

# Discrete event simulation model reduction: A causal approach

Leonardo Chwif<sup>a,\*</sup>, Ray J. Paul<sup>b</sup>, Marcos Ribeiro Pereira Barretto<sup>a</sup>

<sup>a</sup> *CepilUnifeco, R. Narciso Sturlini, 883 – Osasco, São Paulo 06018-903, Brazil*

<sup>b</sup> *Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex UB8 3PH, United Kingdom*

Received 12 February 2005; received in revised form 5 September 2005; accepted 10 May 2006

Available online 22 September 2006

---

## Abstract

Discrete event simulation is an important system analysis technique. But for today's demand for speed, the time to complete a simulation study is considered to be long, even with current developments in hardware and simulation software. In this scenario, simplification methods for simulation models could play a key role. This paper proposes a technique for reducing the complexity of a discrete event simulation model at the conceptual phase of simulation modeling that can be fully automatized through a computer program. We applied this technique on some problems which demonstrate the feasibility of this approach.

© 2006 Elsevier B.V. All rights reserved.

*Keywords:* Discrete event simulation; Model reduction; Simulation model simplification

---

## 1. Introduction

Current simulation studies are being performed by means of more complex models. This is allowed by the development of powerful computer hardware and software (a more detailed discussion of some of these factors is provided in [1]). Complex and huge simulation models forces modelers to face problems they were not used to, referred by Page et al. [2] as “Problems of Scale”.

Despite of the growth of more complex models, several authors reinforced the importance of simpler simulation models [3–9]. Salt [10] asserts that “simplification is the essence of simulation” and Pidd [11] is conclusive in his declaration: “Complicated models have no divine right of acceptance”.

Unfortunately according to Brooks and Tobias [12,13] there is a scarcity of simulation research in the field of simplification of simulation models. These authors also mentioned that the majority of work in this field does not constitute a general methodology to simplify a given simulation model.

The aim of this paper is to propose a technique for reducing the complexity of simulation model at the conceptual phase of simulation i.e. when the results of simulation runs are yet known.

---

\* Corresponding author. Tel.: +55 11 50817472; fax: +55 11 50625414.  
E-mail address: [leochwif@usp.br](mailto:leochwif@usp.br) (L. Chwif).

This paper is organized as follows: Section 2 makes a brief review in literature regarding model reduction or simplification. Section 3 defines the reduction problem; Section 4 gives a brief explanation on the chosen model representation technique for the reduction process. In Section 5, we explain how the reduction algorithm works, and in Section 6, we present a simple application example while in Section 7 a more complex example is shown. Finally, Section 8 concludes this paper, adding directions for future work.

## 2. Literature review

“Simplicity” is an issue extensively studied in several fields, being philosophically discussed in [14] in his famous book “The Myth of Simplicity”. It took different forms in Science and Engineering: examples of reduction processes in mass-spring systems and in control theory are found respectively at [15,16].

In the area of discrete event simulation, as mentioned before, there is a scarcity of research. In fact, according to Sevinc [17]: “No complete theories of model abstraction exist, nor does any sufficiently general procedure”. The field, with only less than half a dozen published articles, is wide open inviting the attention of simulation methodologists”.

A consensual name for this subject has yet to be defined. This work adopted the expression “Model Reduction”, but in literature it can be also founded under names as “Simulation Model Abstraction” [17–19] and “Simulation Model Simplification” [12,13].

According to Brooks and Tobias [13], a simplification method can be divided into three categories: coding tricks, simplifications that preserve the output of interest and simplification that preserve the output approximately. For the sake of quick referencing, we will name here these kinds of simplification as type I reduction, type II reduction and type III reduction.

One of the firsts attempts to find a reduction method is attributed to Zeigler [20]. Others also propose some methods [4,5,8,21]. The majority of these general rules could be classified into three categories of simplification [9]: Omission, Aggregation and Substitution. Courtois [22] discussed some matters related to time scale, which can lead to possible decompositions of simulation models. Sevinc and Foo [23] and Sevinc [17] have developed a simulation model reduction procedure, based on the DEVS developed by Zeigler [20,24]. However, there was a need to run the model to obtain a result for the reduction procedure. McGraw and MacDonald [19] proposed also some techniques for reducing the complexity for engineering and engagement level simulations while Brooks and Tobias [13] proposed them in the realm of Manufacturing Simulation. Nevertheless again these constitutes only some “rules of thumb” for reducing the complexity of simulation models.

Another technique considered by some authors as reduction process of simulation model is metamodeling. In this case, a mathematical model of a simulation model is created in order to imitate its input–output behavior. Barton [25] does a metamodel review in literature and Caughlin [26] presents new procedures for the generation of metamodels. Some new procedures utilize neural network technology to do the input–output correlation [27]. Metamodeling could be considered as reduction type III since the generated input–output behavior is an approximation of the original one.

It is also possible to achieve reduction by changing a modeling paradigm. Yan and Gong [28] proposed to model high-speed networks using discrete event simulation in conjunction with fluid simulation (a technique they called “time-driven fluid simulation”). This approach resulted in “good enough” approximations in feasible computational times in specific cases, which would be impractical if only discrete event simulation was used. Nance et al. [29] demonstrate also that a methodology can induce redundancy, and propose methods for eliminating some redundancies on ACIG graph derived from Condition Specification (see Section 4 and Appendix A.2 for a small review on CS). The ACIG graph is a mapping of attributes of the condition specification, and [29] show that by cutting unnecessary edges of the ACIG graph and executing a simulation directly from it using a direct execution algorithm, it is possible to reduce the computational execution time up to three times in some test problems.

An additional related issue to model reduction relies on how one can measure complexity of the simulation models by an objective measure. Even with an agreed measure, the definition of complexity has no unanimity, as pointed out by Brooks and Tobias [12]. According to Brooks and Tobias [13], however, it is possible to consider complexity of a simulation model by analyzing the amount of components of a model and the amount of connections.

### 3. The reduction problem

The core problem of model reduction is to find a simpler conceptual model that is still valid, with respect to the *Simulation Requirements*. Simulation requirements are the objectives of the simulation study, mainly expressed by desired measures of performance. There could be two approaches to reach to a simpler model: a constructive or evolutive. For the “*constructive*” approach, a simpler model would emerge directly from scratch. For the “*evolutive*” one, a simpler model should be derived from an initial one. For this last case, still there are two possibilities:

- (1) The initial model is more complex, and we attempt to simplify it. This may be called Evolutive/Reductionist approach.
- (2) We produce an initially an over-simplified model and then we start to add necessary details. This may be called Evolutive/Expansionist approach.

In our point of view a proper model reduction technique must have the following characteristics:

- (1) *Be dependent upon simulation objectives or requirements*: a model is mainly created to answer a question at hand and hence one of the inputs of any reduction procedure should be the objectives or simulation requirements that we would like to address.
- (2) *Be performed at conceptual phase of simulation modeling*: since we would like to minimize the total development time of a simulation model, it would be worth that the reduction should be applied at the earliest stages of simulation modeling, i.e. conceptual model, implying we are not able to run the model before the reduction process.
- (3) *Be guaranteed the validity of the simpler model*: in this case, the procedure should be able to maintain the simpler model’s validity (in statistical terms) at least to the simulation requirements chosen; Ideally it would lead to type II reductions and in the worst case type III reductions.
- (4) *Be able to be performed automatically on a computer*: since the main objective of model reduction is to cut down the cycle time of a simulation study, it is not worth it if this process takes considerable time. So this process must be capable of being automated in order that its duration, when compared to all simulation cycle time, becomes negligible.

In order to apply a reduction technique at the conceptual phase of simulation modeling, the simulation model should be represented into a proper model representation technique. So, this is the issue that we are going to discuss in next section.

### 4. Simulation model representation technique

In order to apply a model reduction procedure the Simulation Model should be represented in a proper Simulation Model Representation Technique. There are several techniques for simulation model representation found in the literature regarding discrete-event systems. We could divide these techniques into two categories: diagrammatic (or graphical) methods and formal specification languages.

As graphical languages, we can cite Activity Cycles Diagrams [30–32], Control Flow Graphs [33], Event Graphs [34], Petri Nets [35] and its extensions [36], State Charts [37] and Process Networks [38].

As formal specification languages, one of the most cited is Discrete Event Systems Specification (DEVS) from [20], based on the systems theoretic approach. Another example is the Condition Specification (CS) created by Overstreet and Nance [39], based on the “Conical Methodology”.

By analysing several model representation techniques we found that the one suitable to a simplification technique due to the ability to represent all model details is the Condition Specification. However this representation is difficult to construct from scratch ([40] also pointed out that the Condition Specification demands some buffering mechanism between the representation and the analyst). So we will consider the Activity Cycle Diagram a “higher level modelling” to be build first and then after to be converted into Condition Specification. To allow this conversion, we have developed a series of automatic rules to convert ACD into Condition

Specification. It is not scope of this paper to cover these conversion rules. More detail is provided in [41]. Appendix A reviews the basic principles of the Activity Cycle Diagrams and the Condition Specification. In the next section, we will be presenting our reduction algorithm based on the defined reduction problem to be applied on models represented into Condition Specification format.

**5. Reduction algorithm – “backtracking”**

The basic idea of the reduction algorithm is to follow the influences of the attributes in the Condition Specification on the defined simulation measure of performance. The concepts of attributes (control, output and input) were briefly explained in Appendix A.2. The reduction algorithm will directly affect the *Transition Specification* (since this is the key specification which determines the behavior of the model). From its reduction, the reduction of *Interface Specification* and *Object Specification* is obvious. Therefore, we will concentrate on *Transition Specification*.

Hence, starting by the *Transition Specification* of a simulation model (converted or not from the Activity Cycle Diagram), one must write down all its Condition Action Pairs (CAP’s) in a table, numbering each CAP in an ascending order. We obtain in this case the *Transition Specification* in a “tabular format”. An example of this can be seen in Table 1.

We should construct a second table named “backtracking table”. This table has three columns. The first one contains the CAP number (according to the numbering given in the previous table). The interest attribute (that we would like to backtrack) is present in column 2 and finally in column 3 there must be cited all attributes that could influence the attribute in column 2. The basic configuration of this table’s headers is shown in Table 2.

The reduction procedure starts by the identification of one attribute that represents some desired performance measure, like utilization, throughput, waiting times, etc. From this attribute, the algorithm will start to “backtrack”, or in other words, start to search for others attributes that can influence this. That is the reason for the name of the algorithm. The algorithm consists of two main phases and utilizes the concepts of *Output Attribute*, *Input Attribute* and *Control Attribute* of *Transition Specification*. This is briefly explained in Appendix A, but for further details, you should refer to [42]. These phases are described below.

Table 1  
Example of a transition specification in a tabular format

#	Action cluster name	Condition	Actions
10	{initialization}	INITIALIZATION	Qidle.tail := 0
20	{initialization}	INITIALIZATION	Qtidle.head := 0
30	{initialization}	INITIALIZATION	Qtidle.size := 0
40	{initialization}	INITIALIZATION	cranewt[i] := 0
...	...	...	...
...	...	...	...
250	{fill}	Qtorp.size > 1 and Qcrane.size > 0 and pitt	Qtorp.head := Qtorp.head + 1
...	...	...	...

Table 2  
Headers of the backtracking table

Cap# (1)	Backtracking of: (2)	Influenced by: (3)
----------	----------------------	--------------------

*Phase 1 (Search Phase):*

- (1) Identify the attribute of interest and search (over Transition Specification) one CAP that contains it as an *Output Attribute*. Write into the backtracking table the number of this CAP (column 1) and the name of this attribute (column 2).
- (2) Write down to the backtracking table (column 3), all attributes in the model that can influence the attribute identified in step 1 in this phase. The attributes that can influence it are the *Input Attributes* and *Control Attributes* for this respectively CAP.
- (3) Repeat Steps 1 and 2 of Phase 1 until there is no more CAP left that contains this interest attribute as an *Output Attribute*.

*Phase 2 (backtracking phase):*

- (1) Select an attribute that was not previously selected (i.e. does not appear in column 2) from the list of attributes that are in column 3. If this attribute is an exogenous attribute (that represents other performance measure), it should be marked (with an asterisk for example) to indicate a terminal attribute (that does not influence another attribute), and select another from the list. Copy this attribute to column 2.
- (2) Repeat Phase I for this new attribute.

*Termination:*

If there are no more attributes left in column 3 to backtrack, the algorithm stops. Thus, all the CAP's identified by this procedure (and consequently its attributes) are vital to compute the performance of interest and could not be removed from the *Transition Specification*.

As said before, the procedure begins with the choice of one (or more) performance measure as the attribute of interest. The backtracking algorithm can also begin with more than one attribute of interest. In this case, the backtracking table should be initialized with all these attributes simultaneously.

All remaining CAP's that were not captured by the backtracking algorithm and that not contains any command of the kind FOR, NEXT, STOP, CREATE/DESTROY, INPUT, OUTPUT could be removed from the *Transition Specification* without compromising simulation objectives. Now we have to rebuild the *Transition Specification* from our backtracking table. For those specific CAP's, the procedure is the following:

- (1) STOP: the CAP that contains the command STOP could not be eliminated since it indicates termination of simulation.
- (2) FOR/NEXT: the CAP's that contains FOR/NEXT loop could be eliminated if and only if internal CAP's of this loop are eliminated.
- (3) CREATE/DESTROY: the CAPS containing command CREATE/DESTROY can only be eliminated, if the entity that is created or destroyed by these commands is also eliminated. A priori, these CAP's can be manually eliminated by the analyst, from the observation that all attributes referred to this entity were no longer part of the Transition Specification of the reduced model. For an automatic elimination of these commands, the attributes should be written according object orientation paradigm (OOP). For instance in the example of M/M/1 queue in the [appendix](#), the name of attribute for server status was written according OOP ("server.status").
- (4) INPUT/OUTPUT: these commands utilize attributes as arguments. If one attribute is eliminated by the algorithm, it also should be eliminated from the arguments lists.

It is also important to make some remarks. First is that this technique proposes how it is possible to obtain a simpler model from a more complex model, for selected simulation objectives or performance measures (which may be a subset of the all objectives to be considered all along a simulation study). So it is possible to achieve reduction with more than one simulation objective at the same time. However the number of objectives is inversely correlated to the reduction potential (more objectives – less reduction potential). Secondly, the technique proposed here reduces the structure of the conceptual model (by lowering the total number

of CAP's within the Condition Specification). Therefore, computationally speaking, the execution time should be also reduced, but this is not the main aim of this technique, as it is in [29].

In the next section, we show a simple application example of the described algorithm.

### 6. Simple application example

We will apply the reduction algorithm explained in the previous section to a simple example to demonstrate the feasibility of this approach. In this simulation model there are three machines in series (M1, M2 and M3). Pieces arrive in the system with an interval exponentially distributed with mean of "lambda" minutes. First it is processed on machine M1, then on machine M2 and finally on machine M3 (always in this sequence). After being processed on Machine 3, pieces leave the system. Each machine could process one piece at a time and there is an infinite buffer of pieces between them. The objective of the simulation model is to determine the utilization of machine 2. The processing times of machine M1, M2 and M3 follows a normal distribution with mean m1, m2 and m3, respectively, and 15% standard deviation related to the mean. Fig. 1 shows this described model in ACD format.

Utilizing the ACD/CS conversion rules [41], the *Transition Specification* for this model is depicted in Table 3 (already transformed into tabular format).

The reduction procedure takes place by starting by the performance measure that represents the objective at hand, that is the utilization of Machine 2 (Table 4). Therefore we should initialise the backtracking table by identifying the CAP's that contains "Utilization2" as *Output Attribute*. CAP numbered "33" obeys this condition, and thus should be considered on the backtracking table. As we can observe, for this CAP, "Utilization2" is influenced by attributes "Total\_time2" and "Sys\_Time" and consequently should be placed on column 3 of the backtracking table. "Utilization2" also appear as *Output Attribute* in CAP number 8, which should also be considered in the table. In this case "Utilization2" is only initialised, and is not influenced by any other attribute. The described procedure should be repeated for the next found attribute, which is "Total\_time2" and then successively until the stopping criteria is met (no more attributes left to backtrack). The complete result of the algorithm is shown in Table 5.

Hence only the CAP's 1, 4, 5, 7, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 25, 26 (determined by backtracking algorithm) and CAP 34 (STOP command) affects the utilization of machine 2. Those generate the *Transition Specification* shown in Table 5. Fig. 2 shows the equivalent ACD format for this situation.

If we adopt as a complexity measure simply the numbers of CAP's on the Transition Specification, we found 23/34 that is equivalent to a reduction of 33%.

Note that for this problem the results obtained for the backtracking algorithms is intuitively natural: since there are infinite buffers, the behavior of Machine 3 would not affect the behavior of Machine 2, and since we are only interested in measuring utilization of machine 2, machine 3 could be eliminated. However, it is not guaranteed if a reduction always could be achievable.

Suppose that now we would like to evaluate the throughput of the system (produced\_rate). If we apply the backtracking procedure, it is not possible to obtain a simpler model, because all three machines have influence of the throughput of the system. Another case that the reduction is not possible is in the hypothesis of finite

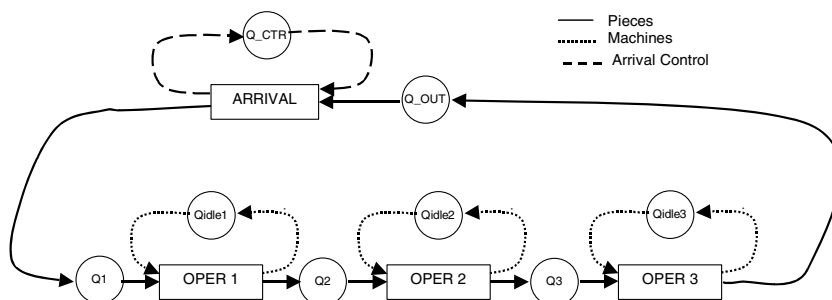


Fig. 1. Complete ACD representation for the 3 Machine Problem.

Table 3  
Transition specification for the 3 machine problem

Cap	Condition	Action
1	INITIALIZATION:	Q1.size := 0
2	INITIALIZATION:	Q2.size := 0
3	INITIALIZATION:	Q3.size := 0
4	INITIALIZATION:	QIdle1.size := 1
5	INITIALIZATION:	QIdle2.size := 1
6	INITIALIZATION:	QIdle3.size := 1
7	INITIALIZATION:	SETALARM(End_arrive, negexp(lambda))
8	INITIALIZATION:	utiliz2 := 0
9	INITIALIZATION:	produced_rate := 0
10	INITIALIZATION:	num_parts := 0
11	INITIALIZATION:	start_time2 := 0
12	INITIALIZATION:	total_time2 := 0
13	WHEN ALARM(End_arrive)	Q1.size := Q1.size + 1
14	WHEN ALARM(End_arrive)	SETALARM(End_arrive, negexp(lambda))
15	(QIdle1.size > 0) and (Q1.size > 0)	QIdle1.size := QIdle1.size - 1
16	(QIdle1.size > 0) and (Q1.size > 0)	Q1.size := Q1.size - 1
17	(QIdle1.size > 0) and (Q1.size > 0)	SETALARM(End_mach1, normal(m1,0.15*m1))
18	WHEN ALARM(End_mach1)	QIdle1.size := Q1.size + 1
19	WHEN ALARM(End_mach1)	Q2.size := Q2.size + 1
20	(QIdle2.size > 0) and (Q2.size > 0)	QIdle2.size := QIdle2.size - 1
21	(QIdle2.size > 0) and (Q2.size > 0)	Q2.size := Q2.size - 1
22	(QIdle2.size > 0) and (Q2.size > 0)	start_time := sys_time
23	(QIdle2.size > 0) and (Q2.size > 0)	SETALARM(End_mach2, normal(m2,0.15*m2))
24	WHEN ALARM(End_mach2)	Q3.size := Q3.size + 1
25	WHEN ALARM(End_mach2)	QIdle2.size := QIdle2.size + 1
26	WHEN ALARM(End_mach2)	total_time2 := total_time2 - (sys_time - start_time)
27	(QIdle3.size > 0) and (Q3.size > 0)	QIdle3.size := QIdle3.size - 1
28	(QIdle3.size > 0) and (Q3.size > 0)	Q3.size := Q3.size - 1
29	(QIdle3.size > 0) and (Q3.size > 0)	SETALARM(End_mach3, normal(m3,0.15*m3))
30	WHEN(End_mach3)	num_parts := num_parts + 1
31	WHEN(End_mach3)	QIdle3.size := QIdle3.size + 1
32	sys_time > 1000	produced_rate := num_parts/sys_time
33	sys_time > 1000	utilization2 := total_time2/sys_time
34	sys_time > 1000	STOP

buffers. Suppose now that the buffer limit is 20 pieces. In this case we have to add at CAP's 13, 14, the condition "Q1.size ≤ 20", to CAP's 18, 19, the condition "Q2.size ≤ 20" and to CAP's 24, 25, 26, "Q3.size ≤ 20". In this situation, the reduction is not possible.

## 7. A more complex example

The later section illustrates a simple example of how the proposed technique is applied. That example is simple and straightforward, which is useful for understanding purposes. In this section we will discuss a more complex example in order to show the feasibility of the proposed technique when applied to more realistic problems. The "Steelworks" problem is based on a real problem and was discussed in [43]. The complete reduction process appears in [44].

In a the Steelworks plant, there are two blast furnaces, which melt iron at certain daily volumes, which blows and fills as many torpedoes as available and are used to transport molten iron. If no torpedo is available, the molten iron is dropped on the floor and waste is produced. Each torpedo can hold a fixed quantity of molten iron. All torpedoes with molten iron go to a pit, where crane(s)-carrying ladles are filled from torpedoes, one at a time. The ladle holds 100 tons of molten iron, which is exactly the volume of a steel furnace that is fed from the crane. There are certain numbers of steel furnaces, which work together and produce the final product of the steelworks. Fig. 3 shows the schematic layout (not to scale) for the steelworks plant.

Table 4  
Final backtracking table for the 3 machine problem

CAP#	Backtracking of:	Influenced by:
33	Utilization2	Total_time2, sys_time
8	Utilization2	–
26	Total_time2	Sys_time, start.time2, End.mach2
12	Total_time2	–
22	Start_time2	Sys_time, Qidle2.size, Q2.size
11	Start_time2	Sys_time, Qidle2.size, Q2.size
23	End_mach2	Qidle2.size, Q2_size, m2*
5	Qidle2_size	Q2.size
25	Qidle2_size	End_mach2
20	Qidle2_size	Qidle2.size, Q2.size
19	Q2.size	End_mach1
2	Q2.size	–
21	Q2.size	Qidle2.size, Q2.size
17	End_mach	Qidle1.size, Q1.size, m1*
4	Qidle_size	–
15	Qidle1.size	Qidle1_size, Q1_size
18	Qidle1.size	End_mach1
1	Q1.size	–
13	Q1.size	End_arrive
16	Q1.size	Qidle1.size, Q1.size
7	End_arrive	Lambda*
14	End_arrive	Qidle1.size, Q1.size, Lambda*

Table 5  
Transition specification of the reduced 3 machine model

Cap	Condition	Action
1	INITIALIZATION:	Q1.size := 0
2	INITIALIZATION:	Q2.size := 0
4	INITIALIZATION:	QIdle1.size := 1
5	INITIALIZATION:	QIdle2.size := 1
7	INITIALIZATION:	SETALARM(End_arrive, negexp(lambda))
8	INITIALIZATION:	utiliz2 := 0
11	INITIALIZATION:	start_time2 := 0
12	INITIALIZATION:	total_time2 := 0
13	WHEN ALARM(End_arrive)	Q1.size := Q1.size + 1
14	WHEN ALARM(End_arrive)	SETALARM(End_arrive, negexp(lambda))
15	(QIdle1.size > 0) and (Q1.size > 0)	QIdle1.size := Qidle1.size – 1
16	(QIdle1.size > 0) and (Q1.size > 0)	Q1.size := Q1.size – 1
17	(QIdle1.size > 0) and (Q1.size > 0)	SETALARM(End_mach1, normal(m1, 0.15*m1))
18	WHEN ALARM(End_mach1)	QIdle1.size := Q1.size + 1
19	WHEN ALARM(End_mach1)	Q2.size := Q2.size + 1
20	(Qidle2.size > 0) and (Q2.size > 0)	QIdle2.size := Qidle2.size – 1
21	(Qidle2.size > 0) and (Q2.size > 0)	Q2.size := Q2.size – 1
22	(Qidle2.size > 0) and (Q2.size > 0)	start_time := sys_time
23	(Qidle2.size > 0) and (Q2.size > 0)	SETALARM(End_mach2, normal(m2, 0.15*m2))
25	WHEN ALARM(End_mach2)	QIdle2.size := Qidle2.size + 1
26	WHEN ALARM(End_mach2)	total_time2 := total_time2 – (sys_time – start_time)
33	sys_time > 1000	utilization2 := total_time2 / sys_time
34	sys_time > 1000	STOP

The main objective of this model is to determine the total quantity of waste during the process (expressed by the variable “totwaste”). The waste is generated if there is no torpedo available to carry molten iron from the Blast Furnaces just after the blasting. The ACD Description of this model is depicted in Fig. 4.

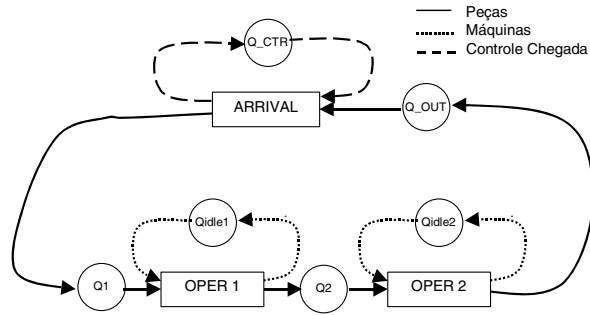


Fig. 2. Reduced ACD representation for the 3 Machine Problem.

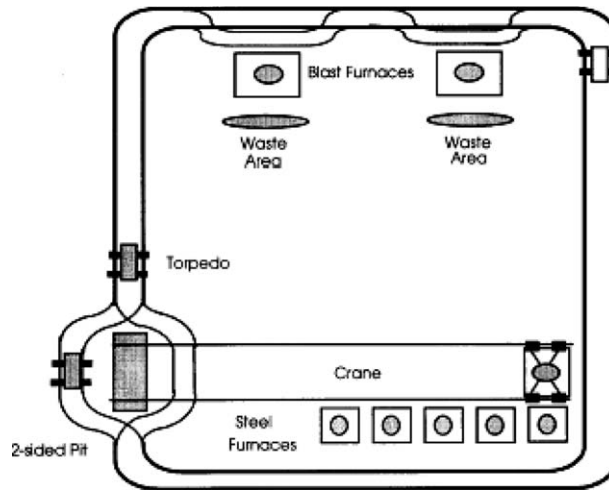


Fig. 3. Layout for the Steelworks Plant.

Due to space constraints, the CS representation (with 221 Caps) will be omitted but is shown in [44]. Backtracking algorithm was applied considering the objective “totwaste” and one additional hypothesis: that the crane is always available. In CS, this could be easily translated as:  $Q_{crane.size} > 0$ . The reduced ACD is shown in Fig. 5 and it has 123 CAP’s, providing a 44% reduction factor.

**8. Conclusions and future work**

This work is an attempt to search for a formal methodology for the Model Reduction Problem at the conceptualization phase of simulation modeling. Besides the simple example shown here, we have made other tests with the algorithm which led to the following conclusions:

- (1) The potential to reduce an existing simulation model does not depend on its complexity, but on simulation objectives and additional hypotheses made on the model, refuting the common sense idea that a more complex model has more reduction potential.
- (2) As shown in the example, there are some cases that no reduction is achieved, because all attributes considered in the model can affect the desired performance measures.

The reduction algorithm proposed by this work follows the evolutive/reductionist approach, i.e. starts with a more complex model and attempts to simplify it. We found this approach feasible most to non-expert modelers which tend to build complex models first [1]. Still the reduction algorithm obeyed the four principles

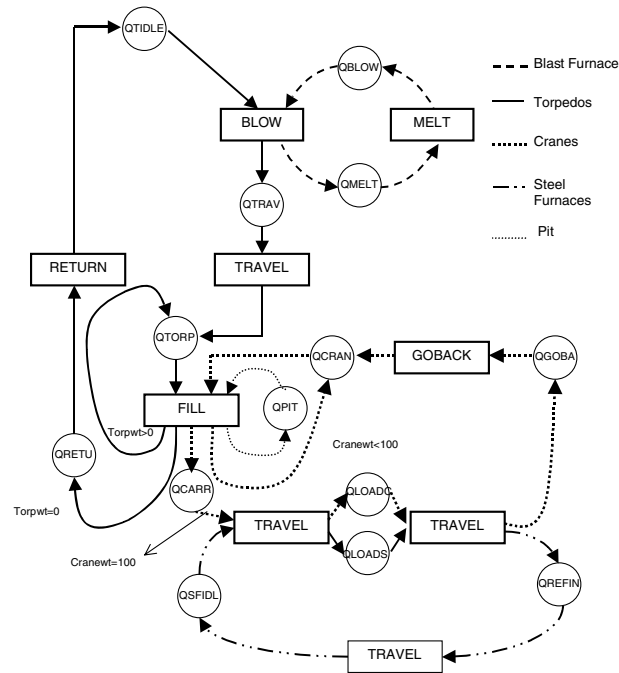


Fig. 4. ACD representation for the Steelworks Plant.

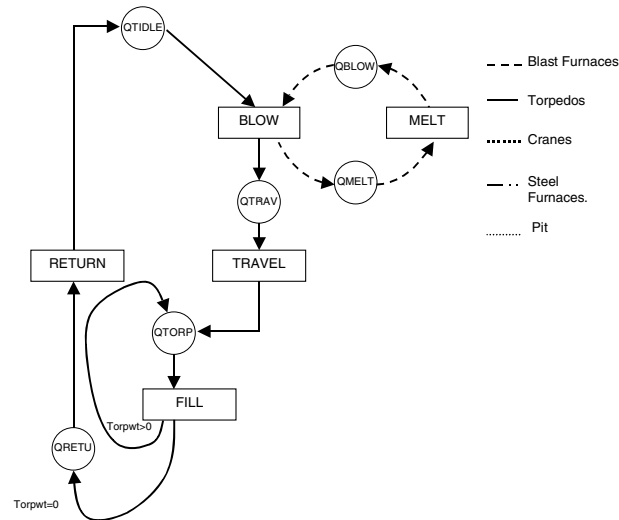


Fig. 5. ACD representation of the reduced model for the Steelworks Plant.

listed in Section 3: (i) it depends upon the simulation objectives, (ii) it is performed at conceptual phase of simulation modeling, (iii) it guarantees the validity of the simpler model since it eliminates only the attributes that do not interfere with the desired measures, being classified into type II reduction process. Finally, it can be performed on a computer (although here it was manually made for didactic purposes, the algorithm could be written in any programming language, since it consists of well-defined sequential steps).

By this work, we demonstrate the feasibility of the reduction algorithm to a discrete event simulation model described in Condition Specification Technique. In this case we reduced the complexity of the model by omission i.e. skipping some parts of the model that do not interfere with simulation objectives. This is however one

possibility of reduction. There are, according to Pedgren et al. [9], other possibilities like substitution and aggregation. Future work will search for techniques that could also deal with these possibilities.

The reduction technique works with a specific model representation technique that is Condition Specification. Since it is very difficult to represent a simulation model in this formalism directly, we created some general rules to aid the process of transformation from the model represented in Activity Cycle Diagrams format into Condition Specification. Hence, we could have two possibilities: to find a reduction technique that works with other representation or to find other conversion rules from another representation that could allow also the application of reduction algorithm into commercial simulation tools.

As mentioned before, despite the importance of this subject, the lack of research in this area is surprisingly high. We hope that this work can serve as an incentive for future researchers that would like to endeavor in this area.

## Acknowledgements

The authors would like to thank the Department of Information Systems and Computing, Brunel University, who provided the resources necessary for part of this research, whilst Leonardo Chwif was a Research Visitor. Thanks also to the Brazilian's Research Funding Agencies CAPES who supported partially this research (under reference no. 0439/98-8).

## Appendix A. Basic review of Activity Cycle Diagrams and condition specification

### A.1. Activity Cycle Diagrams

The Activity Cycle Diagram (ACD) is a model representation technique, disseminated since the late 1960s. Originally, it was based on Tocher's idea of stochastic gearwheels. The ACD depicts the interaction of entities (any component of a system which retains its identity through time), by composing their life cycles. An entity could be either in a Passive State (Queue) or in an Active State (Activity), and the Queue and Activity are the only two symbols that appear in an ACD graph (see Fig. A1). It is important to note: (1) The duration of an activity is always known in advance (given by some stochastic distribution), and an activity state usually involves the co-operation of different types of entities. (2) The duration of an entity in a queue cannot be known in advance.

There are some basic rules or conventions for constructing an ACD: (1) A Queue must contain only one type of entity. (2) An activity always follows a queue and vice-versa (when there are no reasons for queuing before an activity, dummy queues may be incorporated into the representation). (3) All life cycles of each entity type should be closed.

The most common and classical example of an ACD representation is the "Pub Example". Here, for brevity, we will show a simplified version. In this case, the model contains three entities: "man", "barmaid" and the "glass". The man drinks or waits for drinks. The barmaid either pours a drink or is idle. The glass could be used by the barmaid to pour the drink or used by the customer while drinking, or is either waiting to be drunk by the customer or waiting to be poured by the barmaid. The ACD for this simplified version of the PUB is showed in Fig. A2.

If we add entity tokens to the ACD graph (each token representing a single entity), it is possible to do a manual simulation, and the position of each token in an activity or queue gives the state of the system. It has been shown in the literature that some methods of simulating systems is more suitable for the ACD representation, like the Three Phase Method [29].

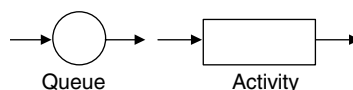


Fig. A1. Basic elements of the ACD representation.

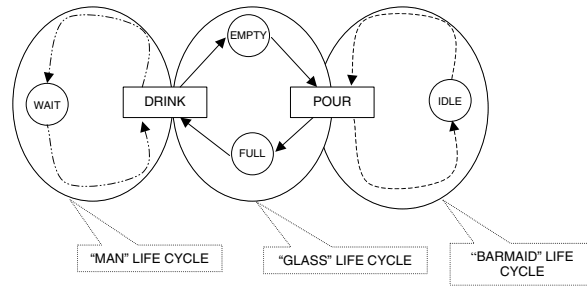


Fig. A2. Simplified ACD of the Pub Model.

A.2. Condition specification

Overstreet and Nance [37], introduced a formal representation technique (formalism) or a formal language called Condition Specification (CS) with the intended aim of providing assistance in the analysis of discrete event simulation models. CS is based on the Canonical Methodology also introduced by Nance (an approach that mingles top-down definitions of a simulation model coupled with bottom-up specifications).

Basically, a CS of a model consists of two basic elements: description of the communication interface for the model – *Interface Specification* (1) and a specification of the model dynamics that includes *Object Specification* (2) and *Transition Specification* (3). CS also defines a *Report Specification* (4), but for the effect of this work we will concentrate on the specifications 1–3.

The Interface Specification identifies the input and output attributes by name, data type and communication type (input or output). The Object specification is a list of all model objects and their attributes. It must have at least one Output Attribute (Simulation Model Requirement or Performance Measure). The Transition Specification consists of a set of ordered pairs called CAP (condition-action pairs). A condition simply is a Boolean expression that would be evaluated during the model instantiation. The CAP structure is similar to “Production Rules” in Knowledge Based Systems, because they always have an “if-then” structure. If two CAP’s have exactly the same condition, they are grouped into Action Clusters (AC). In addition, two conditions always appear in every Transition Specification: Initialization and Termination. The first one is only true at the start of the model instantiation, and the second only at the end of the model instantiation.

All the specifications are written with a kind of “Pascal-Like” language. For the *Transition Specification*, CS syntax provides several primitives, which can be seen in Table A1.

Figs. A3–A5 provide respectively the *Interface Specification*, the *Object Specification* and the *Transition Specification* for a simple M/M/1 Queue Model. This example is taken from [40]. Note that the Transition Specification has five Action Clusters (Initialization, Arrival, Begin Service, End Service and Termination).

For each Action Cluster (AC), attributes are classified into the following three categories: *Control Attribute*, *Output Attribute* and *Input Attribute*. An attribute *x* is a *Control Attribute* if *x* appears in a condition expres-

Table A1  
Condition specification basic syntax [37]

Name	Syntax
Value change description	<objectName>.attribute := newValueExpression
Set alarm	SET ALARM(alarmName, alarmTime (, argList))
When alarm	WHEN ALARM(alarmName)
Cancel	CANCEL(alarmName (, alarmId))
Create	CREATE(objectName{instanceId})
Destroy	DESTROY(objectName{instanceId})
Input	INPUT(attriBList)
Output	OUTPUT(attriBList)
Stop	STOP

**Input Attributes:**

*Arrival\_mean*: positive real;  
*Service\_mean*: positive real;  
*Max\_served*: positive integer

**Output Attributes:**

*Server\_utilization*: positive real;

Fig. A3. Interface specification for a M/M/1 Queue Model.

Object:	Attribute:	Type:
Environment	<i>arrival_mean</i>	positive real;
	<i>service_mean</i>	positive real;
	<i>system_time</i>	positive real;
	<i>arrival</i>	time-based signal;
Server	<i>server_status</i>	(idle,busy);
	<i>queue_size</i>	nonnegative integer;
	<i>num_served</i>	nonnegative integer;
	<i>max_served</i>	nonnegative integer;
	<i>end_of_service</i>	timed-based signal;

Fig. A4. Object specification for a M/M/1 Queue Model.

**{Initialization}****INITIALIZATION:**

```
INPUT(arrival_mean,service_mean, max_served);
CREATE(server);
Serverstatus:=idle;
Num_served:=0;
System_time:=0.0;
SET ALARM(arrival,0);
```

**{Arrival}****WHEN ALARM(arrival):**

```
queue_size:=queue_size+1;
SET ALARM(arrival, negexp(arrival_mean));
```

**{Begin Service}**

*queue\_size*>0 and *server\_status*=idle:

```
queue_size:=queue_size-1;
server_status:=busy;
SET ALARM(end_of_service, negexp(service_mean));
```

**{End Service}****WHEN ALARM(end\_of\_service):**

```
server_status:=idle;
num_served:=num_served+1;
```

**{Termination}**

*num\_served* >= *max\_served*:

```
STOP
PRINT REPORT
```

Fig. A5. Transition specification for a M/M/1 Queue Model.

sion of the Action Cluster; Attribute  $x$  is an *Output Attribute* if the actions of the Action Cluster can change the value of attribute  $x$  and,  $x$  is an *Input Attribute* if the value of  $x$  affects the value of the *Output Attributes*. For the M/M/1 Model, with the Action Cluster “Arrival”, for instance, we can identify that *arrival* is a *Control Attribute*, *queue\_size* and *arrival\_mean* are *Inputs Attributes* and also *arrival* and *queue\_size* are *Output Attributes*.

## References

- [1] L. Chwif, M.R.P. Barretto, R.J. Paul, On simulation model complexity, in: *Proceedings of the Winter Simulation Conference*, 2000, pp. 449–455.
- [2] E.H. Page et al., Panel: strategic directions in simulation research, in: *Proceedings of Winter Simulation Conference*, 1999, pp. 1509–1520.
- [3] S.C. Ward, Arguments for constructively simple models, *Journal of the Operational Research Society* 40 (2) (1989) 141–153.
- [4] H.Y. Yin, Z.N. Zhou, Simplification techniques of simulation models, in: *Proceedings of Beijing International Conference on System Simulation and Scientific Computing*, 1989, pp. 23–26.
- [5] G. Innis, E. Rexstad, Simulation model simplification, *Simulation* 3 (July) (1983) 7–15.
- [6] A.M. Law, D.W. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill Inc., 1991.
- [7] K.J. Musselman, Guidelines for simulation project success, in: *Proceedings of Winter Simulation Conference*, 1993, pp. 58–64.
- [8] S. Robinson, *Successful Simulation – A Practical Approach to Simulation Projects*, McGraw-Hill, London, 1994.
- [9] D.C. Pedgren, R.E. Shannon, R.P. Sadowski, *Introduction to Simulation using Siman*, McGraw-Hill, 1995.
- [10] J.D. Salt, Keynote address: simulation should be easy and fun! in: *Proceedings of the Winter Simulation Conference*, 1993, pp. 1–5.
- [11] M. Pidd, Five simple principles of modelling, in: *Proceedings of the Winter Simulation Conference*, 1996, pp. 721–728.
- [12] R.J. Brooks, A.M. Tobias, Methods and benefits of simplification in simulation, in: *Proceedings of UKSIM*, United Kingdom Simulation Society, 1999, pp. 88–92.
- [13] R.J. Brooks, A.M. Tobias, Simplification in the simulation of manufacturing systems, *International Journal of Production Research* 38 (5) (2000) 1009–1027.
- [14] M. Bunge, *The Myth of Simplicity: Problems of Scientific Philosophy*, Prentice-Hall, Englewood Cliffs, NJ, 1963.
- [15] S.L. Chen, M. Geradin, An exact model reduction procedure for mechanical systems, *Computational Methods Applied to Mechanical Engineering* 143 (1997) 69–78.
- [16] B.C. Moore, Principal component analysis in linear systems: controllability, observability, and model reduction, *IEEE Transactions on Automatic Control* 26 (1) (1981) 17–31.
- [17] S. Sevinc, Theories of discrete event model abstraction, in: *Proceedings of the Winter Simulation Conference*, 1991, pp. 1115–1119.
- [18] F.K. Frantz, A taxonomy of model abstraction techniques, in: *Proceedings of the Winter Simulation Conference*, 1995, pp. 1413–1420.
- [19] R.M. McGraw, R.A. MacDonald, Abstract modeling for engineering and engagement level simulations, in: *Proceedings of the Winter Simulation Conference*, 2000, pp. 326–334.
- [20] B.P. Zeigler, *Theory of Modelling and Simulation*, John Wiley, 1976.
- [21] S. Robinson, *Simulation: The Practice of Model Development and Use*, John Wiley & Sons, 2004.
- [22] P.J. Courtois, On time and space decomposition of complex structures, *Communications of ACM* 28 (6) (1985) 591–603.
- [23] S. Sevinc, N.Y. Foo, Discrete event model simplification via state classification, in: W. Webster, R. Uttamsingh (Eds.), *AI and Simulation: Theory and Applications*, Simulation Series, 22 (3), April 1990.
- [24] B.P. Zeigler, *Theory of Modelling and Simulation*, John Wiley & Sons, 1976.
- [25] R.R. Barton, Metamodeling: a state of the art review, in: *Proceedings of the Winter Simulation Conference*, 1994, pp. 37–244.
- [26] D. Caughlin, Parameter identification methods for metamodeling simulations, in: *Proceedings of the Winter Simulation Conference*, 1996, pp. 756–763.
- [27] C.G. Panayiotou, C.G. Cassandras, W. Gong, Model abstraction for discrete event systems using neural networks and sensitivity information, in: *Proceedings of the Winter Simulation Conference*, 2000, pp. 335–341.
- [28] A. Yan, W. Gong, Time-driven fluid simulation for high-speed networks, *IEEE Transactions on Information Theory* 45 (5) (1999) 1588–1599.
- [29] R.E. Nance, M.C. Overstreet, E.H. Page, Redundancy in model specifications for discrete event simulation, *ACM Transactions on Modeling and Computer Simulation* 9 (3) (1999) 254–281.
- [30] K.D. Tochter, *The Art of Simulation*, English Universities Press, 1963.
- [31] R.J. Paul, D.W. Balmer, *Simulation Modelling*, Chartwell-Bratt, London, 1993.
- [32] M. Pidd, *Computer Simulation in Management Sciences*, fourth ed., John Wiley & Sons, Chichester, 1998.
- [33] B.A. Cota, R.G. Sargent, Control flow graphs: a method of model representation for parallel discrete event simulation, CASE Center Technical Report 9026, Syracuse University, Syracuse, NY, 1990.
- [34] L. Schruben, Simulation modeling with event graphs, *Communications of the ACM* 26 (11) (1983).
- [35] C.A. Petri, *Communication with Automata*, New York: Griffiss Air Force Base, Tech. Report. RADC-TR-65-377 1 (Suppl.1), 1966.
- [36] A.A. Torn, Simulation graphs: a general tool for modeling simulation designs, *Simulation* 37 (6) (1981) 187–194.
- [37] D. Harrel, Statecharts: a visual formalism for complex systems, *Science of Computer Programming* 8 (3) (1987) 231–274.
- [38] L.W. Schruben, Graphical model structures for discrete event simulation, in: *Proceedings of the Winter Simulation Conference*, 1992, pp. 241–245.
- [39] M.C. Overstreet, R.E. Nance, A specification language to assist in analysis of discrete event simulation models, *Communications of the ACM* (1985) 191–201.
- [40] H.E. Page, *Simulation modeling methodology: principles and etiology of decision support*, Unpublished Ph.D. Thesis, Virginia Polytechnic Institute and State University, Virginia, 1994.
- [41] L. Chwif, R.J. Paul, M.R.P. Barretto, Combining the Best of the two: An Activity Cycle Diagram/Condition Specification Approach, in: *Proceedings of the UKSIM*, United Kingdom Simulation Society, 1999, pp. 93–98.

- [42] M.C. Overstreet et al., Model diagnosis using the condition specification: from conceptualization to implementation, in: Proceedings of the Winter Simulation Conference, 1994, pp. 566–573.
- [43] M.R.P. Barretto, L. Chwif, T. Eldabi, R.J. Paul, Simulation optimization with the linear move and exchange move optimization algorithm, in: Proceedings of the Winter Simulation Conference, 1999, pp. 806–811.
- [44] L. Chwif, M.R.P. Barretto, Discrete event simulation model reduction: some case studies, Technical Report 1826, Universidade de São Paulo, São Paulo, Brazil, 2003 (in Portuguese).