Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

COMPARING SKILL-BASED ROUTING CALL CENTER SIMULATIONS USING C PROGRAMMING AND ARENA MODELS

Rodney B. Wallace

IBM
7907 Woodbury Drive
Silver Spring, MD 20910, U.S.A.

Robert M. Saltzman

Department of Decision Sciences
College of Business
San Francisco State University
San Francisco, CA 94132, U.S.A.

ABSTRACT

This paper describes the modeling of a skill-based routing call center using two distinct simulation programming methods: the C language and the Arena software package. After reviewing the features of this type of call center, we describe the salient components of each method in modeling the call center. The paper concludes with a comparison of the pros and cons of using each simulation programming approach in this context.

1 INTRODUCTION

Advances in automatic call distributors (ACDs) have made it possible to have skill-based routing (SBR) which is the protocol for online routing of incoming calls to the appropriate agent. Simulation-based analysis of skill-based routing call centers is expected to be the predominant tool of choice since even for relatively simple systems, available analytical solutions are rather restricted. We have developed a $M_n/M_n/C/K/NPRP$ SBR call center simulation program using two different programming methods: the C language and the Arena software package. This paper documents our experience using these two different methods.

We first describe the skill-based routing call center under study. Next, we review the major components of each simulation program. This is followed by comparing the outputs of the two methods using three numerical examples. Finally, a traditional pros and cons comparison is made between the two simulation programming methods.

2 THE CALL CENTER MODEL

We consider a $M_n/M_n/C/K/NPRP$ call center or multi-server queueing system with C total agents, n different call types and telephone trunkline capacity $C+K$. Calls have types $1,\ldots,n$ and are handled in a non-preemptive priority (NPRP) order. The priorities are associated with the skill levels of the agents. Agents have skills at various skill levels (primary, secondary, tertiary and so on). The number of different skill levels is equal to the number of call types n. Skill level 1 represents a primary skill. Skill level 2 represents a secondary skill level, and so on. Each agent has one and only one primary skill. Agents may have up to $n-1$ secondary skills at unique skill levels.

Agents with the same primary skill j make up the work group j. The number of agents in work group j is denoted by C_j, where $1 \le j \le m$ and $m \le n$. The agent skills are given and are represented by a $C \times n$ agent-skill matrix A. Each entry of the agent-skill matrix A is

$$a_{ki} = \begin{cases} q & \text{when agent } k \text{ supports call type } q \\ & \text{at skill level } i, \\ 0 & \text{otherwise,} \end{cases}$$

where $k = 1,\ldots,C$, $1 \le q \le n$, and $1 \le i \le n$. Thus, the rows of the agent-skill matrix represent the unique agent identification (ID), the columns represent the skill level (column 1 indicates primary skill, column 2 secondary, and so on), and the entry a_{ki} indicates the call type supported. The agent skill matrix is one of the major components that distinguishes our SBR call center.

Callers or customers arrive to the call center in accordance to a Poisson process with rate λ. These callers then, independently of one another, select the type of service desired with probability p_i, where i indicates the type of service requested or simply the customer's type, $i = 1,\ldots,n$. Thus, the arrival process for each call type is characterized by a Poisson process with rate $\lambda_i \equiv \lambda p_i$ for $i = 1,\ldots,n$, where $\lambda = \sum_{j=1}^{n} \lambda_j$. The service times to process calls depend only on the call type and are independent and identically distributed (IID) with exponential service times $1/\mu_i$, where i represents the call type for $i = 1,\ldots,n$.

In the $C+K$ trunklines capacity, the parameter K is the number of waiting spaces or buffers to hold waiting callers. If an arriving caller finds that there are already $C+K$ customers present, then the caller is blocked and is lost to the system. Callers arriving to each work group are handled

or processed using a *first-come first-serve* (FCFS) service discipline among qualified agents. Customer abandonments, retrials, and jockeying are not permitted.

There are two fundamental problems we must address in our SBR call center model construction. They are:

1. What to do When an Arrival Occurs
2. What to do When an Agent Becomes Free.

We address each of theses in the following subsections.

2.1 What to do When an Arrival Occurs

To address the issue of what to do when an arrival occurs, we consider the call routing strategy most commonly used by call center managers, the *longest idle agent routing* policy. The longest-idle-agent routing (LIAR) policy sends calls to the agents that have been waiting the longest for a call since the completion of their last job (i.e., idle the longest). To adjust for priorities, the LIAR policy that we adopt sends calls to the agents that have been waiting the longest (or idle the longest) and have the highest skill-level to handle the call.

2.2 What to do When an Agent Becomes Free

When agents become free, if there are no customers in the n queues then the agents go idle; otherwise, the first customer in the queue that the agents can support at their highest skill level is taken into service. More precisely, when agents become free, the first customer in the queue in which the agents have a primary skill level to support the call is taken into service. We will refer to this queue as the agents primary skill queue. If there are no customers in the agents primary skill queue, the first customer in the agents secondary skill queue is taken (provided the agents have a secondary skill or $a_{k2} \neq 0$). The process is continued in this manner until either a customer is found that the agents can support or all skill levels have been exhausted. If the agents cannot find a waiting customer that they can support then the agents go idle. Customers waiting in the queue that are not supported by freed or idled agents continue to wait until agents that can support their call type become available.

2.3 Key Performance Measures

From a call center manager's perspective, we consider the following seven (7) key performance measures as listed in Table 1.

In the table, we have several random variables of interest. The random variable Q has steady-state distribution for the queue length process of the system. The random variable D has steady-state distribution for the delay time of a caller admitted to the system and random variable D_i has steady-state distribution for the delay time of a caller admitted to the system that requests type i service. The number of callers in the system includes the number of callers in service plus the number of callers in queue or waiting. In the table, the indices i and j indicate the call type or service request ($i = 1, \ldots, n$) and the work group ID ($j = 1, \ldots, m$).

Table 1: Key Call Center Performance Measures.

Performance Measure	Description
1. $P(Q = C + K) = \epsilon$	Probability of blocking
2. $E[D \mid Q < C + K] = W$	Average speed to answer (ASA) given system entry
3. $E[D_i \mid Q < C + K] = W_i$	Average speed to answer call type i given system entry
4. $P(D \leq \tau \mid Q < C + K) = 1 - \delta$	Percent of calls that are answered within τ minutes given system entry
5. $P(D_i \leq \tau \mid Q < C + K) = 1 - \delta_i$	Percent of calls of type i that are answered within τ minutes given system entry
6. υ	Agent utilization
7. υ_j	jth work group utilization

In Table 1, the first performance metric, the probability that an arriving caller is blocked, is a measure of the call center's availability and is sometimes part of the service level agreements (SLAs). The second parameter $E[D \mid Q < C + K]$ and the fourth $P(D \leq \tau \mid Q < C + K)$ are speed-to-answer performance measures and are typically part of the service levels as well. These two aggregate quantities are conditioned given admission or entry into the system. Usually, one of the two and not both speed-to-answer metrics is part of the SLA. Average speed to answer (ASA) is the call center term reserved for $E[D \mid Q < C + K]$.

The last two measures of performance deal specifically with tracking agent's utilization. The *average utilization* for an agent is the percent of time that he/she is busy processing calls or one minus the fraction of time he/she is idle.

3 THE C PROGRAM SIMULATION

We develop a discrete-event simulation to model the $M_n/M_n/C/K/NPRP$ skill-based routing call center model described in the previous section using the C programming language running on an IBM ThinkPad laptop with a Pentium M 1.6 GHz processor and 768 MB (megabytes) of memory. Mazzuchi and Wallace (2004) cite many of the references we used to construct a very robust multi-server simulation model. In this section, we provide a brief sum-

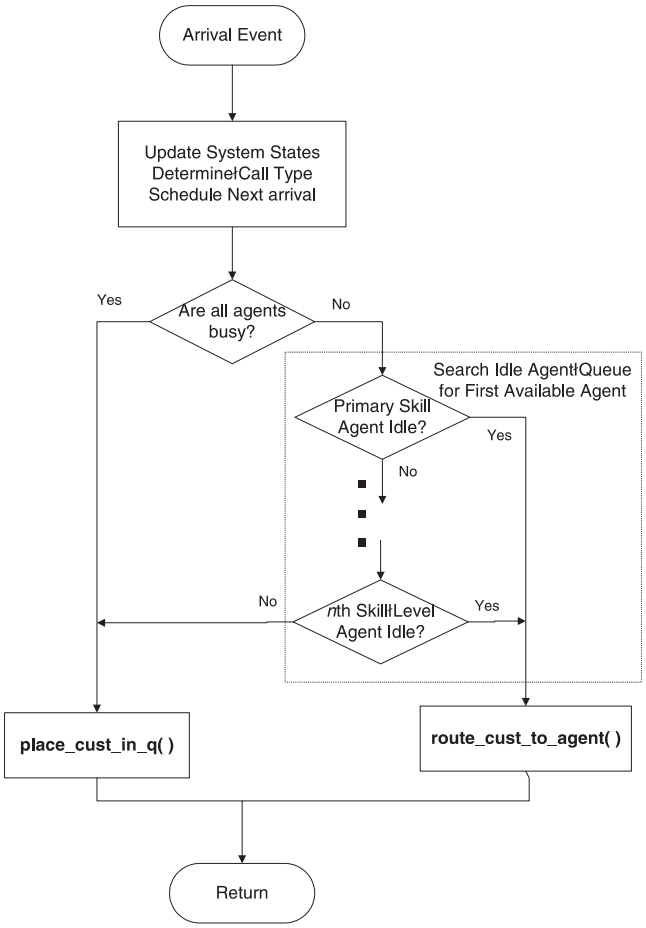

Figure 1: Flowchart for arrival routine, SBR Call Center Model.

mary of the components of our C program while focusing on the unique aspects of skill-based routing.

We use the *next-event time advance* approach for advancing the simulation clock. An event is either an arrival to the call center or a departure from an agent. Thus, our event list consists of $C + 1$ items (i.e., arrival or departure from agent k, where $k = 1, \cdots, C$). Our C program includes all the basic components or subroutines found in discrete-event simulations: system state, simulation clock, statistical counters, initialization, timing routine, event routines, library routines, report generation, and a main program.

We will discuss the logic of two key routines in our C program: arrival and departure. Figure 1 shows a flowchart of the logic we use to show what happens when an arrival occurs. The logic is somewhat standard for arrival routine that supports multiple call types. The exception or uniqueness is how calls are routed to accommodate the longest idle agent routing (LIAR) policy.

To simulate the LIAR policy, we use the following technique. We construct an idle agent queue. When agents become free or idle, their IDs are placed at the end of the idle agent queue. Thus, the agents that have been idle the longest are at the top of the queue. As shown in Figure 1, a first scan of the idle agent queue seeks to find the first agent that has a primary skill to support the call. Agent skill profile information is taken from the agent skill matrix using the agent ID stored in the idle agent queue. If no agents are found then a second scan of the idle agent queue seeks to find the first agent that has a secondary skill to support the call and so on.

The first agent ID that is identified in this sequential search is given the arriving call via the route_cust_to_agent() subroutine. If no agents in the idle agent queue can support the arriving call then the caller is placed in queue, provided there are less than $C + K$ callers in the system.

Next, we will discuss the departure routine that describes what happens when agent k becomes free, $k = 1, \cdots, C$. Figure 2 shows the logic used for departure routine. When agent k completes a call (i.e., a departure occurs), agent k seeks to retrieve the next caller he/she can support that has been waiting the longest in the following manner. Using the agent skill matrix, agent k primary skilled-queue is checked first for a waiting caller. If primary-skilled queue is empty then agent k secondary skilled-queue is checked and so on. If waiting callers are found that can be supported, then agent k gets the waiting caller that is first in queue; otherwise, agent k goes idle with the execution of the make_server_idle() routine. The idle agent ID is stored at the end of the idle agent queue.

We briefly review how the performance statistics are calculated. Performance reporting and data analysis accounts for a majority of the C program. For each call, we record the time the call entered the system (or time the call was blocked), the ID of the agent who handled the call, the time the call entered service and the time the call completed service and exited the system.

From stochastic output processes of the form $\{X_n : n \geq 1\}$, we estimate the steady-state performance measures shown in Table 1 using one long simulation run. During the long run, we delete an initial portion of the observations in order to account for the effects due to initial bias. We choose the initial portion to delete large enough so the system is nearly in steady-state. The remaining observations are divided into a fixed number of non-overlapping batches of equal length. The length of the batches is sufficiently large such that correlation between batches becomes negligible and the batch mean approximately follows a normal distribution. We use sample batch means to estimate variances and construct confidence intervals.

Finally, we have used a number of techniques to validate and verify our SBR call center simulator coded in C. This list of techniques include program logic review by call center

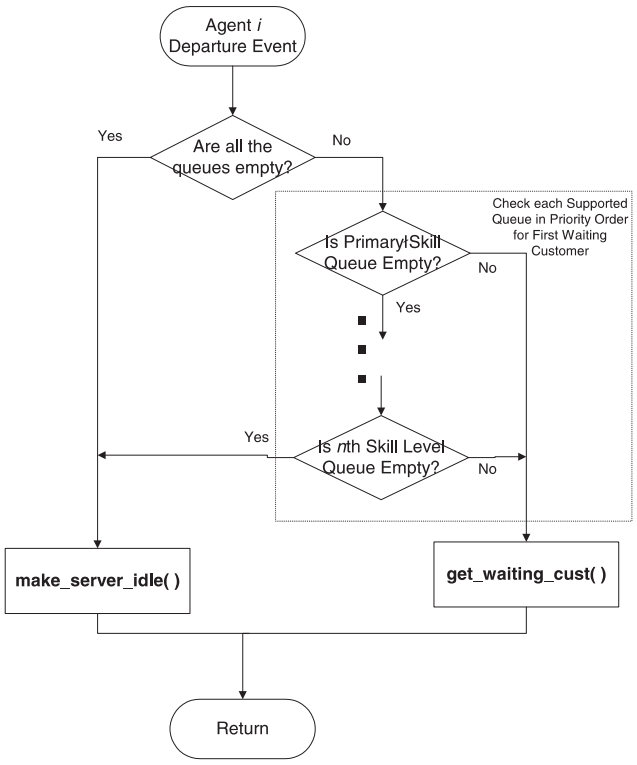

Figure 2: Flowchart for departure routine, SBR Call Center Model.

managers and experts, sensitivity testing, stress testing, trace analysis, and output comparison against known models. See Mazzuchi and Wallace (2004) and Wallace (2004) for details.

4 THE ARENA SIMULATION

We also developed an equivalent call center model using the animated simulation package Arena 7.01 (Kelton, Sadowski and Sturrock, 2004). The model was constructed and run on a Dell Inspiron 600m laptop with a Pentium M 1.4 GHz processor and 512 MB of memory. This section recaps the model's structure, most of which can be seen in Figures 3 and 4 below.

In the top box of modules of Figure 3 the idle agent queue is initially populated with an entity for each agent. These agent entities are shown as an ID number indicating which agent subgroup they belong to, where a subgroup is defined to be a set of agents with completely identical skills. Later on, as calls are allocated to agents who can handle them, ID numbers are removed from the idle agent queue; when an agent becomes idle, its ID number is put back into the queue.

The middle box of modules in Figure 3 begins by creating calls and assigning them key attributes such as their call type, picture and service time. An arriving call is blocked if the total number of calls already waiting in queue equals the maximum queue capacity. Blocked calls are recorded and disposed of. Other calls check to see if there are any idle agents available. If all agents are currently busy (i.e., the idle agent queue is empty), then the call is sent to wait in a queue specific to its call type. If there's at least one idle agent, then the call searches the idle agent queue for an agent possessing the skills required to handle its type. If such an agent is available, the agent's ID number is assigned to the call's "ServedBy" attribute and the call is then sent to the agent's station for service. Meanwhile, this agent's ID number is removed from the idle agent queue. If no appropriately-skilled agent is found, the call is sent to wait in a queue.

The bottom box of modules in Figure 3 handles completed calls. First, call waiting time and service level statistics are recorded. Then the call queues are searched to see if there are any waiting calls that can be served by the agent who has just finished serving the completed call. If so, the call is removed from its queue and sent to the appropriate agent's station for service. If no such call is waiting, the agent becomes idle and its ID number is placed in the idle agent queue.

Figure 4 shows the main call and agent animation, as well as the logic needed for the queuing and processing of calls. On the left side of the figure are the queues (one per call type) where calls wait for an available agent. Eventually, each call reaches the head of its queue, is sent to one of the agent subgroup stations for service (on the right), and returns to the logic for completed calls (bottom of Figure 3). During model execution calls of different types are shown in different colors, and can be seen waiting in various lines and getting served by agents.

Important data modules (not shown in the figures) include the Variable, Expression and Advanced Set modules. Key variables are: NGroups, the number of subgroups of agents who possess identical skills sets; NAgentGroupSize, a vector of the number of agents within each subgroup; AMatrix, the agent skill matrix containing just 1 row per subgroup; MaxSkill, a vector of the number of skills per agent subgroup; NAgentsTotal, the total number of agents; MaxQLength, the maximum number of calls that may be put on hold; and SLTarget, the desired service level target time. The Expression data module defines an expression for the total number of calls currently in queue, as well as an indexed expression holding the service time distributions by call type. Finally, the Advanced Set module defines collections of similar objects, namely, QueueSet, which holds the names of all of the call queues, QueueStationSet, which contains the names of the station modules preceding each queue, and AgentStationSet, which holds the names of the stations preceding the agent subgroup process modules. These sets greatly simplify the logic used in Figure 3.

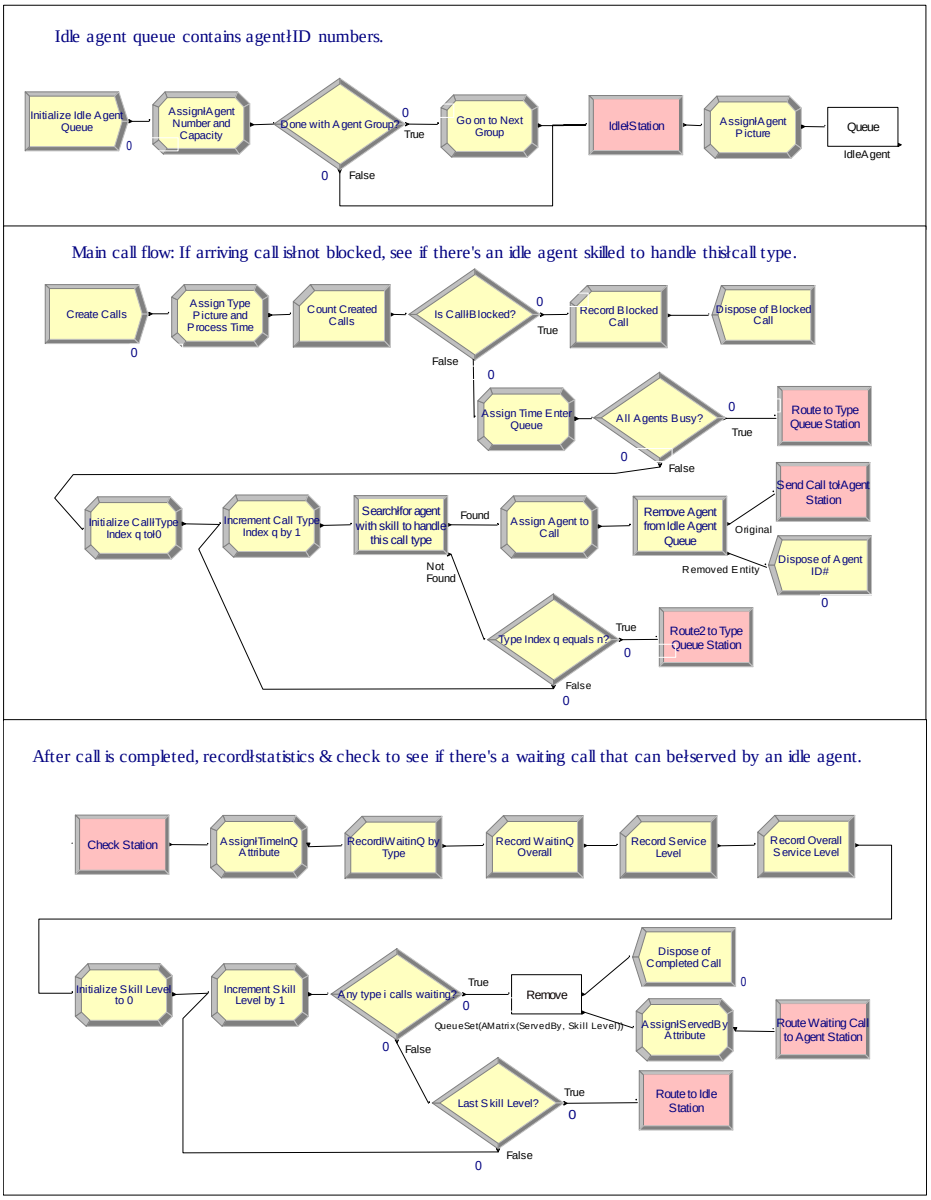

Figure 3: Main logical flow for the skill-based routing model in Arena 7.01.

As with the model built in C, the Arena model also collects output via the batch means method, after deleting output from a long warm up period. Confidence intervals are automatically reported by Arena for all performance measures specified in the Statistic data module except those based upon counters, such as the blocking percentage, which is the ratio of the number of calls blocked to the number created.

5 SIMULATION RUN COMPARISONS

In this section, we compare the simulation output from the C and Arena programs using three numerical examples. Table 2 shows the 95% confidence intervals for each of the key call center performance measures presented in Table 1. In each simulation run, we generate approximately 800,000 observations (simulated calls) in which the first 20% of the observations are discarded to account for the initial bias.

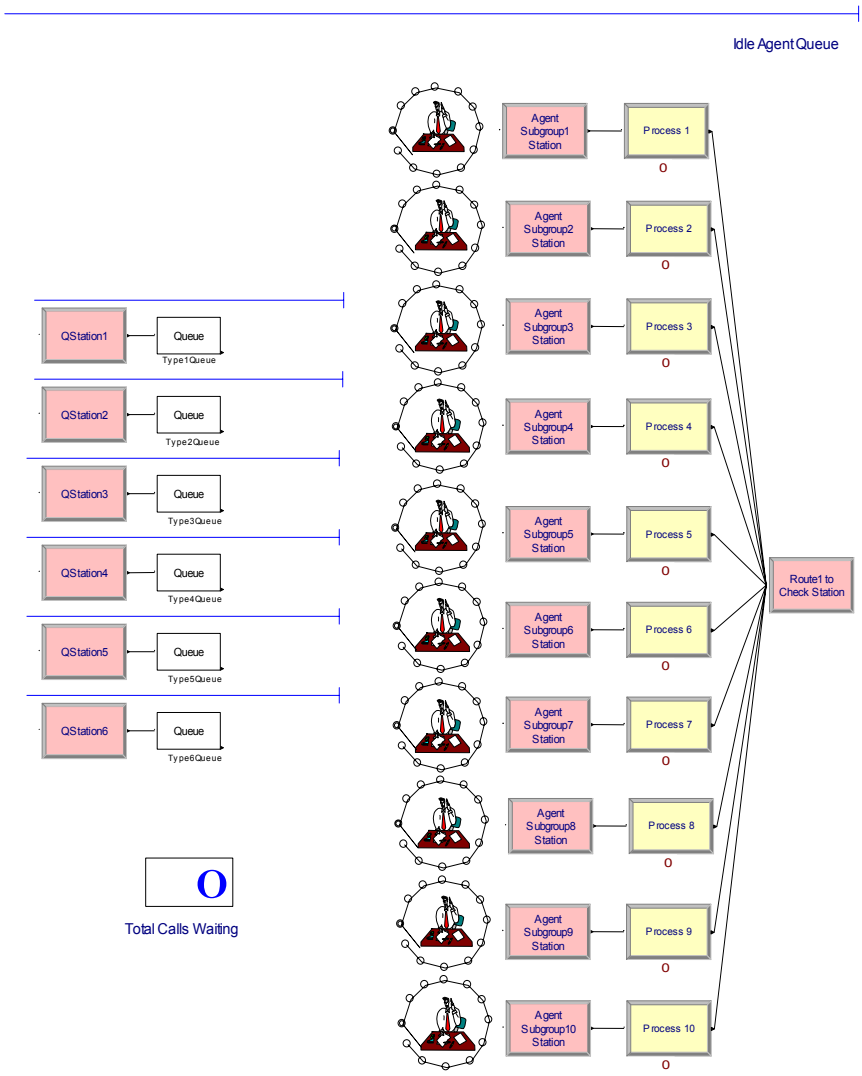

Figure 4: Part of the main animation section of the Arena model.

The outputs from the C program and Arena are remarkably close.

The first example is the Stanford and Grassmann (1998) Bilingual Call Center model. They modeled the continuous Markov chain that describe the Bilingual call center system using a matrix-geometric model in order to estimate average delay for each call type. Their estimates for ASA_1 and ASA_2 are 1.43 and 9.39 seconds. Stanford and Grassmann (1998) occupancy or agent utilization is calculated using

$$\upsilon = \frac{\lambda h}{C_1 + C_2} = 80\%$$

where $h = 1/\mu$ is the mean service time, $\lambda = \lambda_1 + \lambda_2$, C_1 is the number of unilingual agents in work group 1 and C_2 is the number of bilingual agents in work group

Table 2: Comparing the C and Arena SBR Call Center Simulations 95% Confidence Results.

	1. $M_2/M_2/20/\infty/NPRP$ Stanford and Grassmann Bilingual Call Center λ_1 = 0.384 jobs/sec, λ_2 = 0.256 jobs/sec, $1/\mu_1 = 1/\mu_2$ = 25 sec, (C_1, C_2) = (12, 8), τ = 10 seconds		**2.** $M_6/M_6/90/21/NPRP$ λ_i = 1.375 calls/min, $i = 1, \cdots, 6$ $1/\mu_1 = 1/\mu_2$ = 8 min, $1/\mu_3 = 1/\mu_4$ = 10 min, $1/\mu_5 = 1/\mu_6$ = 12 min, $C_1 = \cdots = C_6$ = 15, τ = 30 seconds		**3.** $M_6/M_6/200/50/NPRP$ λ_i = 3.25 calls/min, $i = 1, \cdots, 6$ $1/\mu_1 = 1/\mu_2$ = 8 min, $1/\mu_3 = 1/\mu_4$ = 10 min, $1/\mu_5 = 1/\mu_6$ = 12 min, $C_1 = 27$, $C_2 = C_3 = 30$, $C_4 = 33$, $C_5 = C_6 = 40$, τ = 30 seconds	
Performance Measure	**C**	**Arena**	**C**	**Arena**	**C**	**Arena**
1. Blocking (%)	-	-	[0.47, 0.65]	0.556	[0.40, 0.63]	0.515
2. ASA (min)	[4.41, 5.05]*	[4.34, 4.86]*	[0.35, 0.40]	[0.35, 0.39]	[0.49, 0.60]	[0.54, 0.59]
3. ASA_1	[1.38, 1.60]*	[1.25, 1.45]*	[0.47, 0.54]	[0.46, 0.53]	[0.54, 0.70]	[0.60, 0.66]
3. ASA_2	[8.90, 10.29]*	[8.84, 10.11]*	[0.45, 0.52]	[0.46, 0.53]	[0.44, 0.54]	[0.48, 0.53]
3. ASA_3	-	-	[0.32, 0.37]	[0.34, 0.38]	[0.63, 0.80]	[0.72, 0.80]
3. ASA_4	-	-	[0.33, 0.38]	[0.32, 0.37]	[0.47, 0.60]	[0.54, 0.60]
3. ASA_5	-	-	[0.25, 0.29]	[0.25, 0.29]	[0.40, 0.51]	[0.45, 0.50]
3. ASA_6	-	-	[0.25, 0.29]	[0.25, 0.28]	[0.40, 0.52]	[0.44, 0.49]
4. P(Delay $\leq \tau$|entry) (%)	[83.5, 84.9]	[84.1, 85.3]	[78.5, 80.5]	[78.9, 80.7]	[68.1, 73.3]	[68.5, 70.7]
5. P($Delay_1 \leq \tau$|entry)	[94.5, 95.3]	[95.0, 95.8]	[73.2, 75.8]	[73.7, 76.1]	[65.4, 70.8]	[66.3, 68.8]
5. P($Delay_2 \leq \tau$|entry)	[67.0, 69.6]	[67.2, 69.8]	[73.6, 76.0]	[73.8, 76.2]	[68.8, 73.6]	[69.1, 71.3]
5. P($Delay_3 \leq \tau$|entry)	-	-	[78.8, 81.0]	[79.2, 81.0]	[63.6, 69.4]	[63.7, 66.3]
5. P($Delay_4 \leq \tau$|entry)	-	-	[79.2, 81.4]	[79.5, 81.3]	[68.1, 73.8]	[68.2, 70.6]
5. P($Delay_5 \leq \tau$|entry)	-	-	[82.8, 84.8]	[83.4, 85.2]	[71.2, 76.5]	[71.3, 73.6]
5. P($Delay_6 \leq \tau$|entry)	-	-	[82.6, 84.8]	[83.3, 84.9]	[70.9, 76.3]	[72.0, 74.2]
6. Avg Agent Util (%)	[79.6, 80.2]	[79.5, 80.3]	[90.8, 91.4]	[90.5, 91.3]	[96.6, 97.2]	[96.8, 97.0]
7. Work $Group_1$ Util (%)	[75.2, 75.8]	[75.2, 76.0]	[90.2, 91.0]	[90.1, 90.9]	[96.2, 97.0]	[96.5, 96.8]
7. Work $Group_2$ Util	[86.0, 86.9]	[85.8, 86.8]	[90.3, 91.1]	[90.2, 90.8]	[95.9, 96.7]	[96.2, 96.5]
7. Work $Group_3$ Util	-	-	[90.8, 91.4]	[90.5, 91.3]	[96.8, 97.6]	[97.1, 97.3]
7. Work $Group_4$ Util	-	-	[90.6, 91.4]	[90.5, 91.3]	[96.8, 97.6]	[96.9, 97.1]
7. Work $Group_5$ Util	-	-	[91.2, 91.8]	[90.9, 91.7]	[96.8, 97.4]	[97.0, 97.2]
7. Work $Group_6$ Util	-	-	[91.1, 91.9]	[91.1, 91.7]	[96.8, 97.4]	[97.0, 97.2]

* - Average Speed to Answer (ASA) is measured in seconds.

2. The 20×2 agent skill matrix used in this example has the following structure. The rows of the agent skill matrix representing agents in the unilingual work group have a 1 in the first column and a 0 in the second column. The rows representing the agents in the bilingual work group have a 2 in the first column and a 1 in the second column. Table 2 shows that the estimates all fall within both simulations' 95% confidence intervals.

The second example, a $M_6/M_6/90/21/NPRP$ SBR call center model, is simulation run taken from Wallace and Whitt (2004a). In this example, agents have two skills each. The agent skill matrix, $A^{(2)}_{90\times6}$, used in this simulation run is provided in Appendix A of Wallace and Whitt (2004b). In the last example, we configured the simulators to model a relatively large SBR call center with 200 agents and 250 trunklines. In this example, each agent has three skills. The agent skill matrix, $A^{(3)}_{200\times6}$, used in this simulation run is provided in the supplemental paper by Wallace and Saltzman (2005).

In summary, the results in the table show that the two simulations are clearly modeling the behavior of the same system. We have found the relative difference in the actual estimators for the blocking, mean delay, target delay, and utilization to be less than 0.8%, 8.0%, 2.0%, and 0.2%, respectively. We can improve these results by running the simulation much longer than 800,000 observations.

6 COMPARING THE TWO DIFFERENT SIMULATION METHODS

Table 3 summarizes the comparison of the C program and Arena. Both programs were developed using Pentium M-

based laptops. The C program environment had 50% more memory. As shown in the table, Arena runs much slower than the C program (less than 3 seconds), even with animation disabled; however, it still executes in a reasonable amount of time (less than 3 minutes).

Table 3: Summary of Comparison: C Program Vs. Arena Standard Edition

Evaluation Criteria	**C Program**	**Arena Model**
1. Laptop Specs.	IBM ThinkPad Pentium M 1.6 GHz CPU 768MB RAM	Dell Inspiron Pentium M 1.4 GHz CPU 512 MB RAM
2. System Run Times (a) 20-agent run (b) 90-agent run (c) 200-agent run	 2 seconds 2 seconds 3 seconds	 21 seconds 30 seconds 165 seconds
3. Skills Required	Advanced	Advanced
4. Ease of Construction or Implementation	Tedious statistical analysis	Built-in statistics Challenge to implement LIAR policy
5. Very Flexible?	Yes	Yes
6. Very Scalable?	Yes	Yes, but adding new agent subgroup is nontrivial
7. Costs (a) Tool (b) Modeler's Time	 Cheap Few months	 Very expensive Few weeks
8. Code Maintenance	Challenging	Commercially supported
9. Saleability	Challenging	Animation and built-in flow diagram are a plus

In terms of the skills required of the modeler, certainly advanced C programming skills are required to develop a fully functional discrete-event simulation of a complex multi-server queueing system from scratch. Advanced Arena skills are required primarily because agents must be represented both as resources and as entities with ID numbers who join and depart from an idle agent queue. Other challenges involve correctly using Search, Remove and Route logic modules, and the Advanced Set data module.

Even though the second author had built a number of call center models before in Arena, e.g., Saltzman and Mehrotra [2001, 2004], he still found it challenging to construct parts of this skills-based routing model, especially the LIAR policy. On the other hand, it was relatively easy to animate the Arena program and obtain most of the desired performance measures. Arena has built-in statistical analysis features; however, in the C program, all statistics are constructed from scratch which is a very tedious task. In addition, many counters are collected and reported in a trace file in order to assist in the debugging process.

As for flexibility, we found both models were able to accommodate all the modeling objectives with no limitations. As for scalability, the C program requires no changes to the code in order to model larger systems. To accommodate large systems in the C program, only input values are changed (e.g., the number of agents C, the number of trunklines $C + K$, the agent skill matrix). In Arena, some tasks are easy to do, e.g., scaling up the size of the model by increasing the number of agents within any subgroup. Other tasks are a bit more work, e.g., adding a new agent subgroup because it requires increasing elements of sets and altering the animation layout.

The Arena program was constructed with the very expensive Standard Edition of Arena (thousands of dollars). However, model development time was relatively quick (a few weeks), and the model can be executed in "Runtime Mode" using the inexpensive academic version of Arena. In general, C environments are very cheap (range from free to $100). On the other hand, the model development time is much longer (few months) since the environment is completely developed from scratch. More than half the C development time and code was spent in creating debug code (e.g., trace files) and statistical analysis.

As for code maintenance, historically, Arena's vendor (Rockwell Software) has kept its software "backward compatible" so that newer versions of Arena can convert and run models developed in earlier versions. This bodes well for future use of the model. Rockwell Software actively supports their product and provides training seminars to product users. The C code is fully documented in Wallace (2004); however, advanced C programming and call center modeling skills are required to maintain the existing code or make future updates.

Finally, the last evaluation criteria is *saleability*. Saleability deals with the ability of the modeler to sell or communicate the results to key call center decision makers. These key decision makers are typically not experts in simulation and want to understand the big picture or bottom line. Arena has the edge on the C code as it relates to saleability. Arena's animation capability is definitely helpful in selling the model's validity and usefulness to management (see Figure 4). Also, the Arena program's flow diagram appearance is fairly transparent to managers (see Figure 3).

7 SUMMARY

We have demonstrated the effectiveness and have shown the accuracy of constructing a skill-based routing call center discrete-event simulation using the C programming language and the Arena simulation package. We have compared the two different programming methods against a myriad traditional criteria. Selecting the best method for your needs still depends on the model's intended use and audience.

ACKNOWLEDGMENTS

The authors would like to thank both Damita Elliott and Timothy Melvin for their clever programming suggestions and techniques.

REFERENCES

Kelton, W. D., R. P. Sadowski and D. T. Sturrock. 2004. *Simulation with Arena*, 3rd Edition, Boston: McGraw-Hill.

Mazzuchi, T. A., and R. B. Wallace. 2004. Analyzing Skill-Based Routing Call Centers using Discrete-Event Simulation and Design Experiment. In *Proceedings of the 2004 Winter Simulation Conference*, eds. R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, 1812–1820.

Saltzman, R. and V. Mehrotra. 2001. A Call Center Uses Simulation to Drive Strategic Change. *Interfaces*, 31(3): 87-101.

Saltzman, R. and V. Mehrotra. 2004. A Manager-Friendly Platform for Simulation Modeling and Analysis of Call Center Queueing Systems. In *Proceedings of the 2004 Winter Simulation Conference*, eds. R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, 466–473.

Stanford, D. and W. K. Grassmann. 1998. Bilingual server call centers. *Analysis of Communication Networks: call centers, traffic and performance*, eds. D. McDonald and S. R. E. Turner, 31–47, Providence: *American Mathematics Society*.

Wallace, R. B. 2004. Performance Modeling and Design of Call Centers with Skill-Based Routing, D.Sc. Dissertation, The George Washington University.

Wallace, R. B. and R. M. Saltzman. 2005. Comparing Skill-Based Routing Call Center Simulations Using C Programming and Arena Models: supplementary material. Available online via <http://userwww.sfsu.edu/ saltzman/skillmatrix200.xls> [Accessed April 9, 2005].

Wallace. R. B. and W. Whitt. 2004a. A Staffing Algorithm for Call Centers with Skill-Based Routing. To appear in *Manufacturing and Service Operations Management*.

Wallace. R. B. and W. Whitt. 2004b. A Staffing Algorithm for Call Centers with Skill-Based Routing: supplementary material. Available online via <http://www.columbia.edu/ ww2040/poolingMSOMsupRev.pdf> [Accessed April 7, 2005].

AUTHOR BIOGRAPHIES

RODNEY B. WALLACE is a Technical Solutions Manager at IBM. Rodney has a B.S. and M.S. in Mathematics from the University of Georgia and Southern University. He has a D.Sc. in Operations Research from The George Washington University. His research interests include computer systems performance analysis, queueing theory and simulation. He is a member of INFORMS and his e-mail address is <rodney.wallace@us.ibm.com>.

ROBERT M. SALTZMAN is a Professor in the Decision Sciences Department at San Francisco State University where he teaches courses in Simulation, Operations Management, and Statistics. He received his Ph.D. in Operations Research from Stanford University, and his B.S. degree in Applied Mathematics from Brown University. Rob's main research interests are in decision making via animated simulation and optimization modeling. His email address is <saltzman@sfsu.edu>.