TAKING THE WORK OUT OF SIMULATION MODELING: AN APPLICATION OF TECHNOLOGY INTEGRATION

Gregory S. Baker

Fluor Daniel Manufacturing Technologies
100 Fluor Daniel Drive
Greenville, South Carolina 29607-2762, U.S.A.

ABSTRACT

This paper presents an implementation methodology appropriate for providing a broad range of *proven*, classical Operations Research methods and techniques to the simulation modeler without imposing that s/he become proficient in a programming language. The user interface is enabled by ARENA's robust Applications Specific Template (AST) development environment. The simulation example highlights an inbedded optimization problem, the Knapsack loading problem. The methodology has been used to implement several other optimization algorithms and heuristics including a generalized linear programming algorithm.

1 INTRODUCTION

There is often a need to go beyond the intended algorithmic capability of a general purpose simulation package to perform complex mathematical analysis and/or computations. Some algorithms can become a monumental task when attempted in a simulation package like SLAM, SIMAN, WITNESS, GPSS, etc. To their credit, most simulation software packages provide limited access to languages like C or FORTRAN, both of which are better suited for mathematical and algorithmic implementations of complex numerical solutions. However, this programming approach using a separate programming language has significant drawbacks in today's simulation community where the graphical user interface (GUI) has become a standard.

This paper presents by example an implementation methodology appropriate for providing a broad range of *proven*, classical Operations Research methods and techniques to the simulation modeler without imposing that s/he become a proficient C or FORTRAN programmer. Furthermore, the modeler does not need to know much at all about implementing solutions to classical problems like linear programming, nonlinear or integer optimization, etc.. The implementation exampled herein is enabled by ARENA's robust Applications Specific Template development environment. This template development environment allows third party developers to extend the underlying SIMAN simulation language by providing classical and tailored algorithmic problem solutions to the model developer, presented with the same *look-and-feel* and *ease-of-use* as found in the standard Arena modeling environment.

2 EXAMPLE

Often there is a need to integrate complex numerical-logical algorithms within a simulation model for *what-next* type decision making. Take for example a general consumer products manufacturer with a distributed manufacturing and warehousing network. In addition to manufacturing needs analysis, a ship-to schedule and fleet size type questions, there is an embedded optimal truck loading problem. Pushing down from the macro problem of making it where and shipping what from where to where, we find a micro problem of,

> "...given these potential items to load onto a specific truck, which should be loaded to maximize, say, load profitability."

Load profitability may imply maximum truck load based on volume or weight to minimize the number of trips; or it could be based on order due date to maximize the total number of orders shipped on time; or more directly, a calculated truck load profit.

Whatever the criteria, a typical simulation language requires quite a bit of work to accomplish such an analytical and numerically complex task. Aside from a simple ordered queue a modeler would have to write a considerable amount of SIMAN code to insure an optimal loading procedure. The optimal truck loading

problem is a small piece of the overall system analysis, and because the modeler is not likely to be an expert in optimization theory or techniques, s/he usually opts for the simple ordered queue solution.

3 APPROACH

A better solution than coding complex numerical algorithms in a base simulation language like SIMAN is the integration of applicable optimization algorithms within ARENA, and providing a module access that has the same *look-and-feel* and *ease-of-use* found in the standard ARENA package. This approach removes the modeler's need to know and/or understand the algorithmic implementation of Operations Research techniques and perhaps, just as importantly, it removes the need for detailed, error-prone SIMAN programming, or for that matter, any programming at all.

Figure 1 below conceptually illustrates the exampled integration path. The modeler's same *look-and-feel* and *ease-of-use* as found in standard ARENA, is provided by ARENA's robust template development environment. The appropriate optimization technique is simply picked off the menu and the associated module(s) placed within the current model window.

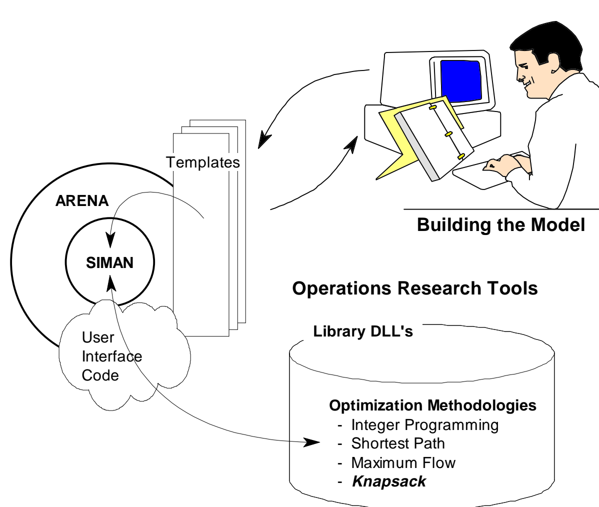

Figure 1: Conceptual Integration

Problem configuration and data communication with the embedded optimization algorithm is a matter of *fill-in-the-blank* from the end-user's point of view. The key is integrating appropriate, and proven, solution algorithms into ARENA via the user-code, C development environment.

In addition to examples and application or algorithmic level user documentation, on-line help files can be added as part of the standard interface.

4 PROBLEM STATEMENT

The following example is a prototype implementation. The application is a classical Knapsack problem, used here to describe an optimal truck loading procedure for the above general consumer products manufacturer and distributor.

In our consumer products manufacturing and distribution example above, the issue identified as, *"...maximum profit per truck load,"* is a form of the classical Knapsack problem. Simply stated, the knapsack problem is:

$$\max p[0]*x[0] + ... + p[i]*x[i] + ... \quad \text{for } i = 0, 1, ..., n-1;$$

where p[i] is the profit contribution of the ith load under consideration and x[i] is a binary value set such that 1 means that the ith load should be loaded onto the awaiting truck and 0 means that it should not be loaded. Therefore:

$$x[i] = 0 \text{ or } 1 \quad \text{for } i = 0, 1, ..., n-1.$$

The maximum profit is subject to the truck capacity. Using w[i] as the item weight and v as the maximum truck weight value, then the maximum profit is constrained by:

$$w[0]*x[0] + ... + w[i]*x[i] + ... <= v \quad \text{for } i = 0, 1, ..., n-1.$$

Implementing an applicable solution method is paramount to model accuracy. Most mathematical algorithms have a range of validity for a specific class of problems. In this example case, we have implemented an integer knapsack solution algorithm that requires integral item weights and profit attributes. That is,

$$p[i] \text{ and } w[i] \text{ positive integers} \quad \text{for } i = 0, 1, ..., n-1.$$

Another implementation condition on this particular algorithm is that each potential load item is in fact loadable. That is,

$$w[i] <= v \quad \text{for } i = 0, 1, ..., n-1.$$

Finally, we must start with a real problem. That is, all the available loads will not fit onto the awaiting truck, or

$$w[0] + ... + w[i] + ... > v \quad \text{for } i = 0, 1, ..., n-1.$$

The numerical calculations are quite extensive and implement a *branch-and-bound* solution technique. The original algorithm is given in Horowitz and Sahni,

(1974). This specific implementation is an adaptation of the C-code provided to the author by Bob Craig of Lucent Technologies.

5 INTERFACE

5.1 The Interface

But the implementation of an algorithmic solution to the Knapsack problem is only part of the total implementation effort. A significant concern is providing a simple interface to the optimization algorithm for the modeler. Such an interface may be constructed using the ARENA template development environment, provided with ARENA PE.

Before we develop the template, we must define the system interfaces requirements: template to algorithm, and modeler to template. The template to algorithm is straight forward, and fixed by our selected implementation. First we need to know the maximum truck capacity be it weight or volume, etc.. Secondly, we need to provide a list of items to be considered for loading. If we consider each load item as an entity, a queue provides a logical list structure for our potential loads. This queue even allows the loads to be user preference ordered. (This implementation does not effect the original load queue ordering which means that when we go to perform the actual truck loading activities, we have preserved our preferred load ordering over the solution space.)

It is important to remember that the optimal truck load solution is determined in zero [simulation] time, i.e. $D_t = 0$, but physically loading the truck will take some yet unspecified amount of time, either per load item or in aggregate. The Case II Pseudo Code (see page 6) makes use of the item loading time, $D_t > 0$, to iteratively optimize the remaining loadout by taking into account any newly available load items.

Since we are using entities as load items, entity attributes may be used to distinguish the individual load characteristics - weight and value. From the modeler, we need only know which two attributes reflect the load's weight and value respectively.

We also require the modeler to provide one more load item entity attribute which is used to designate our actual problem's solution space. Figure 2 shows a simple interface module screen to setup and solve our example truck loading optimization problem.

This is a zero-time module, so upon exiting the block, no time has elapsed and the entities in the specified load queue have simply been flagged as to-be or not-to-be loaded. The actual truck loading process will take some time either on a per load basis or in aggregate. Although our job of providing a solution

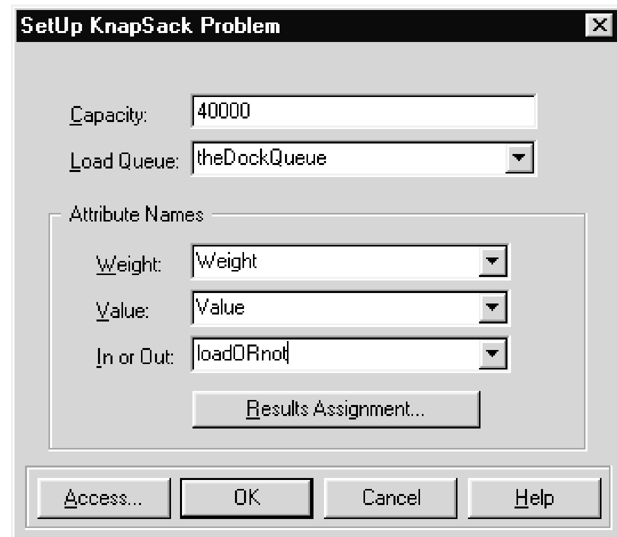

Figure 2: Sample Problem Definition/Input Screen

is complete, it would be nice to add a simple interface module that allows the modeler to pull one or more of the queued load items and send it/them on to a truck loading process. Figure 3 below shows a sample load selection module interface.

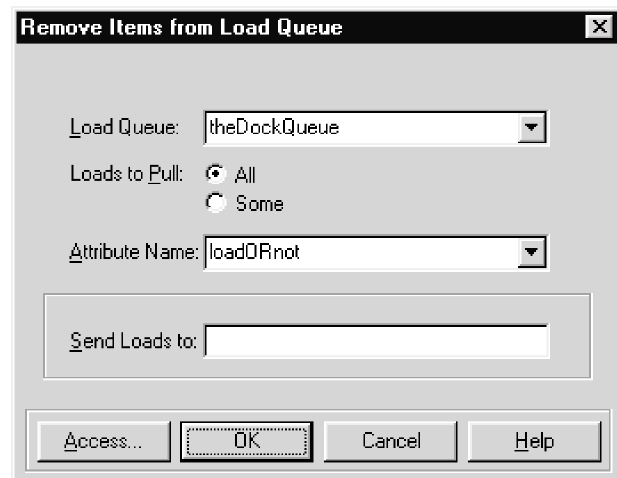

Figure 3: User Input Interface

As in the problem setup interface module, we need to provide the name of the queue where the load items reside and again, the attribute name of the load/leave flag. Finally, the modeler needs to specify how many loads to remove from the queue and send on to the loading process: the 'Loads to Pull' field. The re-

sponse ‘All’ is just that, remove all entities in the queue with their load/leave attribute set to one, and send them on to the next model block. No time delay is provided between load removals.

Load Queue: theDockQueue
Loads to Pull: All / Some 1
Attribute Name: loadORnot

Figure 4: ‘Some’ Option Selection

Another way to approach the loading process is via the ‘Some’ selection. Here the modeler may remove one load at a time (with supplied looping) allowing time for additional loads to be queued-up during the item’s loading delay cycle. Then, before the next load is selected, the problem can be reconfigured and solved again based on the new set of potential loads, and the truck’s remaining capacity, v_{new}. When the modeler selects the option ‘Some’, a quantity field becomes visible and is initialized to one as shown in Figure 4 above.

5.2 Case I Pseudo-Code

Examine the “remove all” selection above. One implementation follows.

...Event: shipping prep completed for load item entity k. $TNOW = t_j$

QUEUE, theDockQueue: Detach;

...Event: truck arrival and prep completion at the shipping dock. $TNOW = t_n$

SETUP/SOLVE KnapSack Problem;
REMOVE: All entities flagged to Truck queue;

...Event: start truck loading. $TNOW = t_n$

DELAY: Truck Loading Time;

...Event: closeout truck for exiting process: $TNOW = t_m$
Truck exits to destination. $TNOW = t_{exit}$

Here, only loads readied before the start of loading, $t_j <= t_n$, are considered. The total truck loading time is $t_m - t_n$. The truck load “value” is assured optimal.

5.3 Case II Pseudo-Code

Case II from above, is the “remove some” selection. The following pseudo-code implements a load loop to load one-at-a-time from theDockQueue, reevaluating the what to load next each load cycle.

... Event: shipping prep completed for load item entity k. $TNOW = t_j$

QUEUE, theDockQueue: Detach;

... Event: truck arrival and prep completion at the shipping dock. $TNOW = t_n$

While Truck Not Full
SETUP/SOLVE KnapSack Problem;
REMOVE: First entity flagged;
DELAY: Truck Loading Time per Cycle;
EndWhile;

... Event: closeout truck for exiting process. $TNOW = t_m$
... Event: Truck exits to destination. $TNOW = t_{exit}$

In this second case, all loads readied before the last item to top off the truck is selected for loading, $t_m - t_i$ where t_i is the unit load cycle time of the last entity loaded onto the truck are considered.

6 RESULTS

Three basic examples were modeled. Examples one and two use simple ranked queues for load ordering and selection: FIFO (first in first out) and HVF[Value] (high value first based on the Value attribute.) Loads are picked from queue rank one position until the remaining truck capacity is less than the capacity contribution (attribute Weight) of the first entity remaining in theDockQueue.

The third example also uses a FIFO queue ranking but only loads the entities flagged by the Knapsack solution. That is, if n load item entities are loaded onto an awaiting truck, in case one and two it is the first n items that get loaded. But in the third example this is not guarantied to be true.

Deviations, 1a and 2a, extend the basic ordered queue logic of examples 1 and 2 respectively by not closing out the truck when the next potential load item (first in theDockQueue) is too “big” to fit onto the truck. Rather these two examples continue to search the remaining queued loads for the first that will fit, regardless of its load queue rank, and loads it. This search is continued until no more loads will fit onto

the truck (i.e. topping off the truck, so to speak.) These deviations are guaranteed to be as good or better than the base examples at little additional cost in modeling.

In all examples, the truck capacity is fixed at 40,000 pounds. The experimental design included creating N potential load items with their Weight attribute and Value attribute randomly assigned from the triangular distribution,

TRIA(100,800,1000) (lbs.)

The profit per truck is given by,

SUM($Value_i$) for i = 1....n

where n is the number of entities loaded onto the truck, n < N. In all three examples, N =120. Each example was exercised for 1000 replications of a single truck loading. The average profit is calculated over the 1000 samples for each example, and summarized in Table 1.

Not surprisingly, the contest is between examples 2a and 3. Example 2a uses a value ordered queue and tops off the truck with anything that will fit. Example 3 makes use of our truck loading optimization algorithm. Several things are evident:

1) On the average, the value of trucks loaded optimally is higher than those loaded via the ordered queue and then topped off.

2) The optimal truck loaded value is always equal-to or better-than the modified ordered queue approach.

3) As shown in Figures 5 and 6, the ARENA model is easier to specify for example 3 than it is for example 2a.

The results are better and the task of model development is made easier; what else do we need to consider? The answer is resources. Here we need to consider two resources in particular: computer memory (RAM) and run time.

Memory is cheap! That is to say, that the potential value of the modeling activity makes the cost of memory insignificant, although memory is truly cheap today. Simple benchmarking of the Knapsack algorithm indicates that the additional memory needs are very small compared to a typical ARENA model's memory requirements anyway.

So run speed is the remaining issue to consider. To quantify this several replications of 5000 samples were run and the SIMAN Report run duration was compared. (Not real scientific, but a benchmark none the less.) Results: example 2a ran about 3.23 minutes and example 3 ran about 3.52 minutes. This implies a run length cost of about 9%. Remember that these example were only testing the truck loading procedure. That is, as stated earlier, the overall model from which our examples were extracted is quite large in comparison. If we use a factor of 10x than we might expect that model to show about a 1% runtime increase due to the use of an optimal solution instead of a close answer. Truly a small price to pay.

Easy to use, quicker to model and debug, guaranteed optimal vs. close-enough, and cheap to use. So where's the choice?

Table 1: Confidence Interval Summary

	Average Loaded Value	0.95 C.I. Half-Width	Minimum	Maximum
1	39499	428	35495	45693
1a	40076	420	35495	46514
2	49055	356	45046	53472
2a	49654	370	45645	54112
3	52712	381	48318	57205

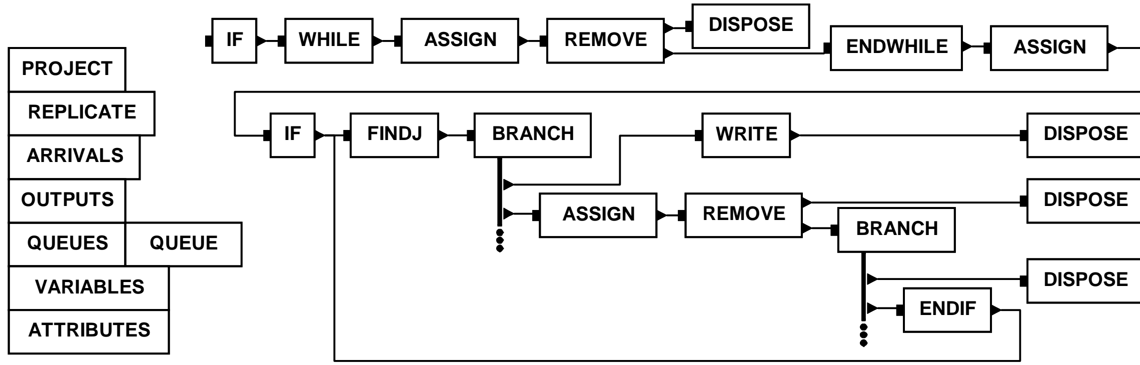

Figure 5: ARENA Model Window for Example 2a

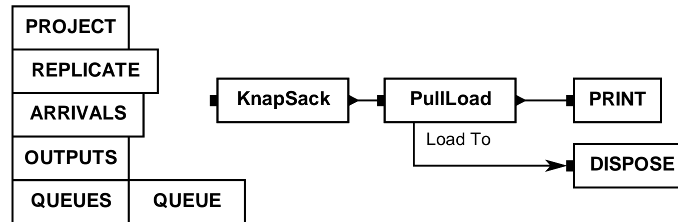

Figure 6: ARENA Model Window for Example 3

7 CONCLUSION

The recognition that many problems encountered within the systems we would like to simulate fall into families, or problem domains, for which solution methodologies already exist prompted this work. The paper has illustrated an approach for providing the simulation modeler with existing, proven solution methods to classical problems domains. This implementation is summarized in following three steps:

1) Identification of a recurring problem domain and applicable solution algorithm in a programming language like C or FORTRAN and/or pre-compiled in a program library (.lib) or dynamic link library (.dll).

2) Develop a user interface in the ARENA template development environment. This interface must delineate the user interaction for problem specification and solution extraction as well as the ARENA interface to the imbedded solution algorithm.

3) Integrate the compiled or pre-compiled solution code into the ARENA user code development environment to generate an ARENA accessible dynamic link library.

Then simply provide the template and accompanying dynamic link library files to the modeler. These files are placed in the ARENA home directory or the project root directory and are can be pulled in as needed or automatically. The modeler need only select from the template panel the desired module(s), place them in the model window, and fill in the blanks: no modeler required programming.

Systems Modeling has positioned the ARENA discrete simulation software package in a leadership role by providing both the Template development environment and the user code development package and then inviting third party developers to take advantage of

these tools to extend and tailor the ARENA simulation and analysis system. This paper examples one such extension to provide the ARENA user community with a generalized solution method to the classical Knapsack problem.

REFERENCES

Horowitz, E. and S. Sahni. 1974. Computing partitions with applications to the knapsack problem. *Journal of ACM* 21: 277-292.

AUTHOR BIOGRAPHY

GREGORY S. BAKER has been involved with simulation for twenty-five years and has been modeling with Systems Modeling products for ten years. He is a Principal Manufacturing Simulation Specialist for Fluor Daniel Corporation and applies simulation methods to a variety of discrete and batch semi-continuous industries in the general areas of plant improvement and new facilities design. He has developed several PE Templates for his own use and for use by other modelers internally. Templates have included, an advanced Containers (tanks) toolbox for batch/semi-continuous hybrid systems; high-speed and high-volume conveyor toolbox; bulk material handling conveyor toolbox for the mining and foods industries well as templates to seamlessly integrate complex numerical algorithms illustrated in this paper. Most of his solutions are implemented as an ARENA Template and .dll (dynamic linked library) file pair.