence Era*, M.S. Elzas, Oren, T.I., and Zeigler, B.P. (eds), North-Holland, to appear.

[31] Palm, D.C. The distribution of repairmen in servicing automatic machines. (Swedish), Industritidningen Norden, Vol. 175, p. 75, 1947.

[32] Salton, G., *Automatic Information Organization and Retrieval*, McGraw-Hill, New York, NY, 1968.

[33] Schruben, L. Simulation modeling with event graphs. *Communications of the ACM 26*, 11 (November 1983), pp. 957-963.

[34] Tocher, K.D. and Owen D.G. "The automatic programming of simulations." In *Proceedings Second International Conference on Operational Research*, 1960, pp. 50-68.

[35] Vidallon, C. GASSNOL: A computer subsystem for the generation of network oriented languages with syntax and semantic analysis. *Simulation '80*, (Interlaken, Switzerland, June 25-29), 1980.

[36] Wasserman, A.I. *Tutorial: Software development environments*, IEEE Computer Society Press, MD, 1981.

[37] Woodward, M.R., Hennell M.A., and Hedley D. A measure of control flow complexity in program text. *IEEE Transactions on Software Engineering SE-5*, 1 (January 1979), pp. 45-50.

[38] Zeigler, B.P. *Theory of Modeling and Simulation*, Wiley, New York, 1976.

[39] Zeigler, B.P. *Multifacetted Modelling and Discrete Event Simulation*, Academic Press, New York, 1984.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: SRC-86-001 (also TR-86-8 Dept of CS)
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): Diagnostic Assistance Using Digraph Representations of Discrete Event Simulation Model Specifications

5. TYPE OF REPORT & PERIOD COVERED: Interim Report for period June '85 to March '86.

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): C. Michael Overstreet, Richard E. Nance

8. CONTRACT OR GRANT NUMBER(s): N60921-83G-A165 B012

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Systems Research Center and Dept. of C.S., Virginia Tech, Blacksburg, Virginia 24061

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Sea Systems Command, SEA61E, Washington, D.C. 20362

12. REPORT DATE: 26 March 1986

13. NUMBER OF PAGES: 34

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Naval Surface Weapons Center, Dahlgren, VA 22448

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This report is distributed to scientists and engineers at the Naval Surface Weapons Center and is made available on request to other Navy scientists. A limited number of copies is distributed for peer review.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Engineering, testing and debugging, diagnostics, Simulation and Modeling, model validation and analysis, model development, model specification, model diagnosis, computer assisted modeling, automated model analysis.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Automated diagnosis of digraph representations of discrete event simulation models is illustrated as an effective model verification technique, applicable prior to coding the model in an executable language. The Condition Specification is shown to provide an effective representation, from which automated analysis can initiate with a digraph extraction. Subsequent diagnostic simplification techniques are applied to the digraph, either automatically or in concert with the modeler. Three categories of diagnostic assistance are defined: analytical,

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

comparative, and informative. The categories represent diagnostic techniques that are progressively more difficult to automate. Examples of techniques assigned to each category are included, and a concluding caution is made that diagnostic techniques requiring modeler interaction should not be ignored in the preoccupation with complete automation.