Simulating Patient Flow through an Emergency Department Using Process-Driven Discrete Event Simulation

M. S. Raunak, L. J. Osterweil, A. Wise, L. A. Clarke
University of Massachusetts Amherst
{raunak, ljo, wise, clarke}@cs.umass.edu

P. L. Henneman
Tufts-Baystate Medical Center
philip.henneman@bhs.org

Abstract

This paper suggests an architecture for supporting discrete event simulations that is based upon using executable process definitions and separate components for specifying resources. The paper describes the architecture and indicates how it might be used to suggest efficiency improvements for hospital Emergency Departments. Preliminary results suggest that the proposed architecture provides considerable ease of use and flexibility for specifying a wider range of simulation problems, thus creating the possibility of carrying out a wide range of comparisons of different approaches to ED improvement. Some early comparisons suggest that the simulations are likely to be of value to the medical community and that the simulation architecture offers useful flexibility.

1. Introduction

Healthcare delivery can be very expensive and is often subject to delays that can be costly as well as dangerous to patient health. In our view both the problems of cost and delay might be addressed by devising superior approaches for evaluating the utilization of resources. The resources used in delivering health care include both equipment, such as beds, radiographic imaging equipment, and operating suites and people, such as doctors and nurses. All of these resources are very costly. Accordingly these resources tend to be allocated sparingly in most health care settings. This can lead to a lack of sufficient resources that can be at least partly to blame for delays in delivering health care. Adding resources of only one type (e.g. more doctors) is rarely sufficient to reduce delays, however, as patient care typically requires the use of many types of additional resources (e.g. beds, nurses, and equipment). Increasing resources in an unbalanced way can thus lead to underutilization of some resources and increased expense, but no improvement in service.

Effective approaches to studying how to balance resources in clinical settings thus seems to offer the opportunity to demonstrate how to reduce delays in treating patients, while still maintaining efficient utilization of expensive resources. This paper suggests such an approach, namely the use of discrete event simulation of clinical processes that are defined sufficiently precisely and completely to make clear the ways in which resources are used at every process step.

Our approach emphasizes careful attention to both process details and resource issues. We note that the delivery of health care is not always straightforward, and may proceed in many different ways. Thus the description of a process for delivery of healthcare seems to us to require the use of a powerful process description facility. Describing the resources used in healthcare delivery is very challenging as well, as there are many different types of resources, and there may be a considerable amount of flexibility in how different resources may be used to support the performance of different steps under different circumstances. Thus, for example, medications are typically given by nurses, but under unusual circumstances, doctors may do this. Moreover, the most critically ill patients are typically given medical care first, but in the case of a disaster, the most critically ill patients may be bypassed in order to deliver care to those for whom the care is most likely to be most effective. Thus, an important benefit of our discrete event simulation approach might be enabling the study of the effects on both realism and efficiency that might result from allowing resource substitution flexibility.

There are considerable challenges in dealing with these issues of process and resource specification [11, 13]. Our view, however, is that meeting these challenges can lead to a discrete event simulation facility that could be an effective basis for experimenting with resource mixes and utilization strategies.

This paper describes such an approach using the delivery of Emergency Department (ED) care as an example. The paper indicates very early results and some directions for future research.

2. Prior Related Work

We note that there have been other attempts to use discrete event simulation as a vehicle for exploring ways to increase efficiency and reduce waiting time in a hospital ED. Connelly and Bair [2] present a discrete event simulation model named EDSim developed using Extend, a general purpose commercial simulation tool, to investigate the ability to predict actual patient care times using simulation. The Extend simulation tool [8] is primarily queue based and provides an interactive modeling environment with a compiled modeling language, modL [8], to facilitate building of reusable and hierarchically decomposable components.

The study by Connelly and Bair looked at the effect of two different triage methods on patient service time in the ED. The authors collected real-life patient data from an academic ED to drive the simulations of their modeled ED activities. They modeled such ED activities as physical examination, nursing activity, imaging and laboratory studies, and bedside procedures such as suturing, casting, and intubation. In addition to individual patient care paths, EDSim also considered continually updated job queue prioritization and mid-task preemption capabilities for ED staff. All staff activities were prioritized according to patient acuity. According to the study, this model was able to predict average patient service time within 10% of actual values. For individual patient paths, however, only 28% of individual patient treatment times had an absolute error of less than one hour. The authors suggest that one of the reasons their results did not accurately predict the actual timings of real events was because their simulation did not support making changes in resource levels (specifically staffing levels) at different times of the day. Moreover, it appears that their simulation did not allow for the possibility of specifying complex constraints on resource utilization and task interruption.

Other simulation studies have focused on staff scheduling in hospitals in general and EDs in particular [4]. A couple of studies have been done to analyze alternative nurse scheduling techniques in EDs [3, 9]. A study by Rossetti et. al [14] looked at the use of computer simulation to test alternative ED attending physician-staffing schedules and to analyze the corresponding impacts on patient throughput and resource utilization. A second area of focus in ED simulation studies has been to look at process changes and their impact. McGuire studied [10] the use of simulation to test process improvement alternatives for reducing the length of stay for ED patients. Another study [15] shows the use of ED simulation studies to perform 'what-if' analysis regarding the effect of process change and staff level change on patients' length of stay. A recent study by Hoot et al. [7] has used discrete event simulation in forecasting imminent crowding in the emergency department based on changes in waiting room counts. This simulation took past and present patient-level data as input and produced future patient-level data as output. Many of these studies [4, 15] used Arena [5], a commercial simulation tool, to obtain their results. Arena is an object-based, hierarchical modeling tool that is used in a wide range of applications. Like Extend [8], Arena simulations are driven by queuing models, whereas our simulations are driven by process and resource definitions.

Another area of research that has received attention lately is that of scheduling ED staff under different constraints. Chun describes a *Staff Rostering System* [1] for creating nurse rosters for the Hong Kong Hospital Authority that manages over 40 public hospitals. The system defines different constraints to be satisfied while creating the roster. For example, one constraint assures that an adequate number and mixture of staff are present all the time to maintain an acceptable level of service quality. Other constraints are used to ensure that no staff member is either overworked or underutilized according to their terms of appointment. This type of scheduling is done at the macro level of shift assignment, whereas we are primarily concerned with more micro level task based resource assignment in our work.

All of these simulation studies have taken a factory view of the ED, where patients come in like orders queued on a factory floor, often with fixed priority, and drive the process by requesting resources. Many of these studies were concerned with only one type of resource, i.e. either the attending physician or nurse and focused on only one issue of resource management, such as scheduling. A recent study [6] has identified this issue and proposes a different way of modeling and studying ED processes. The study argues that with only a factory view of the world, low acuity patients will continually be starved from receiving services and many will not receive treatment at all. The study also argues in favor of modeling the skill hierarchy of ED staff, using skill-based request specification, and allowing an ED agent to prioritize its tasks. By modeling changing request priorities and resource mapping based on requested skill sets, this study was able to produce simulation results that align more closely with how resources in a real ED are actually used. This work is closer in philosophy and approach to our own work, in that it places a greater emphasis on modeling resources. Our work incorporates a similar focus on resources and also seems to support more generality in resource substitutability. But our work complements the focus on resources with a focus on process as well. In particular, our approach, which builds on our previous research [12], suggests that closely correlating the specific need for resources with specific detailed process steps should lead to simulations that are more accurate and potentially capable of better prediction of the effect of making changes. In the next section we detail this approach.

3. Our Approach

In our work we use the Little-JIL process definition language to specify processes used in delivering medical care in an ED. We then couple these process definitions with specifications of resources and constraints in order to drive discrete event simulations of how different configurations of process steps, resource mixes, and constraints handled different flows and mixes of patients. This approach features the separation of various key simulation concerns into well defined components that can support considerable flexibility and precise specification detail, as we shall now describe.

3.1. Process Definition

Little-JIL is an executable process definition language with a well-defined semantics. A Little-JIL process definition is a hierarchy of steps, each of which is best thought of as a method invocation. Each step defines a set of resource types needed to support its execution. One of these resource types, called the agent, is distinguished from the others in that it specifies the characteristics of the resource instance that is to be responsible for carrying out the execution of the step. Each step can optionally also be preceded by a pre-requisite and/or followed by a post-requisite. Each requisite can be defined to evaluate a boolean expression or decomposed into hierarchical step structure. Each step also specifies a set of formal parameters that is analogous to the argument list of a method. Execution of a runtime instance of a step begins with the identification of the agent that will be made responsible for executing the step based on the resource type specification in the step, followed by the evaluation or execution of the pre-requisite. If the pre-requisite does not throw an exception (details of exceptions are discussed shortly), step execution proceeds by first acquiring the agent instance and then acquiring all other specified resources, binding to the step’s formal parameters a corresponding set of arguments, and then remanding execution of the step to the assigned agent. The step’s execution concludes with the evaluation of the step’s post-requisite (if any).

Assignment of a step to an agent is done by placing that step on the agent’s agenda (each agent resource has an agenda consisting of agenda items, each of which is an instance of a step that the agent is obliged to carry out), a complete specification of the step to be performed, including (but not limited to) specification of its arguments, available resources, its parent, and its children.

Typically, a step has substeps, in which case its execution consists of coordinating the execution of the substeps. It is often the case that the agent for such a step is an *AutoAgent*, which means that execution of the step is sufficiently clear and bureaucratic that the Little-JIL execution system is best qualified to perform it. In Little-JIL, substep execution sequencing is specified as part of the step definition and may be specified as being sequential, parallel, or as a choice among the substep alternatives. Execution of a leaf step is at the discretion of the step’s agent, however.

Of particular interest is the fact that Little-JIL assumes that execution of a step may fail and result in the throwing of a typed exception, where the type indicates the nature of the exception. Thus, for example, failure of a pre- or post- requisite causes the throwing of an exception. But other failures may arise as well. To handle exceptions arising within the scope of a step, each step can specify a set of exception handlers, each annotated with the type of exception it handles. Each exception handler is also a step that may be hierarchically decomposed. Thus exception handlers define the subsequent actions when exceptional events occur as a result of a step execution within the sub-tree under the handling step. Similar to the exception handling mechanism, Little-JIL also allows specification of reactions that define control flow as a result of out-of-scope events such as global messages. Reaction handlers are also steps that can be decomposed. Space limitations prevent describing all the features of Little-JIL. Details of the language can be found in [17] and an example of a Little-JIL process is shown in Figure 1, which we describe next.

Figure 1 shows a part of the definition of a patient care process in an ED. In addition to the coordination specification provided by the step hierarchy, this diagram also shows yellow boxes that describe the types of resources required by some of the different steps. The yellow boxes are annotations, and do not show the more precise language used to make these specifications. To avoid visual clutter, the diagram does not show the parameters for each step.

The root step of this diagram, TreatPatientsInsideED, represents the process of providing care to an individual patient once the patient has been placed in a bed inside the main ED. The TreatPatientInsideED step is connected to its parent (not shown in this figure), whose responsibility is to instantiate the process shown in Figure 1 for each patient arriving at the ED. As a result, at any point during the simulation, there may be many instances of the process shown in Figure 1 running in parallel.

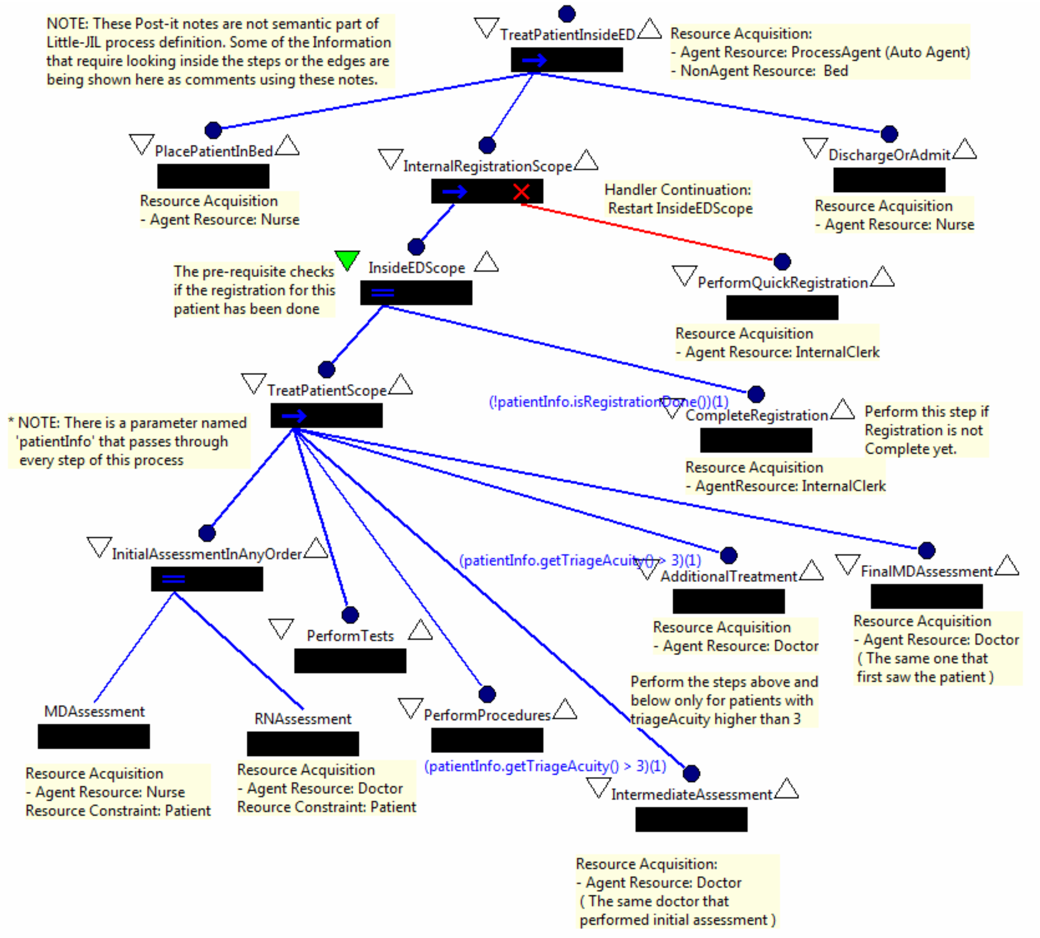
NOTE: These Post-it notes are not semantic part of Little-JIL process definition. Some of the Information that require looking inside the steps or the edges are being shown here as comments using these notes.
TreatPatientInsideED
Resource Acquisition:
- Agent Resource: ProcessAgent (Auto Agent)
- NonAgent Resource: Bed
PlacePatientInBed
Resource Acquisition
- Agent Resource: Nurse
InternalRegistrationScope
Handler Continuation:
Restart InsideEDScope
DischargeOrAdmit
Resource Acquisition
- Agent Resource: Nurse
The pre-requisite checks if the registration for this patient has been done
InsideEDScope
PerformQuickRegistration
Resource Acquisition
- Agent Resource: InternalClerk
TreatPatientScope
(!patientInfo.isRegistrationDone())(1)
CompleteRegistration
Perform this step if Registration is not Complete yet.
* NOTE: There is a parameter named 'patientInfo' that passes through every step of this process
Resource Acquisition
- AgentResource: InternalClerk
InitialAssessmentInAnyOrder
(patientInfo.getTriageAcuity() > 3)(1)
AdditionalTreatment
FinalMDAssessment
PerformTests
Resource Acquisition
- Agent Resource: Doctor
Resource Acquisition
- Agent Resource: Doctor
(The same one that first saw the patient)
Perform the steps above and below only for patients with triageAcuity higher than 3
MDAssessment
RNAssessment
PerformProcedures
Resource Acquisition
- Agent Resource: Nurse
Resource Constraint: Patient
Resource Acquisition
- Agent Resource: Doctor
Reource Constraint: Patient
(patientInfo.getTriageAcuity() > 3)(1)
IntermediateAssessment
Resource Acquisition:
- Agent Resource: Doctor
(The same doctor that performed initial assessment)

Figure 1. A simplified ED process in Little-JIL

Typically, when a patient arrives in an ED, the patient is first triaged outside, assigned acuity level by a triage nurse and then registered by a clerk, where the patient's insurance information is collected and an ID-band is generated and placed on the patient. At this point the patient waits for a bed to become available to be placed inside the ED, where treatment starts. In some cases, a high acuity patient may be immediately placed inside the ED, bypassing the outside registration process. Thus, the process in Figure 1 shows how the registration of such patients is coordinated into the flow of treatment inside the ED.

Figure 1 shows that the agent resource responsible for the step TreatPatientInsideED is an *AutoAgent*, as the main purpose of this step is to create a scope. In this case, the *AutoAgent* starts the patient care process for an individual patient and immediately requests a Bed as a resource. Once the bed resource has been acquired by TreatPatientInsideED a resource instance of type *Bed* then becomes available to steps within

TreatPatientInsideED's scope. The *AutoAgent* then carries on with the process by activating TreatPatientInsideED's children from left to right, starting with PlacePatientInBed to be executed by a *Nurse* agent.

Throughout this process, there is a parameter named *patientInfo* (not explicitly shown in the figure) that passes through each step. This parameter carries information related to the current state of the patient for whom treatment is being provided. As agents carry out different steps, they may use information carried in the *patientInfo* parameter coming into a step as well as set values inside the parameter that are then available to latter steps (and their agents) subsequently. In some cases such as AdditionalTreatment, state information within the *patientInfo* parameter is used to determine if the step needs to be carried out or not. As shown in Figure 1, once the patient is placed in a bed, the patient treatment part of the process begins with the activation of non-leaf steps InternalRegistrationScope and InsideEDScope respectively. Notice there is a pre-requisite (indicated by a green downward arrowhead to the left of the step bar) defined at the InsideEDScope step, whose execution checks the *patientInfo* parameter to determine if some form of registration (external or quick) has been performed for this patient. If this is a high acuity patient who has been placed inside the ED without external registration and if an ID band has not yet been generated through a quick registration step, the pre-requisite step of InsideEDScope will throw an exception of type RegistrationNotDone. The exception will propagate up to the parent step, InsideEDScope, which has been specified to be able to handle this type of exception by activating a handler step named PerformQuickRegistration. This exception handler also specifies how to continue execution once the exception has been handled, namely by restarting execution of InsideEDSceope. An agent of type internal clerk is specified to perform the QuickRegistration step. Note that whenever a quick registration is performed, it needs to be followed by another step to complete the registration. This step, CompleteRegistration, can take place in parallel with other treatment activities being performed on the patient. This potential concurrency is captured by defining CompleteRegistration and TreatPatientScope, two child steps, under a parallel non-leaf step. InsideEDScope and TreatPatientScope both define treatment of a patient inside the ED as a series of tasks that include initial assessment by a nurse and a doctor, performance of tests and treatment procedures, followed by more assessments (IntermediateAssessment) and additional treatments for higher acuity patients. In this process definition the initial assessment is performed by a nurse (RNAssessment) and a doctor (MDAssessment). Both of these steps have been placed under a parallel step, InitialAssessmentInAnyOrder. This specifies that a patient must be assessed both by a nurse and by a doctor, but that these two assessments can not take place simultaneously. This is achieved by defining the patient as a resource, which implies that the patient instance must be acquired as a resource by both MDAssessment and RNAssessment steps. By constraining the patient resource to be the same patient for both steps the resource manager can then ensure that the patient cannot be acquired simultaneously, and thus that the two steps take place sequentially.

The TreatPatientScope sub-process ends with FinalMDAssessment step, where the process specifies that the same doctor who saw the patient earlier will revisit the patient to decide whether to discharge or admit the patient to the hospital. Finally, based on the decision the doctor makes as part of completing the FinalMDAssessment, a nurse agent will be assigned to carry out the DischargeOrAdmit step.

3.2. Process Execution

The simulation capability we are using for this work differs from most simulation capabilities in that our simulations are driven by a Little-JIL process definition that is rigorously defined, can be quite detailed, and is indeed executable. The simulation capability has been created by making relatively modest modifications and enhancements to the Little-JIL process execution capability that already exists. Thus the simulation architecture is best described by first summarizing the Little-JIL process execution architecture. Figure 2a provides a high-level depiction of the Little-JIL execution architecture. The Step Sequencer is a central feature of this system, receiving requests for execution of the process (in this case a notification of a patient arrival), and then supervising the forward progress of process execution as steps complete. The Step Sequencer performs its work by accessing the Little-JIL process to determine which step(s) are to be executed next (based upon information about step(s) that have completed), and then assembles the items needed to get the step executed.

Most specifically, the Step Sequencer consults the Resource Manager to convey requests for resources (an agent resource and other supporting resources) that are instances of the types defined as being needed by the step being executed. The Resource Manager is responsible for searching its internal repository of resource instances and selecting those that seem particularly well-suited for meeting the needs of the requesting step. Determination of which resource is best suited often requires understanding the circumstances under which the step is being performed. Information about circumstances is generally obtained through an inquiry about the state of the process execution. Once the Resource Manager has identified the needed resource instances, the step is placed as an item on the agenda of the selected agent resource. This agenda item also includes the input and output arguments for the step. These arguments are accessed through a Parameter Manager.

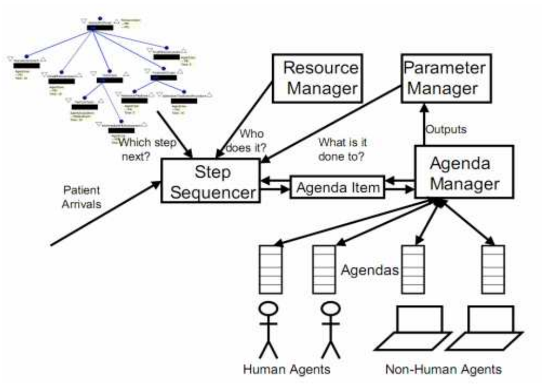

Figure 2a. Little-JIL Runtime Architecture

During execution each agent must monitor its agenda, select a step to be performed, perform the step, and signal step completion once result values have been bound to the appropriate output arguments. Note that the monitoring of an agenda of a non-human agent (e.g. an MRI or an electronic health record system) is probably done by automatic polling. Live agents (e.g. doctors, nurses, and registration clerks) must monitor their agendas themselves. In all cases, an agent may have the capacity to perform more than one step at a time, and so may have multiple agenda items open simultaneously. An agent signals completion of a step by placing an annotation in the step instance's agenda item, and passing the agenda item through the Agenda Management system back to the Step Sequencer, which proceeds with execution of subsequent steps.

3.3. The JSIM Simulation Architecture

As noted above, the JSim simulation capability has been built by making relative modest additions and modifications to the Little-JIL process execution system just described. Figure 2b shows the simulation system architecture. This architecture allows any combination of human and non-human agents to be simulated. In Figure 2b red workstation icons indicate the agents to be simulated. Note in particular that by choosing to simulate all agents except one, the result would be a system that could be of use in training individuals to be agents of the type not simulated. In any case, note that the main additions to the execution system are a simulation TimeLine, facilities for simulating the behaviors of all of those agents that are being simulated (Agent Behaviors), and a facility for collecting the results of a simulation run. In addition, the Step Sequencer has been modified (e.g. so that it accepts instructions about when to proceed to the next step from the TimeLine), and the user now provides information about the distribution of patient arrivals.

Briefly, the JSim simulator works as follows. A simulation begins with the user providing an arrival distribution specification, and specifications of agent behaviors, through the Agent Behaviors module. To begin, JSim initializes the TimeLine to zero to start off a simulation run, initializes the root step of the Little-JIL process to be the step currently being executed, and places a start event for that step in the TimeLine. The simulation then proceeds as an iterative loop in which the most proximate event in the TimeLine is acquired and simulated. The perpetuation of the simulation results from the fact that each step is responsible for placing in the TimeLine one or more events that represent such key activities as step completion, spawning of substeps, etc. Each such event has a designated simulated time at which it is to occur. Thus, for example, a step completion event is generated at the start of the simulation of the step, and the time of this event reflects how long it is expected to take for the step's agent to complete the performance of the step. The TimeLine module keeps all events in sorted order, so that the Step Sequencer can easily determine which event is to be simulated next.

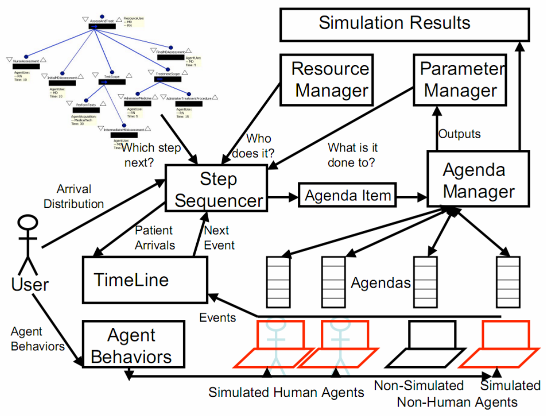

Figure 2b. JSim Simulation Architecture

The Step Sequencer then proceeds very much as it does when executing the process. In particular, it picks event consults the Resource Manager to obtain the needed resources (including the agent) and the

Parameter Manager to obtain the needed input arguments. Once all needed resources and arguments have been obtained, the Step Sequencer packages them into an agenda item and delivers the agenda item to the Agenda Manager for placement on the agenda of the agent assigned to perform the step up the events to be simulated in order and for each. Performance of the step, in turn, results in more events being placed in the TimeLine. To determine the times at which different events, such as starting or completion of a task, are to be performed by an agent, JSim uses the Agent Behaviors module, which has been initialized with information about how to model agent behaviors. To specify simulated agent behaviors in a flexible way, we have developed an XML based rule language [16] called the JSim Agent Behavior Specification language (JABS). Examples of how agent behaviors are specified using JABS is briefly discussed in section 4. This Agent Behaviors specification replaces the actual interaction with live agents in a JSim simulation. JSim allows the specification of agent behavior to be done primarily in two ways:

- Stepwise: There is a specification for how a step execution is to be simulated, and the specification does not vary with different instantiations in the process or for different agents that may perform the step.
- Agentwise: There is a specification for how to simulate the behavior of each different agent that may be assigned to carry out each of the steps to which it might be assigned. Thus, for example, this type of specification allows the possibility that different agent instances may require different amounts of time to perform the same step.

JABS also allows nested specification of agent behavior in order to allow combinations of the above two approaches. In both cases, if the step uses input parameters or produces output parameters, the Agent Behavior Specification must define how the agent uses and converts its input arguments into outputs.

The Resource Manager contains summary data about the capabilities agent resources, such as a list of the types of steps that each agent resource can perform. This information supports the Resource Manager in selecting the agent to be assigned responsibility for performing a step whose execution is to be simulated.

Execution time is estimated by means of a formula that takes into account the max, min, and mean execution times along with a distribution function, agent skill level, agent task load, agent capacity, and inquiry into various execution state parameters (e.g. amount of crowding and threat level). In addition to estimating and modeling execution time, JSim also allows for estimation and modeling of the lag time between step assignment, and initiation of step execution. For now, these formulas are estimates based on interviews with ED professionals, and analysis of statistical data. These formulas will have to be improved and sharpened as our work progresses.

For the ED simulation example, specification of the transformation of parameters is made easier by restricting the number of parameters used by the steps of the simulation. The only parameter type of importance in these simulation runs is the *patientInfo* packet that represents a patient to whom health care is being delivered. Thus each step takes the *patientInfo* descriptor packet as input and sends it as output after having added to or modified information inside the packet according to the Agent Behavior Specification, discussed above.

The *patientInfo* packets are initialized with patient information such as acuity level, and other items needed to accurately reflect the Patient Distribution specification input by the user of the simulation. JSim supports the definition of parameters that define this distribution, and is potentially capable of addressing such issues as arrival rates and distributions of acuity levels and symptoms. Using these parameters JSim generates patient arrivals according to the Patient Distribution specification parameters. Each patient arrival signal causes the instantiation of a new instance of TreatPatientInsideED, as shown in Figure 1.

Finally, note that it is quite challenging to address the need to model the selection of resource instances (especially the agent resource instance) to be used in responding to a resource request from a step. This is the subject of ongoing research in developing a flexible resource manager that can support resource requirements during the execution or simulation of complex processes [11, 12]. As already noted, the problem here is to select a resource instance whose characteristics and current situation most closely match the needs of the step to be executed, under the current circumstances. Considerable literature on resource allocation indicates that there are many reasons why this is a very difficult problem. In the case of ED resource allocation the problem is made even more difficulty by our sense of the desirability of incorporating additional flexibility from resource substitutability. Our experience suggests that modeling resource substitutability could help identify strategies for facilitating patient flow and improving the utilization of scare resources.. The details of how we have approached this problem are beyond the scope of this paper. But it is important to note that Resource Management is a well-defined separate component in our simulation architecture, and thus changes in resource assignment strategies and substitutability policies should be readily incorporated into new simulations by either modification or replacement of an existing Resource Manager (e.g. by new Resource Managers specialized for different domains).

Indeed, a key feature of the architecture of this simulation system is separation of concerns such as is exemplified by having Resource Management comprise a separate component. The architecture incorporates a separate component for the definition of processes, a different component for dealing with resources, and still another component for representing the behaviors of the various resources/agents under different circumstances. In creating this separation, we have created a simulation facility with an unusually large amount of flexibility. In particular we expect that it should be quite feasible to use it to relatively easily carry out the following kinds of comparative studies:

- For a given process, run different simulations with different resource mixes. A goal in doing this might be to determine which mixes of resources best fit the process that is currently in use.
- For a given process, modify the resource mix over time. Our approach should be better able to produce simulation results that are consistent with observed ED performance by enabling replication of how ED resource mixes change over time. This approach would also be useful in studying how different changes in resources mixes could effect better resource utilization and reduced delay in treating patients.
- For a given resource mix, run different ED process variations. Studies of this sort might be particularly useful in cases where there are severe constraints on resource availability. These studies should be able to suggest how best to utilize the resources that are available. The simulations might, for example, suggest how an ED might modify its procedures in periods of congestion.
- For a given resource mix and a given process, use different resource substitution rules. Studies of this sort might complement studies of how to modify processes when resources are highly constrained by suggesting how to modify the priorities that agents (especially human agents) attach to the handling of their tasks. Thus, for example, these studies might suggest the possible benefits of having physicians and nurses cover some of each other’s tasks under certain exigent conditions.

Other types of simulation studies suggest themselves as well. But a particular feature of these studies is their ability to fix one area of concern (e.g. the process), and then evaluate it against variations in various other areas of concern. Thus our goal is to explore the efficacy of our factored approach to the creation of simulations.

4. Experience

We have implemented a system having the architecture depicted in Figure 2b, and have used it to create simulations that are driven by a range of processes defined in Little-JIL. While we have not yet completed many of the studies that we have outlined in the previous section, we have completed some prototype simulations and done some preliminary evaluation of their results. This section presents a representative sample of some of these simulations.

We started out with a simplified ED process such as the one shown in Figure 1. We modeled patient arrivals as specific patient arrival events placed initially in the TimeLine. We specified agent behaviors by using JABS to specify the type of agents and other resources required for carrying out each process step. For example, the leaf-steps QuickRegistration, MDAssessment, and RNAssessment require as agents registration clerks that work inside the ED, a doctor, and a nurse respectively. TreatPatientInsideED specifies the need for a *Bed* resource.

4.1. Specifying Scenarios and Agent Behavior

For initial simulations, we first populated the TimeLine with a fixed set of patient arrivals to begin the simulation run. Later we used a Poisson distribution of mean 10.0 to generate the times at which different patient arrivals are simulated. We say that a set of patient arrival events makes up the *scenario* for a simulation run. Figure 3 shows an example of the specification of one such scenario, namely one in which there are three patient arrivals, and they occur at times, 8, 17, and 23.

```
<scenario>
  <message
    type="edparams.PatientArrivalMessage"
    time="8"/>
  <message
    type="edparams.PatientArrivalMessage"
    time="17"/>
  <message
    type="edparams.PatientArrivalMessage"
    time="23"/>
</scenario>
```

Figure 3. Example of patient arrival specification

Figure 4 shows an example of part of an Agent Behavior specification.

```
<step name="PlacePatientInBed">
  <started>
      <complete>
        <fixed value="10" />
      </complete>
  </started>
</step>
<step name="CompleteRegistration">
 <started>
  <group>
    <set-field parameter="patientInfo">
      <field name="isRegistrationDone">
        <boolean value="true" />
      </field>
    </set-field>
    <complete>
       <linear-range min="10" max="20" />
    </complete>
   </group>
  </started>
</step>
<agent name="ha001-doctor">
     <step name="MDAssessment">
       <started>
         <complete>
           <linear-range min="10" max="20"/>
         </complete>
        </started>
     </step>
</agent>
```

Figure 4. Example of Agent Behavior Specification

In Little-JIL, each step is formally defined using a finite state automaton. During the execution or simulation of a Little-JIL process, a step goes through the states *posted*, *started*, *completed*, and/or *terminated*. Although not required, for very fine-grained control JABS supports specification of the behavior of an agent upon entry into each of these states. Figure 4 shows examples of how such behaviors can be specified upon entry into the "started" execution state for some of the steps in the process shown in Figure 1. In this example, once the PlacePatientInBed step is started, any agent made responsible for carrying out the step, will generate the *complete* event on this step after 10 simulation time units. Similarly, the second rule in Figure 4 specifies the time it takes for any agent assigned to the step CompleteRegistration to be computed using a uniform distribution between 10 and 20 simulation time units. The agent behavior specified as part of completing the CompleteRegistration step also sets the value of a boolean field inside the 'patientInfo' parameter to become true. The third rule, which is a nested rule, specifies that a specific doctor agent, with id 'HA001' takes somewhere between 10 to 20 simulation time units when assigned the task of performing MDAssessment.

4.2. Summarization of Simulation Output

The most basic form of output from a JSim simulation run is a trace. Simulation run traces include such information as when each instance of a step became ready to be executed (entered the "initial" state), when it was assigned to an agent's agenda (entered the "posted" state), when the agent started performing that step (entered the "started" state), and when it completed the step's execution (entered the "completed" state). From this basic trace output we can then compute a range of summary information. In particular, in our early work we have computed the following kinds of summary information:

- *Length-Of-Stay (LOS) for patients*: This is the time a patient spends from arrival to discharge or hospital admission.
- *Resource Utilization*: The fraction of time resource instances are actually being used (i.e. are assigned as agents or other resources for the execution of one or more steps) as opposed to waiting for steps to be assigned to them.

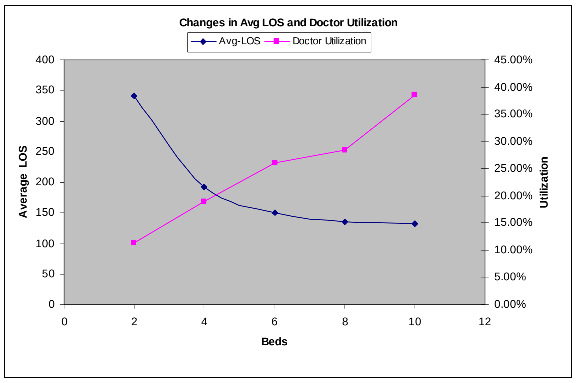

Figure 5. Sample result of simulation runs

This simulation capability has allowed us to explore how variations in resource mixes and allocations seem to affect such key measures as LOS. Thus, for example, we have run a number of simulations in which the ED process was fixed (using the process shown in Figure 1), but with different resources mixes. Figure 5 shows some of the summary results. In this case, all resources except the number of available beds were kept fixed, and the number of beds was varied from 2 to 10. As expected, the average length of stay (the blue line in Figure 5) shows a decrease with increasing numbers of beds. However, we see less reduction in the average LOS as more beds are added. In general, this is to be expected as we have kept all other resources fixed. We see a diminishing effect on reducing length of stay as a result of adding just one type of resource. In the same graph, we have plotted (red line) the average utilization of doctor resources. We can see that, with the given resource mix, the doctor utilization increases steadily with increasing numbers of beds. This is also an expected behavior, as doctors were underutilized when there were fewer beds, and more beds allowed more patients to be brought in to be seen by doctors, thus reducing idle time for doctors. However, the doctor utilization does not improve in a straight line or a smooth curve. This unexpected outcome could be a result of the particular scenarios or resource mixes we have used. It clearly underscores the need for careful validation of simulation output results, an issue that we address shortly.

In our very early work, we have modeled different mixes of resource instances such as doctors, nurses, clerks, beds etc. inside the Resource Manager repository, all using the same ED process. We have also created different ED processes (with a lot more details than is shown in Figure 1), and run simulations using the same fixed resource mix. We have also run simulations with various numbers of patients, different arrival distributions, and different simulation periods. This work has reinforced our view that the separation of concerns supported by the JSim architecture makes it quite easy to carry out a wide range of comparison studies, where the analyst can focus on process changes, resource mix changes, patient mix changes, or various combinations of these factors. For example, the simulation results for this paper were obtained by simplifying a process that we had been using previously, but that is too large for the space available in this paper. To create and simulate the process described here, we simply modified the process by deleting a few steps and abstracting out the lower level details of others such as PerformTests and PerformProcedures. Then we made minor modifications to the agent behavior specification to support this modified process. The changes in the resource mix specification, as it is dealt with by a separate component, were easy and largely independent of the process changes. All changes were completed in less than one hour.

5. Future Work

The early results obtained using this simulation facility seem encouraging. But much work remains to be done if we are to have enough confidence in this capability to use it as a predictive tool to suggest improvements in the performance of actual EDs. Thus, we suggest two future thrusts for this work: validation and prediction.

Validation of our simulation facility: Simulations of a simplified ED process with minimal resources have usually shown output results that are in line with our expectation, although, as shown in Figure 5, some results may appear to be counterintuitive, at least at first. Thus we have immediately encountered the well-known problem of validating a simulation. While we are testing our simulations, we understand that testing is of limited value, as software testing typically assumes the existence of an “oracle”, which encapsulates information about what comprises “correct” output. Especially in view of the fact that a simulation is intended to make predictions and produce results that are not previously known, such oracles are difficult to devise. One type of oracle that we are exploring, however, is actual practice. To gain more confidence in our simulation results, we plan to compare our simulated models with actual EDs of various size and complexity to see how our results compare to actual patient lengths of stay.

In addition, we are carefully examining our simulation trace output to be sure that the simulation is carrying out process steps in the right order, acquiring resources that are appropriate, and modeling agent behaviors in ways that are consistent with our inputs. These simulation validation efforts will continue in parallel with our increasing attempts at prediction for the foreseeable future.

Prediction: Our well-modularized simulation infrastructure architecture provides considerable flexibility in performing different types of simulations. As suggested above we plan to use this capability for predicting effects of changes in a) process, b) resource utilization, c) priorities of steps, d) resource substitution policies, and e) various combinations of all these. We plan to perform different ‘what if’ types of studies related to changing resource mixes, patient arrival patterns, and changes in process. For example, we want to study the effect of allowing certain steps in the ED process that are usually performed sequentially to run in parallel.

We are also very interested in studying effects of dynamic changes in the process definition or resource assignment based on the state of the running system. For example, usually a nurse is the type of agent that performs the patient discharge activity. We would like to study the impact of starting to assign this step to doctor agents when patient waiting time crosses a certain threshold. Simulating this sort of dynamism seems to be difficult for many existing simulation systems, but seems relatively straightforward using our approach. Thus, attempting this kind of simulation and determining whether it yields results that are truer to actual observed practice should help us to determine whether the modularity and flexibility of our JSim architecture does indeed offer the benefits of greater accuracy that we have been suggesting.

In general, our future experimentation will be aimed not only at attempting to improve the performance of EDs, but also at coming to understand better the software architectural choices that seem to support greater experimental flexibility and improved simulation accuracy. Clearly this work is only just beginning.

6. Acknowledgements

We would like to thank Prof. George Avrunin for his valuable feedback about this early work. This material is based upon work supported by the National Science Foundation under Awards CCF-0427071, CCF-0820198, CCF-0829901, and IIS-0705772.

Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation

7. References

[1] Chun, A. H. W., S. H. C. Chan, et al. (2000), “Nurse Rostering at the Hospital Authority of Hong Kong”, Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence 0-262-51112-6, AAAI Press / The MIT Press**:** 951-956.

[2] Connelly, L. G. and A. E. Bair (2004), Discrete Event Simulation of Emergency Department Activity: A platform for System-level Operations Research." Academic Emergency Medicine, vol. 11, no. 11: 1177-1185.

[3] Draeger, M. A. (1992), “An Emergency Department Simulation Model Used to Evaluate Alternative Nurse Staffing and Patient Population Scenarios”, Proceedings of the 24th Conference on Winter Simulation, Arlington, VA.

[4] Evans, G. W., Gor, T. B., and Unger, E. (1996), “A simulation model for evaluating personnel schedules in a hospital emergency department”, Proceedings of the 28th Conference on Winter Simulation, Coronado, CA.

[5] Hammann, J. E. and Markovitch, N. A. (1995), “Introduction to ARENA”, *Proceedings of the 27th Conference on Winter Simulation*, Arlington, Virginia.

[6] Hay, A. M., E. C. Valentin, et al. (2006), “Modeling Emergency Care In Hospitals: A Paradox - The Patient Should Not Drive The Process”, Proceedings of the 38th Conference on Winter Simulation, Monterey, California.

[7] Hoot, N. R., Leblanc L. J. et al. (2008), “Forecasting Emergency Department Crowding: A Discrete Event Simulation”, Annals of Emergency Medicine, Vol. 52, No. 2.

[8] Krahl, D. (2003), “Extend: an interactive simulation tool: extend: an interactive simulation tool”, *Proceedings of the 35th Conference on Winter Simulation: Driving innovation*, New Orleans, Louisiana, December 07 - 10, 2003.

[9] Kumar, A. and R. Kapur (1989), “Discrete Simulation Application - Scheduling Staff for the Emergency Room”, IEEE Winter Simulation Conference, E. A. M. a. K. J. M. a. P. Heidelberger. Washington, D.C.**:** 1112--1120.

[10] McGuire, F. (1994), “Using Simulation to Reduce Length of Stay in Emergency Departments”, IEEE Winter Simulation Conference, J. D. T. a. S. M. a. D. A. S. a. A. F. Seila. Orlando, FL**:** 861--867.

[11] Raunak, M. S., Osterweil, L. J. (2005), “Effective Resource Allocation for Process Simulation: A Position Paper”, Proceedings of the 6th International Workshop on Software Process Simulation and Modeling (ProSim), St. Louis, MO, May 2005.

[12] Raunak, M. S., Osterweil, L. J. (2005), “Process definition language support for rapid simulation prototyping”, Proceedings of the International Software Process Workshop (ISPW), Beijing, China, 2005.

[13] Raunak, M. S. (2007), “Resource Management in Complex and Dynamic Environments”, Department of Computer Science, University of Massachusetts Amherst.

[14] Rossetti, M. D., Trzcinski, G. F. et al. (1999), “Emergency department simulation and determination of optimal attending physician staffing schedules”, Winter Simulation Conference, Squaw Peak, Phoenix, AZ.

[15] Samaha, S., W. S. Armel, et al. (2003), “The use of simulation to reduce the length of stay in an emergency department”, WSC '03: Proceedings of the 35th Winter Simulation Conference, New Orleans, Louisiana, 1907-1911.

[16] Wise, A. (2008), “JSim Agent Behavior Specification Language”, http://laser.cs.umass.edu/documentation/jsim/la-nguage.html.

[17] Wise, A. (2006), “Little-JIL 1.5 Language Report”, University of Massachusetts Amherst, Amherst, MA.