Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

COMPUTER ASSISTANCE FOR MODEL DEFINITION

Henk de Swaan Arons
Eelco van Asperen

Faculty of Economics
Department of Computer Science
Erasmus University Rotterdam
P.O. Box 1738, H9-28
3000 DR Rotterdam, The Netherlands

ABSTRACT

Modeling requires considerable knowledge of the various stages of the simulation process. The modeler needs to know a great deal of the system to be modeled (domain specific knowledge), the ins and outs of the modeling process itself (the degree of detail of the model) and how to implement the model in a simulation language. Each of these stages would benefit from some kind of knowledge-able support. In this article a decision-making process is described that supports the modeler to build a model step by step. As a vehicle the Arena simulation environment has been used. The support is based on information provided by the modeler and is essentially data-driven. It suggests which modules could be used best, which parameters need to be determined and helps to formulate route information. This research aims for an implementation of this support using a knowledge-based system.

1 INTRODUCTION

Modeling is one of the most important parts of the simulation process. Not only in discrete-event systems but also in continuous-time systems (System Dynamics). This problem was one of the main research aims of the research project (Toolkit 1997) that investigated how modeling could be improved. It is obvious that in some respect this support is preferably knowledge-based.

One approach to knowledge-based support has been discussed in (De Swaan Arons 1983) and (De Swaan Arons 1999). In the first paper a mathematical model is selected based on user input. The approach in the second paper essentially attempted to reuse existing simulation implementation models. Over time many implementation models have been developed using some simulation language. These models can be stored in a database with the intention to reuse them in a later stage. When a new model has to be designed a modeler could try to retrieve an existing model from the database similar to the one to be developed. To facilitate this selection process, the models stored in the database need to be parameterized in such way that a query on this database can be formulated and will hopefully result in a non-empty dynaset. Since the modeler in principle has no knowledge or does not want to have knowledge of the parameters used to describe the models in the database an expert system could be used to transform the design specifications into the right query, taking all kinds of design considerations into account. This approach is essentially goal-driven: given a number of simulation models the knowledge-based system attempts to gather the information from the user (or from other sources) on the basis of which one or more models could be selected. Currently, this approach is further investigated and experiments are being carried out.

Unfortunately, this approach has a complication. The chance that a database would contain exactly the same model the user is after is negligible. One minor difference makes two ‘equal’ models different from each other. For example, if two models are equal in all respects apart from the distribution function in one single Arrive module, humans would probably see these models as very similar although they are – strictly speaking - different. Therefore, the modeler would be very pleased when a support system would recognize the similarity between the wanted model on the one hand and some models (if there are any) in the database on the other hand.

In this paper, a different approach is discussed that may be more feasible in certain cases. In this approach the knowledge-based system essentially supports the modeler in the decision-making process. The modeler may have extensive domain knowledge and a clear view of the system to be modeled. He may have little experience with simulation tools and would certainly appreciate expert advice. The approach discussed in this paper could provide such help and eventually lead to automatic model generation. To test this approach we have selected the Arena simulation tool.

Thus far only, some minor efforts have been made to automate this support. By its Model Jump-Start Wizard the simulation package Arena, version 3.01 (see also Kelton, W.D., R.P. Sadowski, and D.A. Sadowski 1998) provides a toy system in building small simulation models. Based on the data provided by the modeler through a questionnaire Arena builds up a simple simulation model that is ready to run. In many respects this approach lacks essential aspects. For example, the Wizard supports only models containing a number of identical servers that are placed in line.

In the following section, we analyze how an Arena model can be designed and what factors may influence the various decisions that have to be made. These decisions concern, among others, the number of Arrive, Server, Inspect and Depart modules that have to be used, and the possible use of other modules such as the Sequences module. Furthermore, in each of these modules the values of many parameters have to be determined. This analysis could be formalized in that it could help the modeler to design and build the simulation model he is after. A schematic outline of this approach is discussed in section 3. A few examples illustrating how this approach works are given in section 4. There are various ways to formalize such a process. Because of the reasoning aspects of the suggested approach a knowledge-based approach seems to be most appropriate and therefore some relevant techniques are discussed in section 5. The approach discussed in this paper is only a first step on a long road toward real computer-aided modeling of simulation systems. In section 6 some remarks are made about how to proceed on this road.

2 CONCEPTS

In this section, we discuss the various concepts on which the approach is based. To fully understand the implications it is necessary to introduce the concept entity and to give a brief introduction of some basic terminology used in Arena.

2.1 Entities

An entity in a discrete-event system could be a product going from one machine to another, a person entering a bank and going from one desk to another, a document in a administration going from one desk to another or a telephone call forwarded from one person to another. Such an entity could be seen as an instance of an entity type. So, an entity type represents a separate class of entity instances. Person and Product are two such entity types. An entity type can have two or more entity subtypes. For example, the entity type Person may have the entity subtypes Male, Female and Child.

There needs to be a reason to distinguish between several subtypes. The plain fact that the entity subtype Person could have subtypes Male, Female and Child is not enough. If all persons follow the same route through a system, travel from one location to the other in the same time and have the same processing times at various locations, they behave as completely identical entities. In such a case there is no need to distinguish between them and define subtypes for them. However, if males, females and children follow different routes, have different processing times or have different route times then there is a need to define a subtype for each of them.

2.2 Arena in Brief

Many good simulation languages are available. However, since Arena (version 3.01) is available at our Faculty, for the sake of convenience this simulation language has been taken as the modeling vehicle in this article.

It would have been too ambitious to attempt to cover the modeling process of any simulation model no matter its size or complexity. We restrict ourselves to those simulation models that can be constructed using the most generic modules provided by Arena. Often used modules are: Arrive and Depart, Server, Inspect, Sequences, Variables, Statistics and Simulate. These are powerful modules providing a lot of built-in functionality.

A modeler could create impressive models with these modules. The Arrive module is used to model the location in the system where entities enter the system. All kind of information can be defined in this module such as the inter-arrival time and the batch size of the entities entering the system, etc. The actual processing occurs at Server or Inspect modules (an Inspect module behaves like a Server module but in addition to this it has a built-in inspect function that make the various entities proceed along different routes). Detailed information such as processing times can be specified in these modules. The Depart modules are those locations in the system where the entities leave the system. The Sequences module contains route information and information of the processing times at the Server or Inspect modules for the various entities. The Statistics and Variables modules contain additional information concerning some specific variables and statistics. Finally, the Simulate module controls the overall simulation process.

2.3 Modules

In the previous section, we have enumerated some important modules at the most generic level of Arena. It is clear that a model will contain several of these modules. It is not determined how many of each have to be used, nor is it known in advance which parameters need to be specified and along which routes entities proceed through the system.

When starting from scratch, the modeler has to make decisions based on previous investigations. For example, what entities (e.g., products, persons, documents) have to be distinguished in the simulation model, how many distinct Arrive and Depart modules have to be chosen, how many Server and Inspect modules have to be used, and so on. These questions primarily concern the static information of the model. However, equally important is the information of the routes the various entities have to follow when they move through the system. Furthermore, many parameters contained by the various modules have to be determined. Examples are the distribution functions of the inter-arrival times at the Arrive modules, the various processing times at the Servers, the route times and so on. In the following sections we discuss most of these aspects in more detail.

2.3.1 Arrive Modules

Based on the discussion in section 2.1, it can be stated that each entity type requires a separate Arrive module. Therefore, the number of Arrive modules is equal to the number of entity types. When subtypes are defined, in a sense we have distinct entity subtypes sharing the same Arrive module. We can distinguish several possible scenarios.

Scenario 1: The model has only one entity type having no entity subtypes. This scenario is the most simple one, see Figure 1. All entities of this single entity type follow the same route, starting from the same Arrive module along several server modules and leaving the system through a Depart module. This route can be specified either in the Leave Data sections of both the Arrive and Server modules or in the Sequences module. In the first case in the Leave Data section, the Route / StNm option has to be selected, the station name of the next-coming station (either a Server or Depart module) has to be specified and the Route Time. When the route is specified in the Sequences module, the Leave Data sections will have the options Route / Seq specified instead.

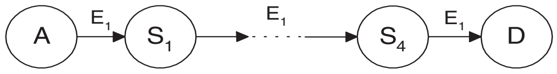

Figure 1: Only One Entity Type Having No Subtypes

Scenario 2: The model has only one entity type having two or more entity subtypes. The instances of the subtypes E_{11} and E_{12} follow different routes or follow the same route but have different processing times at shared servers, see Figure 2. Because of this there is a need to distinguish between the entity subtypes. For the server S_1 it is necessary to know which of the entity subtype is being served. First of all the entity subtypes E_{11} and E_{12} may have different processing times. Furthermore, both subtypes proceed along different routes. This information can be specified by using the Sequences module. In Arrive module A_1 the Assign / Assignment Type / Other = Sequence, Value = discr(p, Seq_{11}, 1 – p, Seq_{12}) has to be defined. Furthermore, in the Sequences module the sequences Seq_{11} and Seq_{12} have to be specified:

Seq_{11}: S_1, S_2, S_4, D
Seq_{12}: S_1, S_3, S_4, D,

together with the corresponding processing times. In the Arrive and the subsequent Server modules in the Leave Data section the corresponding route time have to be specified. For the Arrive and the Server modules the option Route / Seq has to be specified.

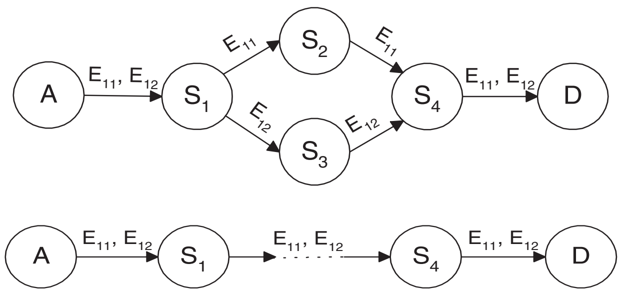

Figure 2: Since Some Instances of Entity Type E_1 Follow Different Routes (**a.**) or Have Different Processing Times at Shared Servers (**b.**) There is a Need to Specify to Subtypes E_{11} and E_{12}

Scenario 3: The model has two or more entity types, none of these having entity subtypes and all instances of all entity types follow the same route. Generally, instances of two different entity types E_1 and E_2 have different processing times at shared servers. This scenario is presented in Figure 3.

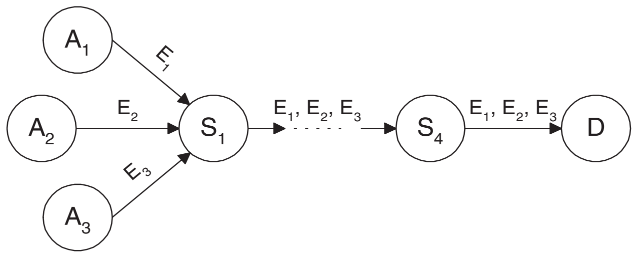

Figure 3: Instances of E_1, E_2 and E_3 Generally Have Different Processing Times at Servers They Share

Although the instances of all entity types will follow the same route, their processing times at the various servers will generally vary. So, the processing times of instances of entity type E_i at server S_j have to be defined at Arrive module A_i. This can be done by specifying a common attribute p at each Arrive module: at Arrive module A_i this attribute p will be given a distribution function d_{ij} for the processing time of entities E_i at Server S_j. The route information can be described using the same procedure as described in scenario 1. However, when the Sequences module is used the processing times can also be specified there instead of in the Server Data sections of the subsequent servers.

Scenario 4: The model has two or more entity types, none of these having entity subtypes, but instances of one entity type may follow a route that differs from instances of other entity types. This scenario is depicted in Figure 4. The instances of entity type E_1 emerging at Arrive module A_1 are routed to server S_1, while the instances of the entity type E_2 are routed directly to server S_2. Notice that instances of E_1 and E_2 share server S_2 and may have different processing times at this server. The figure clearly indicates that they proceed along different routes. Also later on in the model the instances of the entity types follow different routes: E_1 instances proceed to S_3 and finally to D_1, and E_2 instances proceed directly to D_1.

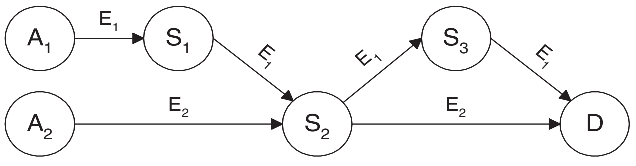

Figure 4: Instances of Entity Types E_1 and E_2 Follow Different Routes Sharing One or More Servers (in this case S_2)

Scenario 5: The model has two or more entity types, one or more of these having two or more entity subtypes. Finally, we discuss the scenario that one or more of the entity types have two or more subtypes which is depicted in Figure 5. Entity type E_1 has two subtypes E_{11} and E_{12}, E_2 has no subtypes. Instances of E_{11} and E_{12} follow different routes. Instances of subtype E_{11} occur with a probability of p and, consequently, instances of subtype E_{12} occur with a probability of 1 – p.

Again, for this model there is a need to specify the sequences Seq_{11} and Seq_{12}. The Sequences module has to define the sequences Seq_{11} and Seq_{12}:

Seq_{11}: S_1, S_2, D
Seq_{12}: S_1, S_3, S_4, D

Of course, also Seq_2, the sequence of E_2 instances has to be specified:

Seq_2: S_1, S_3, S_5, D

For all Arrive and Server modules the options Route / Seq have to be specified.

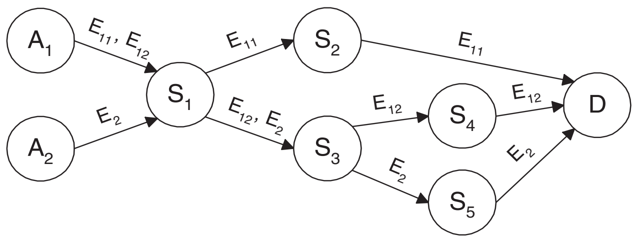

Figure 5: Instances of Entity Subtypes E_{11} and E_{12} and Instances of Entity Type E_2 Follow Different Routes Sharing One or More Servers

2.3.2 Server Modules

The number of servers and the parameters contained in them need to be determined. This is a less complicated procedure than determining the number of Arrive modules. In fact, all instances of either entity types or subtypes in a system have to be processed. When we restrict ourselves to the most generic level modules at the Common template panel only Server or Inspect modules can do the job. Inspect modules can be considered to be a special type of Server. For each of these servers the Station name, Resource, Capacity Type, Capacity, Process Time and Leave Data (route information) have to be provided. In most cases, however, the processing time and the route information is specified in the Sequences module.

2.3.3 Inspect Modules

Inspect modules are very similar to Server modules. Both perform an operation on instances of entity (sub)types. Generally an operation at a Server results in a modified entity. For example, in a manufacturing environment a product may undergo operations such as cleaning or milling, in an office a document could be inspected and signed, and so on. Inspect modules can carry out the same type of operation but can also inspect an entity resulting in pass or fail. For this reason it is very important to distinguish between Server and Inspect modules. This difference also has consequences for the definition of the sequences used. The pass or fail routes hold for all instances that have been processed at the Inspect module and can't be influenced by information in a Sequences module, see also section 3.2.

2.3.4 Depart Modules

The number of Depart modules can also be determined. Instances of either entity types or subtypes leave the system through one or more Depart modules. In most cases the modeler can specify how the instances leave the system. For each Depart module some parameters have to be specified such as the Station name, possibly some count information, tally variables and the type of statistics to be generated.

2.3.5 Sequences Module

The result of the decision process in connection with the number of Arrive modules (see also section 2.1) could make it necessary to use a Sequences module and thus to define the various sequences. The model depicted in Figure 5 gives rise to the definition of the three sequences Seq_{11}, Seq_{12} and Seq_2. So, in a previous stage the decision needs to be made with respect to whether or not a Sequences module has to be used and which sequences it has to specify. For each of the sequences the routes of the instances involved have to be specified as well as their processing times at the various servers.

2.3.6 Simulate Module

In order to run a simulation model in Arena one should use the Simulate module. In this module operational information has to be filled in, such as the simulation length, the warm up period, the number of replicates, whether or not the system or the statistics have to be initialized between the replications and so on. For the time being we restrict ourselves to some basic simulation runs.

3 THE DESIGN PROCESS

In this section we work out the decision process discussed in the previous section. As stated before we restrict ourselves to the modules of the Common template panel: Arrive, Server, Inspect, Depart, Simulate, Sequences.

First of all, in section 3.1, we provide a list of the parameters that essentially describe the models we consider given the restrictions mentioned earlier. Next, in section 3.2, we present a schematic overview that reflects large parts of the decision process discussed in section 2 and will eventually lead to the determination of the parameters of 3.1.

3.1 The Parameters Involved

In the modules discussed thus far many parameters can be specified. Some of these concern options that are not relevant to the research presented in this article, for example parameters concerning animation or transporters.

In the following an overview is given of the parameters that may have to be determined in the modules Arrive, Server, Inspect, Depart, Simulate and Sequences.

Arrive	
Enter data	Station name
Arrive data	Batch size
	Time between
	Mark time attribute
	Assign type (either Attribute or Other)
Leave data	Route / StNm Station name
	Route time
Server	
Enter data	Station name
Server data	Resource
	Capacity type (also includes schedule)
	Capacity
	Resource statistics (y/n)
	Process time
Leave data	Route / StNm
	Station name
	Route time
Inspect	
Enter data	Station name
Server data	Resource
	Capacity type (also includes schedule)
	Capacity
	Resource statistics (y/n)
	Process time
	Failure probability
Leave data	*Pass inspection leave data* Route / StNm
	Station name
	Route time
	Fail inspection leave data
	Route / StNm
	Station name
	Route time
Depart	
Enter data	Station name
Count	Type of counter
	Counter
	Increment
Tally	Tally
	Type of statistics
	Attribute
Simulate	
Project	Title
	Analyst
	Data
Replicate	Number of replications
	Beginning time
	Length of replications
	Terminating condition (leave empty)
	Initialize system
	Initialize statistics
	Warm up period
Sequences	Name
	Modules visited
	Processing time for each entity module combination

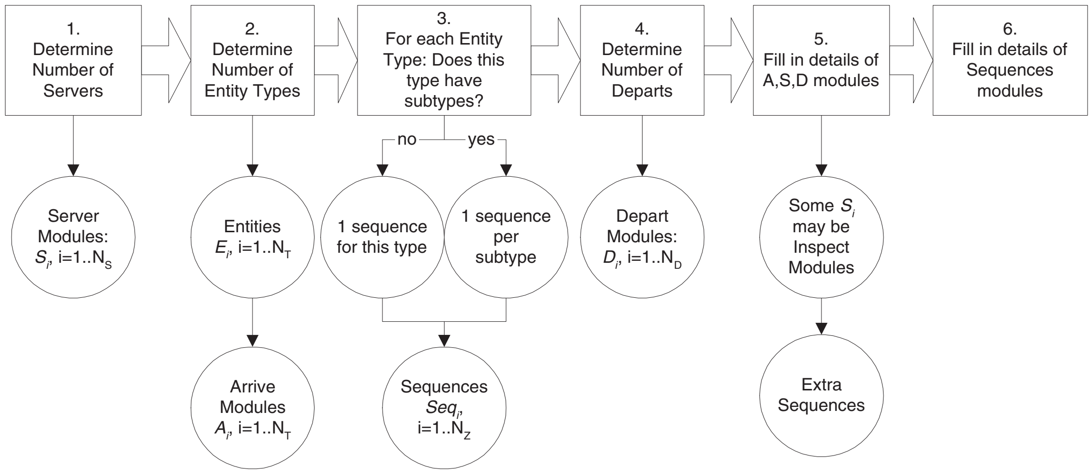
1. Determine Number of Servers
2. Determine Number of Entity Types
3. For each Entity Type: Does this type have subtypes?
4. Determine Number of Departs
5. Fill in details of A,S,D modules
6. Fill in details of Sequences modules
no
yes
Server Modules: S_i, i=1..N_S
Entities E_i, i=1..N_T
1 sequence for this type
1 sequence per subtype
Depart Modules: D_i, i=1..N_D
Some S_i may be Inspect Modules
Arrive Modules A_i, i=1..N_T
Sequences Seq_i, i=1..N_Z
Extra Sequences

Figure 6: Design Schema

3.2 Design Details

The design process consists of six steps, illustrated in Figure 6 (steps are represented by rectangles; results are represented by circles). The first step is to determine the number of activities to be modeled using the Arena Server and Inspect modules (noted as S_i). Having identified the number of locations, the user is then asked to name each one.

Step 2 aims to identify the number of entity types along the lines of section 2.3.1. Again, once the number of entity types has been established, the user is requested to name them. Each entity type E_i has a corresponding Arrival module A_i, which has to be named (a meaningful default such as "Arrival <name of entity type>" could be provided).

The third step is to determine whether an entity type has subtypes. If an entity type has subtypes, then these subtypes may follow different routes through the system or the subtypes may have different processing times at the servers visited. To model the routes of the entities through the system, we have decided to use the Arena Sequences module. The Sequences module specifies the modules visited by the entity associated with it and provides the option to set attribute values (such as the processing time) at each module.

The number of locations at which entities leave the model is the subject of step 4; the user is asked whether there is a need to distinguish between the entity (sub)types as they leave the system (for example, based on the physical location or whether specific statistical information is to be determined). Again, the user is asked to name the exit location, which are implemented using the Depart module.

Steps 1 through 4 have determined the initial number of modules in the model; step 5 then fills in the details of the modules, such as inter-arrival time for entities in the Arrival modules. For all Server modules, the user is queried whether the operation performed by the Server may fail; if so, then this Server module will be modeled using an Arena Inspect module. Using an Inspect module has consequences for the Sequences module. In Arena, if the route of an entity is described using a Sequences module, then the sequence will describe the route for the successful entities (i.e., the entities that pass the inspection). The routing information for the entities that fail the inspection must then be modeled using a direct connection from the Inspect module to the next module and a sequence that describes the route from that module onwards. This is illustrated in Figure 7: the first sequence $Seq_1 = S_1, I_1, S_2, D_1$ describes the route for the entities that passed the test in I_1; the second sequence $Seq_2 = S_3, D_2$ is used for the entities that failed the test in I_1. All entities that pass through an Inspect module have the same distribution function for the inspection process; the route out of the Inspect module is determined solely by the outcome of this distribution function and does not depend on the entity (sub)type.

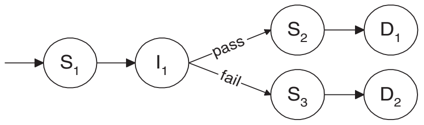

Figure 7: Inspect Module

The last step, step 6, is to determine the routes the entity types and subtypes follow on their way through the model. The starting point for each sequence is known to be the Arrival module for the (sub)type associated with that sequence (except for the extra sequences that may have been added in step 5); the end point is a Depart module. The user is now asked to provide the route for each entity (sub)type and to provide additional details such as the processing time when a Server or Inspect module is visited.

3.3 Corresponding Arena Actions

The results of the design process must now be translated into an Arena model. These are the parameters that have to be set for the Arena modules used. In the following the information that needs to be specified by the modeler is printed in italic.

```
Arrival:
    Enter Data:
        Station: a station name
    Arrival Data:
        Time Between: a distribution function
        Assign/Other/Sequence:
            No SubTypes: name of the sequence
            SubTypes: name of number of subtypes
    Leave Data:
        Route, Seq, Route Time: number of minutes

Server:
    Enter Data:
        Station: a station name
    Server Data:
        Process Time:
            same for all entities: a distribution function
            different: attribute name Process Time
    Leave Data:
        Route, Seq, Route Time: number of minutes

Inspect:
     Enter Data:
        Station: a station name
    Server Data:
        Process Time:
            same for all entities: a distribution function
            different: attribute name Process Time
        Failure Probability: a probability p
    Pass Inspection Leave Data:
        Route, Seq, Route Time: number in minutes
    Fail Inspection Leave Data:
        Route, StNm
        Station: name of next station
        Route Time: number of minutes

Depart:
    Enter Data:
        Station: a station name
    Count:
        Individual Counter, Counter: name of the counter
        Increment: 1

Sequences:
    For each sequence:
        Add: Sequence_Name
        Steps:
            the modules visited on the route, starting after Arrival
        At each step:
            Steps, Add: Station: a station name
            if processing time not the same for all entities that visit
            this station:
                Assignments, Add: Assignment Time: Attribute
                    Attribute: Process Time
                    Value: a distribution function
```

To create a visually pleasing result, Routes from the Animate panel could be added to connect the stations that are visited.

4 SOME EXAMPLES

The following examples are taken from (*Course Arena version 3.0*, 1997) and (Kelton, W.D., R.P. Sadowski, and D.A. Sadowski 1998).

In this section we will look at a possible implementation of this part of the modeling process.

4.1 The First Example

In the first example, see Figure 8, typical metal parts (T) are painted in three stages: preparation, paint, and dry. Some of the parts are special (S), don't get a preparation, are painted immediately and are given a second coating before they are dried. This example fits scenario 2.

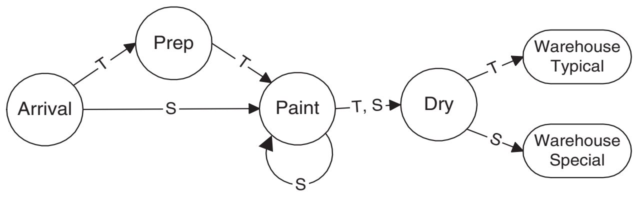

Figure 8: The Paint Model

Enter the number of locations at which an activity takes place: **3**
Name location 1: **Prep**
Name location 2: **Paint**
Name location 3: **Dry**

How many different kinds of types will be processed: 1
Name the entities of type 1: **Metal Part**

Name the point at which <u>Metal Part</u> enters the model: **Arrival**
Batch size of <u>Metal Part</u>: **1**

Does <u>Metal Part</u> have subtypes? **Yes**
How many subtypes are there of <u>Metal Part</u>: **2**
Name subtype 1 of <u>Metal Part</u>: **Typical**
Fraction of occurrence: **0.9**
Name subtype 2 of <u>Metal Part</u>: **Special**

Is there a need to distinguish between the (sub)types as they leave the system (e.g., physically or statistically)? **Yes**
Enter the number of locations at which (sub)types leave system: **2**
Please specify exit location 1: **Warehouse Typical**
Please specify exit location 2: **Warehouse Special**

Specify the inter-arrival time of <u>Metal Part</u> at <u>Arrival</u>:
Exponential(6)

Is the processing time at <u>Prep</u> identical for all types?
Yes, Triangular(4,5,6)
Is the processing time at <u>Paint</u> identical for all types? **No**
Is the processing time at <u>Dry</u> identical for all types? **Yes, 5**

Can the activity of Prep fail? **No**
Can the activity of Paint fail? **No**
Can the activity of Dry fail? **No**

Specify the route for Metal Part, Typical from Arrival Typical:
Prep
Paint
The processing time at Paint for Typical: **Normal(10,2)**
Dry
Warehouse Typical

Specify the route for Metal Part, Special from Arrival Special:
Paint
The processing time at Paint for Special: **Triangular(7,9,11)**
Paint
The processing time at Paint for Special: **Triangular(3,4,5)**
Dry
Warehouse Special

4.2 The Second Example

The second system we want to model is example 5.1 from Simulation with Arena (see Figure 9), in which two different electronic units are preprocessed and then sealed; the sealer machine may fail. Units that have been sealed successfully are shipped. Failed units are reworked in an attempt to fix the problem; some are reworked successfully and then shipped, the others are scrapped.

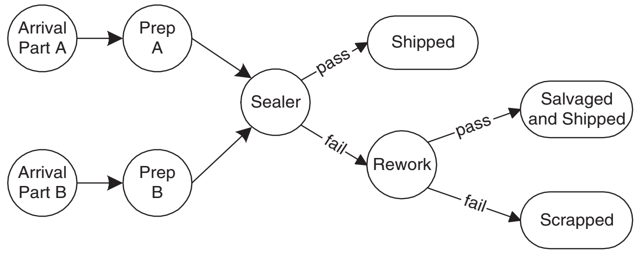

Figure 9: The Sealer Model

Enter the number of locations at which an activity takes place: **4**
Name location 1: **Part A Prep**
Name location 2: **Part B Prep**
Name location 3: **Sealer**
Name location 4: **Rework**

How many different kinds of entities will be processed: **2**
Name the entities of type 1: **Part A**
Name the entities of type 2: **Part B**

Name the point at which Part A enters the model: **Arrival Part A**
Batch size of Part A: **1**

Name the point at which Part B enters the model: **Arrival Part B**
Batch size of Part B: **4**

Does Part A have sub-types? **No**
Does Part B have sub-types? **No**

Is there a need to distinguish between the (sub)types as they leave the system (e.g., physically or statistically)? **Yes**
Please specify the exit locations:
Shipped
Salvaged and shipped
Scrapped

Specify the inter-arrival time of Part A at Arrival Part A:
Exponential(5)
Specify the inter arrival time of Part B at Arrival Part B:
Exponential(30)

Is the processing time at Part A Prep identical for all types?
Yes, Triangular(1, 4, 8)
Is the processing time at Part B Prep identical for all types?
Yes, Triangular(3, 5, 10)
Is the processing time at Sealer identical for all types?
No
Is the processing time at Rework identical for all types?
Yes, Exponential(15)

Can the activity of Part A Prep fail? **No**
Can the activity of Part B Prep fail? **No**
Can the activity of Sealer fail? **Yes**
Can the activity of Rework fail? **Yes**

Specify the route for Part A from Arrival Part A:
Part A Prep
Sealer
The processing time at Sealer for Part A: **Triangular(1, 3, 4)**
What is the probability of failure for Sealer: **0.09**
Continue with the route for entities that pass inspection at Sealer:
Shipped
Specify the route for entities that failed inspection at Sealer:
Rework
What is the probability of failure for Rework: **0.20**
Continue with the route for entities that pass inspection at Rework:
Salvaged and Shipped
Specify the route for entities that failed inspection at Rework:
Scrapped

Specify the route for Part B from Arrival Part B:
Part B Prep
Sealer
The processing time at Sealer for Part B: **Normal(2.4, 0.5)**
Thank you, the routes after Sealer have already been determined.

5 IMPLEMENTATION ASPECTS

The approach described in section 3 can be formalized and programmed. Input of such a program is the information provided by the modeler (as indicated in sections 2 and 4). Output could be provided at several levels. It could be a list of suggestions with respect to the number of modules used, the values for the various parameters, etcetera. The modeler could use these to manually build the model in Arena. A more sophisticated program could do the Arena programming.

Most programming languages would do the job. However, it seems more appropriate to use a knowledge-based tool. Even in these simple cases some reasoning is required and the best type of processing that can deal with this is known as knowledge-based processing. The need for this will increase when more types of modules or modules taken from less generic template panels are involved. In the discussion thus far we have not yet considered modules such as Statistics (with a rather complex structure) and Variables. We have restricted ourselves to the modules provided at the Common template panel, the most generic level in Arena. Each of these modules provide more functionality at the price of less flexibility. Modules from more specialized template panels such as the Transfer (e.g., containing the heavily used modules Transporter and Conveyor), the Blocks and the Elements template panel level are more specialized and allow the modeler to customize the model to his needs, see also (Pegden, et al. 1995). This will make the model much more complex than we have discussed thus far and therefore the reasoning process that should support the modeler will also have an increased complexity.

We have indicated that a knowledge-based approach would assumable be more appropriate to support the modeler and briefly discuss the various methods. Essentially there are two ways of knowledge-based reasoning: goal-driven and data-driven. Also a combination of these methods is possible and sometimes necessary.

The goal-driven approach is characterized by the presence of a set of hypotheses (goals), one or more of which represent the solution of the problem. In the context of this paper relevant hypotheses are a (large) number of simulation models which need to be known in advance. This approach is discussed in (De Swaan Arons 1999). In brief: a knowledge-based system selects one of the hypotheses (a simulation model) and attempts to gather the necessary information (either from the user, a database or some other data source) based on which the hypothesis can be either adopted or rejected. In order to achieve such a result the knowledge-based system follows a reasoning process based on a number of business rules and a number of facts that are already known to the system in advance or will be retrieved from some data source (for example, a database) during processing.

In contrast to this there is the data-driven approach. This is particularly appropriate in those cases in which the number of solutions are essentially infinite. This is the case in modeling when no models exist that could match with the model the modeler is after. A knowledge-based system could help the modeler to create a simulation model from scratch. Again, the knowledge-based system will help to create the right model based on a large list of answers to questions asked. The result is achieved by a reasoning process based on a number of business rules.

Generally, the goal-driven approach is characterized by a relatively small number of mostly coherent questions. The reasoning process takes care of this. It also prevents the user to be requested about information that is not relevant to the problem at hand.

In contrast to this, the data-driven approach generally requires a lot of information from the user and this information could be superfluous.

Naturally, when no simulation models exist from which the right one must be selected (so the set of hypotheses is empty), the goal-driven approach is useless and the data-driven approach is the only one that can be used.

6 CONCLUSIONS

The present article describes an approach that supports the modeler to build an implementation model in Arena based on a conceptual discrete-event model. In fact, it can be considered as a first step towards the automatic implemention of a conceptual model in Arena. At first sight it seems to be only a matter of implementation but it is not. Any simulation language has its own characteristics that may affect the contents of the conceptual model. In this respect, the support described affects both the conceptual and the implementation model.

The present research has been restricted to those models that could be built by using only the Arena modules at the most generic level of aggregation, i.e., modules that have a high degree of built-in functionality. These modules are found in Arena's Common template panel. Even with these modules some impressive models can be built. The approach described in this article offers a framework to automatically build this type of models.

Most simulation models can only be built by using modules at a less generic level of aggregation. At the cost of less built-in functionality, they offer the modeler more flexibility.

Based on the results described in this article a continuation of this research could be expected to be useful. This will be done along two tracks. First the approach described in this article will be automated. Furthermore, the research described in this article will be extended to models that can only be built by using modules at a less generic level of aggregation.

7 LITERATURE AND REFERENCES

Course Arena version 3.0, 1997.

De Swaan Arons, H. 1983. *Expert systems in the simulation domain*. Transactions IMACS: Mathematics and Computers in Simulation 25-1: 10-16.

De Swaan Arons, H. 1999. Knowledge-Based Modeling of Discrete-Event Simulation Systems, In Proceedings of the 1999 Winter Simulation Conference, ed., John Charnes, Douglas Morrice, Dan Brunner, and James Swain.

Developing Applications with the Aion Development System 1996. Student Guide, version 2.0 (course material related to AionDS version 7.0), Platinum Technology.

Kelton, W.D., R.P. Sadowski, and D.A. Sadowski. 1998. Simulation with Arena. Boston: McGraw-Hill.

Pegden, C.D, R.E. Shannon, and R.P. Sadowski. 1995. Introduction to Simulation Using SIMAN. New York: McGraw-Hill Inc.

Toolkit 1997. Logistic problems and simulation models. TNO Inro, Internal report 97/NL/112, (in Dutch).

Taylor ED 1998. User manual. Utrecht: F&H Simulation BV.

AUTHOR BIOGRAPHIES

HENK DE SWAAN ARONS graduated in Applied Mathematics at Delft University of Technology in 1972. From that time till 1982 he was appointed lecturer at the Faculty of Mathematics and Computer Science of this university with teaching and research in the field of parallel computation, modeling and simulation. Since 1982 he concentrated on research and teaching in the field of expert systems. He was the project leader of several research projects under which two Esprit projects on knowledge-based scheduling in manufacturing. In 1991 he got his Ph.D. degree in Computer Science at Delft University of Technology. The thesis was mainly concerned with the design, applicability and applications of expert system tools, in particular the Delfi3 system. Since 1992 he is an associate professor at the Department of Computer Science of the Faculty of Economics at Erasmus University Rotterdam. At present, his research focuses on simulation and expert systems, with the emphasis on economical applications. His email address is `<deswaanarons @few.eur.nl>`.

EELCO VAN ASPEREN graduated in Business Computer Science at Erasmus University Rotterdam in 1993. From 1991 to 2000 he was a member of the IT support group at the Erasmus University Rotterdam, for the department of Computer Science and from 1995 for the Faculty of Economics, focusing on the design and implementation of large scale computer facilities. Since January 2000 he is an assistant professor at the Department of Computer Science of the Faculty of Economics at Erasmus University Rotterdam. At present, his research focuses on simulation. His email address is: `<vanasperen@few.eur.nl>`.