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

LANGUAGE BASED SIMULATION MODELS AS MANAGEMENT TOOLS FOR ASSEMBLY LINES

Thomas Schulze

University of Magdeburg
School of Computer Science
PSF 4120
39102 Magdeburg, GERMANY

Marco Schumann

Fraunhofer IFF
Planning and Visualization Techniques
Sandtorstrasse 22
39106 Magdeburg, GERMANY

Gordon D. Rehn

Process Engineering
Deere & Company
One John Deere Place
Moline, IL 61265, U.S.A.

ABSTRACT

This paper demonstrates that despite the trend to Point & Click environments, the traditional approach of using general-purpose simulation languages is still eligible. The authors share their experiences gained from building a complex simulation using the language SLX^{TM}. On the basis of examples from the projects, the efficient modeling features of the SLX languages are highlighted.

1 INTRODUCTION

Simulation models for operational use in manufacturing systems are becoming increasingly important. Typically, this type of models is developed to support the management of manufacturing systems. Therefore, we call this model category Management Simulation Models (MSM). At the beginning, we will explain the uniqueness of this category and we will answer the question: Why it is necessary to develop such models? Following, we specify requirements for simulation systems to fulfill the specific circumstances for MSM. Then, we describe an assembly line example to present some specific problems that have to been modeled. Finally, we illustrate solutions for selective problems.

2 MOTIVATION

Manufacturing and material handling systems are the best known applications of simulation models. See Law and McConas (1999) and Rohrer (1998) for a more detailed description. Simulation models for manufacturing systems can be divided into three general categories: design of systems, management of systems and training the staff in systems. In recent years most simulation projects are in the first category.

Currently a new trend can be observed. An increasing number of simulation models will be used in the day-to-day operation of manufacturing facilities. These models help manufacturers to evaluate the system capacity for new orders, for changes in the operator team and for changes in operating conditions. They support the management of manufacturing systems for analysis of throughput and detection of bottlenecks. Management can evaluate operating decisions relative to the performance of the system. In general the aim of models in this category is to support the management. Management wants to gain experiences from the future. We call this model category Management Simulation Models (MSM).

MSM's can be exploited both in the design phase and in the operating phase of a factory. Once the time is invested to build the original model for the design, continued use of the model as a MSM maximizes the value of that investment. Also, involving operating people in the training of the MSM improves the system understanding and will increase the chances that the simulation results will be used.

Do new requirements, new needs exist for Management Simulation Models relative to the classic manufacturing simulation models? We will answer yes and identify two differences. First the MSM require greater level of detail than models for manufacturing system design. For example, complex control mechanisms and strategies may have to be implemented. Second, the MSM must be initialized with the state of the real system. The simulation can start from a null and idle status and runs until it reaches the current state of the real system, or the model can be initialized directly with the state of the real system.

Law and McConas (1999) identify two types of simulation packages for manufacturing simulation: general-purpose simulation packages like ARENA, AweSim, MODSIM III, Simple++ and SLX. The other type is application oriented simulation packages like AutoMod, AutoSched, ProModel and WITNESS. The advantages and disadvantages of both categories have often been discussed in the past. Whereas users of general purpose simulation packages value the flexibility of their modeling package, users of specialized simulation package often argue that they can very rapidly construct their models by taking advantage of already predefined model elements of domain specific libraries. On the other hand, the advantage of using a library of model elements can easily turn into a disadvantage when a model element does not support a desired behavior. The flexibility of a general-purpose package can create a substantial amount of work when even the simplest model element needs to be developed from scratch. The choice of a package from one of these categories seems often to be a matter of personal preference, modeling skill, and simulation experience.

We will contribute to the discussion by presenting our point of view focused on the area of MSM. When predefined model elements can not attain the desired level of abstraction, then the elements have to be adapted or new ones have to be created. It is easy to use parameters for changing attribute values but difficult to adapt complex control and decision strategies. Management simulation tools require a high level of detail and the needed control strategies can not often build up on default strategies. The boundaries of adapted or updated strategies have been reached very quickly.

An alternate for MSM is the use of general simulation packages. This software type can be subdivided into language and graphical oriented packages. We prefer and recommend the use of language oriented packages. One reason is the modeling of complex control and decision strategies. These strategies can be modeled better in language constructs than in graphical description.

In the next section, we describe the derived requirements for simulation languages used for development and application of MSM.

3 REQUIREMENTS FOR SIMULATION LANGUAGES AND MODELS

Simulation languages in general offer services to the following subtasks in simulation models: define, create and destroy simulation objects; data manipulation; time advance and stochastic treatment. At first we will analyze the necessity of these subtasks for using in the development process of MSM. In the second part we will point out which general services of the simulation models would be useful for applications.

3.1 Requirements during the Developing Process

3.1.1 Create and Destroy Simulation Objects

The simulation language must offer the use and the creation and the destruction of different types of simulation objects. The types differ in their abstraction level.

The lowest level covers well known simple data types (integer, float and string) and complex data types (arrays, records and objects with different attributes). This level is necessary for MSM. The next abstraction level includes objects that can be moved through the model. These objects are often called entities. This level must be available for MSM. The following abstraction level span resource-oriented objects like machines, transporters and operators. This level has not been available because implemented level of detail inside these objects is often too low for MSM users.

3.1.2 Data Manipulation

A high level of detail requires a lot of data to describe the system-state. Every change of the system must be reflected in the related data structures. Simulation languages have to offer effective statistic methods for reading and writing data, updating data and dynamic methods for data manipulation like sorting and grouping. A large part of written lines in MSM, often about 80 %, are related to data manipulation. Effective methods must be available.

3.1.3 Time Advance

Time advance in simulation models can be classified into time-delays, condition-delays and dormant-delays (Schriber and Brunner 1996). Condition-delays are the most used form in MSM. This class is important for the description of detailed control and decision strategies and we will discuss only this kind.

Simulation languages offer two different possibilities for modeling such condition delays. The first possibility is the automatic monitoring of selected data by the simulation system. Examples in the GPSS-language are the blocks TEST and GATE and in the SIMAN-language is the SCAN-block. The other possibility is the user-written monitoring. The user decides when the condition will be satisfied. The SIGNAL- and WAIT-Blocks are examples in the SIMAN-language. The automatic monitoring will be preferred for clear and well-structured models. The implementation of automatic monitoring differs in the simulation language world. (Schriber and Brunner 1999). Multiple execution of complex condition-delays can lead to an increasing computing time (Schulze and Preuß 1997). Effective implementation, like the use of control-variables in SLX, should be available for fast runs of MSM.

3.1.4 Stochastic Handling

Stochastic features play a negligible role in MSM. With an increasing level of detail and decreasing time horizon of model the relevancy for stochastic features becomes less important.

3.2 General Services for Applications

The developed simulation models should be characterized by following services:

3.2.1 User-Friendly Support for Input and Output Data

The user of MSM want to modify the input data in their known environment. Spreadsheets and databases are well known and used features for input and output data. Simulation models must be offer interoperability to other software systems.

3.2.2 Animation

Animation of the manufacturing processes is not necessary for every application. In many cases, however, it is very helpful for the management to explain new situations in the manufacturing process using an animation and depict how problems can be solved.

3.2.3 Intranet Environment

This is a look into the future. It would be desirable to use web-based architectures for the application of MSM. The simulation model is located on an Intranet-server. All advantages of client-server architectures could be used in this case. The manager on the workplace would not be stressed with license, security-key or maintaining problems.

4 DESCRIPTION OF THE ASSEMBLY-LINE EXAMPLE

Speed of model development is one of the often emphasized advantages of the „Point and Click" simulation development environments. Users of such environments can easily click different kinds of predefined resources into his/her model. Typically, resources can be configured with an extensive set of parameters, such as input/output buffer size, loading time, processing time and many more. In the case of factory simulation, most tools allow to connect resources by edges denoting the direction of material flow. Let's consider the simple example of an assembly line where a product has to pass all stations in a sequential order. Clicking in, parameterizing, and connecting resources can be accomplished in a matter of a few minutes and the simulation model is ready to run. Figure 1: SEQARABIC shows a screen shot of a Simple++ simulation model. Even a beginner would probably be able to accomplish such a model in less than five minutes.

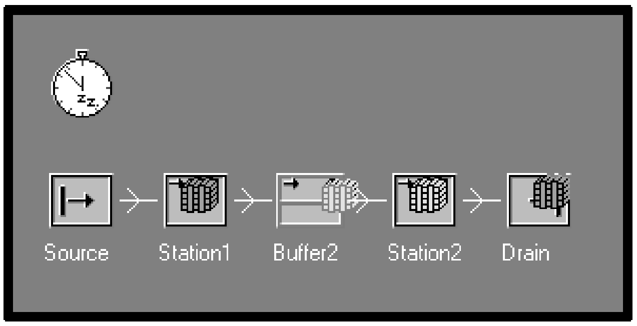


Figure 1: Screenshot from Simple++

However, this kind of examples is rather academic in nature. Most scenarios will require a modification of the standard model elements. Sooner or later one will reach a point where model elements need to be extended to match a certain application. This is where the power of a modeling language comes into play.

Now, let's consider a model of assembly stations that involves a couple more realistic constraints. How fast can one still develop the model using predefined model elements? The assembly line is characterized by the following facts:

- The assembly stations are connected with a continuously moving conveyor. The speed of the conveyor is automatically adjusted to the desired number of products to be produced.
- Processing time at each assembly station varies depending on the options of the manufactured product. In order to compensate for very time-consuming tasks, some tasks may interfere with tasks to be executed at the next station, i.e. the product is already moving to a certain extend into the next station.
- Some stations, however, require the corresponding tasks to be completely finished before moving into the next station. This is due to special equipment needed at these stations.
- If delays accumulate and the product moves into the next station too far, the conveyor needs to be stopped until all work of the previous stations is finished.

- Work can only be carried out when a worker is available. Each task requires a corresponding qualification.
- In case there are no new tasks to do for a worker, he/she will assist one of his/her co-workers and therefore shorten the process time through team-work.
- After a break, each worker resumes where he/she left off. Jobs not being finished at the end of a shift are carried over into the next shift which can be composed of a different number workers with a different combination of qualifications.
- Shifts are comprised of four types of segments. They start with a start-up segment followed by a productive (work) segment and end with a clean-up segment. The work segment is further divided by inserting break segment of different length. Depending on their station workers can have their breaks at different times.

5 DETAILED PROBLEM SOLVING

In this section, we choose a few of the above problems and demonstrate their implementation. Of course, we do not claim that these problems can only be solved with the simulation system we selected. Most likely, they can be implemented using any commercial manufacturing simulation system.

From our point of view, however, there are vast differences between the simulation systems with regard to how easy it is for the simulation developer to step down one abstraction layer if the pre-defined model elements do not match the specific requirements. Basically, all simulation systems provide this possibility. ARENA, for instance allows use of different panels and thereby allows programming at the SIMAN abstraction level. If a problem still cannot be solved at SIMAN level, almost unlimited flexibility is gained by including C code into the simulation.

No doubt, any kind of problem can be accomplished in other simulation systems. The only question is how much effort one needs to put in creating the simulation model.

The authors think that the simulation system SLX provides a very efficient approach. Like other systems, SLX allows for programming at different abstraction layers. The advantage is, however, that there is no paradigm shift when changing between these layers since all of them are programmed using the same language.

The remainder of this chapter will give insight to a few implementation features in SLX language.

5.1 Controlling of the Material Flow

In our model, the material flow is simulated using the object classes for a conveyor, assembly stations, carts, and units (the actual product). Whereas the conveyor and the carts are active objects, assembly stations and units are implemented as passive objects.

The conveyor continuously moves the carts until it detects an exceptional condition that cause the conveyor to stop. The carts monitor the current position on the conveyor and generate new task orders when they enter the next assembly station. Once a new task order is issued, workers check with their qualification and start working on a task accordingly.

All these model elements depend on each other and need to be coordinated. Therefore, powerful mechanisms of expressing conditions are needed. In a computer simulation, model elements often have to wait for a condition that will be fulfilled at a currently unknown time in the future.

The above example of an action property was taken from the conveyor code and gives an example for a conditional wait. Execution of the code is delayed until the condition becomes true. The variable forming the condition can be modified by an external event. The key word control informs SLX to observe the value of the variable WorkPeriod. In case of changes, SLX automatically reevaluates the condition in the wait until statement and resumes the movement of the corresponding puck accordingly. The wait until statement is one of SLX’s most powerful statements.

5.2 Controlling of Workers

In our simulation, workers are modeled as active objects that can have four different states: passive, available, working and interrupted. Figure 2 illustrates possible transitions between these states.

Each worker is available for only one shift a day. At off-shift times, a worker remains in the state “passive”. When the corresponding shift begins, workers become “available”. An available worker starts working in case there are pending tasks that match his/her qualification. If there is no more work to do, the worker returns back to the available state. While a worker is busy doing a task, he/she might be interrupted by a break. At the end of a break the worker resumes where the task was interrupted.

It is not the intention to present each transition in detail. Instead, we would like to point out a few key feature of the SLX kernel language.

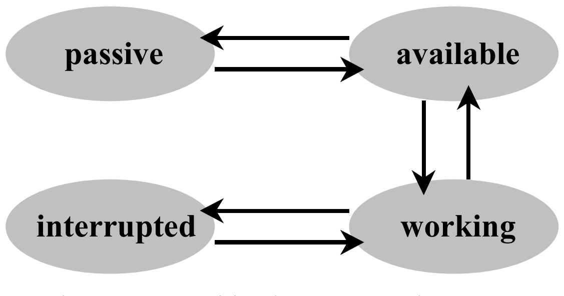


Figure 2: Transition between Worker States

Workers are modeled as active elements. When a worker starts to work on a task, the time needed to accomplish that task is calculated and a scheduled time advance is issued. The actual time needed for a task, however, depends also on the shift schedule (break times) and on the availability of other workers who may cut down the time for a task. Especially the availability of additional workers is difficult to determine in advance. Therefore, flexible mechanisms for rescheduling are essential. In the remainder of this section, we will present two approaches how rescheduling can be describe in the SLX language.

The first approach uses the interrupt / resume statement pair. At the beginning of a break, the corresponding scheduled puck for the worker is interrupted by the shift control. SLX automatically stores the amount of time left until the puck was originally scheduled. At the end of the break, the shift control resumes the worker's puck. The previously stored remaining time is added to the current clock time and the puck is therefore rescheduled. Figure 3 illustrates this idea.

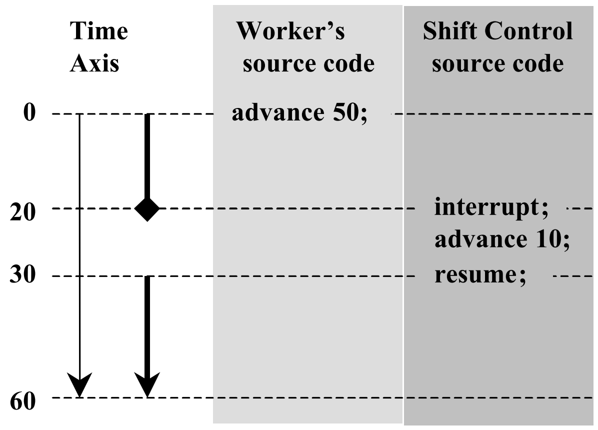


Figure 3: Interrupt and Resume for Controlling of Workers

At time 0, the worker starts working on a task that takes 50 time units to be completed. After 20 time units, a break for 10 time units occurs. The worker's puck is interrupted at time 30 and resumed at time 40 by the shift control. The time for the task's completion will therefore be postponed to time 60.

A second approach for rescheduling pucks is the reschedule statement. Whereas the interrupt/resume combination is useful when the amount of time spent for a certain task remains unchanged, the reschedule statement can be used in case the time spent on a task needs to be modified. This, for example, occurs when a worker receives assistance from a co-worker.

Assuming the above example, the worker is assisted by a co-worker beginning at time 20. Both workers are able to finish the remaining work in the 2/3 of the time. In the original situation, there would be work for another 30 time units. Having assistance cuts the remaining time down to 20 time units and both workers are now finished at time 40. This is depicted in Figure 4.

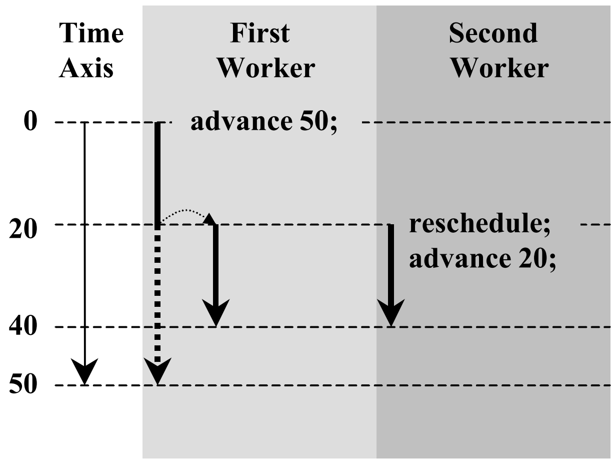


Figure 4: Rescheduling of a Worker's Puck

5.3 Shift Scheduling

Productive times of the workers are defined by the shift schedule. There can be up to three shifts a day. As mentioned above, in our model a shift is comprised of four different types of segments. At the beginning and the end of each segment, a corresponding procedure will be activated that describes the actions to be taken in response of that event.

A sequence of segments beginning with a start-up segment and ending with a clean-up segment is called a module. Since not all workers in a factory have to follow the same schedule, there may be parallel modules within a shift. Each worker is assigned to one shift and one module within that shift.

When designing a simulation the author always has to decide whether to implement a certain class as an active or passive object. Only active objects are able to advance in time, e.g. experience scheduled or state based delays. Although it may seem natural to describe an object's behavior as active, too many active objects can become an obstacle. In case of errors, a simulation run with many active objects is difficult to debug. Also, the question of how to handle simultaneous events is more complex with a large number of active objects.

At first glance, one might suggest implementing the shift schedule using active objects and create an instance for each module within a shift since several modules exist in parallel but progress in different time steps.

Here, SLX provides an innovative approach of expressing this type of parallelism (Henriksen 1996). SLX uses pucks to keep track of active objects. When an active object is instantiated a corresponding puck is created at the same time and points at the first line of executable code for that object.

In our model, only one active object is employed for the shift schedule. Parallel modules are expressed by using a mechanism that is called internal parallelism, which is executed through a fork statement:

```
    fork
    {
        offspring actions
    }
    parent
    {
        parent action
  }
```

Execution of the fork statement creates an additional puck for the currently active object. The newly created puck is placed in the active puck list, poised to execute the offspring action clause of the fork statement. The parent puck executes the optional parent action clause.

Applied to the shift schedule this mechanism can be utilized as follows: Actions to be taken at the beginning of a segment are executed before the fork statement. Then, the fork statement is issued. The offspring puck advances to the segment's end, executes the segment's end actions and terminates. "In parallel", the parent puck advances to the next start time of a segment from any module and the same procedure starts over. In order to assure that the final actions of the previous segment are always executed before the next segment's start actions, the priority of the offspring puck is raised. Therefore, at any given time, pucks poised for a segment's end action are considered before pucks poised for a segment start action.

The corresponding section of SLX code is shown in Figure 5. Using internal parallelism provides an efficient approach of modeling active objects and helps to limit the number of object instances. The fork statement only creates a new puck. All offspring pucks operate on the same set of data as the parent puck does.

5.4 Statistics and Outputs

Collecting, preparing and presenting of simulation data is an essential task in every simulation model. Three challenges from our simulation will be pointed out: statistics for different time periods, flexible management for statistical data and calculating not-standard parameters for queue statistics.

5.4.1 Statistics for different Time Periods

Creation of statistics for different time periods is a typical task inside MSM. Often theses time periods are linked to the shift regime. Every shift has its own conditions that influence the state of the system. For example the number of workers per shift is not equal in all shifts. The number of operators in the morning shift is higher than in the night shift. Different numbers lead to variable conditions for the utilization of the equipment. It was necessary to present shift-depended statistics. The traditional solution would be to define one statistic for each shift. Statistical data will be typically collected at several places inside the simulation model. Traditionally, a distinction would be necessary at each of these places in order to ensure the modification of the correct statistic.

SLX offers a more flexible and well-structured solution based on the statistic class random_variable and interval. Our approach comprises three steps (see Figure 6): The first step is the instantiation of the needed random_variable objects and one interval object. The number of random_variables objects depends on the equipment quantity. A second step includes the generation of a link between the random_variables and the interval object and finally the third step is the implementation of the shift control. There exist only one place in the SLX simulation model where shift changes have to be coded.

```
class cl_Schedule
{ ...
    for (CurrentEntry= each cl_ScheduleEntry in ScheduleEntries)
    {

        // advance to schedule entry's begin time.
        if (CurrentEntry->Start > TimeOfDay())
            advance (CurrentEntry->Start - TimeOfDay());

        DoSegmentStartActions( ModuleName, ScheduleName, Shift, SegmentType);

        // determine time of segment end
        Duration= CurrentEntry->Duration;

        fork
        {
            ACTIVE->priority++; // schedule segment end calls before start of new segment
            advance Duration;
            DoSegmentEndActions( ModuleName, ScheduleName, Shift, SegmentType);
            terminate;
        }
    }
 ...
    }
```

Figure 5: Implementation of the Schedule Class

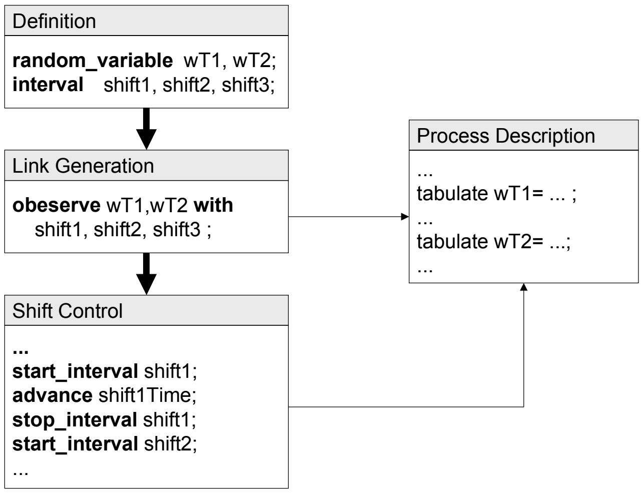
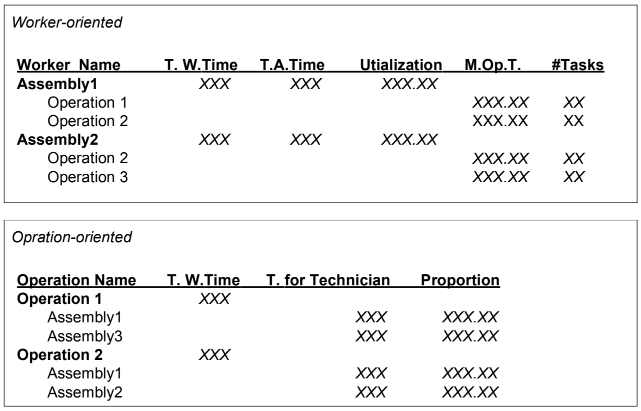


Figure 6: Three Step Approach for Statistics in Different Time Periods

5.4.2 Flexible Management for Statistical Data

The relevancy for flexible data structures and data presentation will be demonstrated in the following example. A worker holds different qualifications so he or she can perform different operations or tasks. There exist a 1:m relation between the worker and his qualifications. On the other side, the operations will be done by different workers. Here exist a 1:n relation between the operation and the workers. The execution of an operation can be concurrent (by parallel working) or sequential. Statistical data about the operating time must be collected and presented. Two different tables have to be used for presentation of the data. Although both tables use a different view, they basically present the same data. Whereas, one table is worker-oriented and is sorted by workers and operations, the second table presents the data from an operation-oriented view and is sorted by operations and workers. See Figure 7 for table structure examples. The worker-oriented table allows management for detecting bottlenecks and the second table offers additional information on the human resource for solving the process.

The challenge writing the simulation model is to find flexible and efficient data structures as a basis for the required presentation forms. The classical solution is the usage of a two-dimensional (n,m) arrays for each statistical parameter, such as minimum, maximum, mean etc. One has to bear in mind that these are sparse arrays because every worker can hold a maximum of five qualifications. Additionally, the matrix-approach is not flexible and requires changes in the source code to adjust the matrix size.


Worker-oriented

Worker Name	T. W.Time	T.A.Time	Utialization	M.Op.T.	#Tasks
Assembly1	*XXX*	*XXX*	*XXX.XX*		
Operation 1				*XXX.XX*	*XX*
Operation 2				XXX.XX	XX
Assembly2	*XXX*	*XXX*	*XXX.XX*		
Operation 2				*XXX.XX*	*XX*
Operation 3				*XXX.XX*	*XX*

Opration-oriented

Operation Name	T. W.Time	T. for Technician	Proportion
Operation 1	*XXX*		
Assembly1		*XXX*	*XXX.XX*
Assembly3		*XXX*	*XXX.XX*
Operation 2	*XXX*		
Assembly1		*XXX*	*XXX.XX*
Assembly2		*XXX*	*XXX.XX*

Figure 7: Examples of Structures for Result Tables

With SLX, better approaches can be implemented which will be presented here. The basic idea is to use two set ranked by different attributes of the contained objects. These objects consist of two key-attributes for unique identification and one random_variable object attribute for collecting the observed data. The following SLX source code shows the definition of the utilized class WT_Statistics.

```
   class WT_Statistics
   {
      string(30) WorkerName;
      string(30) OperationName;
    random_variable WorkTime ;
 }
```

The attributes WorkerName and OperationName uniquely identify the object. In case a new combination occurs, a new object will be instantiated dynamically. References to the objects will be inserted in both sets. The object physically exists only once but references are stored in the both sets. The definition of these sets is as follows:

```
set (WT-Statistics ) ranked
(ascending WorkerName, ascending
OperationName)
WorkerStatistic;

set (WT-Statistics ) ranked
(ascending OperationName, ascending
WorkerName)
OperationStatistic;
```

Each set has two keys for defining ascending sort order for the contained object references. The primary key for the set WorkerStatistic is the attribute WorkerName. Alternatively, the primary key for the set OperationStatistic is the attribute OperationName. The two sets are the basic objects used for the report generation. Figure 8 displays the used approach.

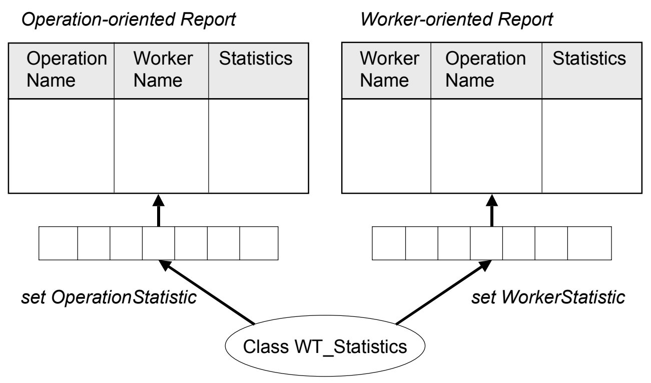


Figure 8: Process of Generating the Output Tables

5.4.3 Calculating Non-Standard Parameters for Queue Statistics

Queues are typical elements for modeling waiting areas of entities. The waiting behavior of the entities will be observed during the simulation run. Statistical computations will be done at the end of run and these results are known as standard parameters. The standard parameters characterize the waiting behavior of queued entities.

One goal in our simulation was to track additional parameters about waiting zones. For example, the simulation has to gather data about the total empty times of the waiting area. Standard parameters for queue statistic contain values for the number of zero-entities and the mean for waiting times for entities. But these parameters are not related to the waiting lines.

Our solution was the definition of the new object class waiting_room based on the existing default class queue. The composition-approach was used to define the new class. The new class consists of the existing class queue and further necessary attributes. The calculated new two non-standard parameters are the total time of empty space in the waiting line and the utilization of the waiting line as a relation between the empty time and the occupied time. The following SLX code shows the definition of the new class.

```
class waiting_room
{
    queue st_queue;
    pointer (*) lastObject ;
    float lastEmptyTime ;
    float emptyTime ;
    boolean empty ;
}
```

6 CONCLUSION

Management Simulation Models (MSM) are characterized by a high level of detail. Manufacturing oriented simulation packages have limits if existing model elements can not match the required degree of fidelity. Often, there is too much effort needed to modify these model elements. One known solution is the use of flexible simulation languages. The simulation languages have to fulfil requirements like management of simulation objects, flexible data structures and effective features for modeling conditions-delays.

Our experiences derived from the simulation project are that SLX meets the requirements. The model execution speed was extremely fast regarding the complexity of the model. A focus of future work in the area of MSM is the reduction of the amount of time spent to build the model. This goal can be reached by using more computer-based tools for model generation. The extensibility features of SLX offer basic mechanisms for starting the development of necessary tools.

ACKNOWLEDGMENTS

The authors would like to thank Deere & Company for granting permission for publication of this material.

REFERENCES

Henriksen, J. O. 1996. An introduction to SLX^{TM}. In *Proceedings of the 1996 Winter Simulation Conference*, ed. J. Charnes, D. Morrice, D. Brunner, and J. Swain, 468-475. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Law, A., and M. McComas, 1999. The simulation of manufacturing systems. In *Proceedings of the 1999 Winter Simulation Conference*, ed. P.A. Farrington, H. B. Nembhard, D. T. Sturrock, and G.W. Evans., 56- 59

Rohrer, M. 1998. Simulation of Manufacturing and Material Handling Systems. Chapter 14 in *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, ed. J. Banks, 519-546, New York, New York: John Wiley & Sons, Inc. 1998.

Schriber, T. J. and D.T. Brunner. 1996. Inside simulation software: How it works and why it matters. In *Proceedings of the 1996 Winter Simulation Conference*, ed. J. Charnes, D. Morrice, D. Brunner, and J. Swain, 23-30. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Schriber, T. J. and D.T. Brunner. 1999. Inside simulation software: How it works and why it matters. In *Proceedings of the 1999 Winter Simulation Conference*, ed. P.A. Farrington, H. B. Nembhard, D. T. Sturrock, and G.W. Evans, 72-80. Winter Simulation Conference Board of Directors.

Schulze, T. and F. Preuß. 1997. Benchmarks für diskrete Simulationssysteme. In *Proceedings Fachtagung Simulation und Animation 97*, ed. O. Deussen and P. Lorenz, 43-55. SCS International, 1997.

AUTHOR BIOGRAPHIES

THOMAS SCHULZE is an Associate Professor in the School of Computer Science at the Otto-von-Guericke-University in Magdeburg. His research interests include modeling methodology, public systems modeling, manufacturing simulation, and distributed simulation with HLA. He is an active member in the ASIM, the German organization of simulation. His email and web addresses are <tom@isg.cs.uni-magdeburg.de> and <www-wi.cs.uni-magdeburg.de/>.

MARCO SCHUMANN is an employee at the Fraunhofer Institute in Magdeburg, Germany. He holds a Master's degree in Computer Science from the Otto-von-Guericke-Universität Magdeburg. His experiences in developing simulations and applications for the Internet include a one-year-stay at the University of Wisconsin - Stevens Point. His main research interest lies in application of simulation methods in the field of factory planning and virtual training environments. His email and web addresses are <www.cs.uni-magdeburg.de/~maschuma> and <schuma@iff.fhg.de>.

GORDON D. REHN is a Staff Engineer and Process Owner of Modeling & Optimization analysis in the Process Engineering Dept. of Deere & Company, the worldwide corporate headquarters of John Deere. He received his B.S.M.E. from Iowa State University, and is a registered Professional Engineer in the State of Illinois. He is a member of IIE. He has preformed discrete event simulation analysis of manufacturing operations since 1976 for internal Deere projects, as well as consulted on simulation projects outside the Deere organization. His email address is <gr54556@deere.com>