Resource Specification for Prototyping Human-Intensive Systems

Seung Yeob Shin[1], Yuriy Brun[1], Leon J. Osterweil[1],
Hari Balasubramanian[2], and Philip L. Henneman[3]

[1] School of Computer Science, University of Massachusetts, Amherst, MA, USA
{shin,brun,ljo}@cs.umass.edu
[2] Department of Industrial Engineering, University of Massachusetts, Amherst, MA, USA
hbalasubraman@ecs.umass.edu
[3] Department of Emergency Medicine, Tufts-Baystate Medical Center, Springfield, MA, USA
henneman@baystatehealth.org

Abstract. Today's software systems rely heavily on complex resources, such as humans. Human-intensive systems are particularly important in our society, especially in the healthcare, financial, and software development domains. One challenge in developing such systems is that the system design must account for the constraints, capabilities, and allocation policies of their complex resources, particularly the humans. The resources, their capabilities, and their allocation policies and constraints need to be carefully specified, and modeled. Toward the goal of supporting the design of systems that make effective use of such resources, we introduce a resource specification language and a process-aware, discrete-event simulation engine that simulates system executions while adhering to these resource specifications. The simulation supports (1) modeling the resources that are used by the system, and the ways in which they are used, (2) experimenting with different resource capability mixes and allocation policies, and (3) identifying such undesirable situations as bottlenecks, and inefficiencies that result from these mixes and policies. The joint use of detailed resource specifications and simulation supports rapid evaluation of human-intensive system designs. We evaluate our specification language and simulation framework in the healthcare domain, on a software system for managing a hospital emergency department.

1 Introduction

Many software systems constantly interact with humans and other complex resources. Insufficient attention to these interactions at system design time can reduce the quality and effectiveness of the system. In this paper, we tackle the development of software systems that interact with complex resources. We argue that understanding both the *process* the resources follow, and the resources themselves in terms of their *availability*, *skills*, and *constraints*, early in the development process can improve system quality, ease validation by directly involving domain experts and customers in the design process, and allow for documentation of assumptions and requirements, communication among the developers, and traceability and improved debugging.

We provide a language for formally specifying and modeling complex resources and their interactions with one another and the software components, and an automated

A. Egyed and I. Schaefer (Eds.): FASE 2015, LNCS 9033, pp. 332–346, 2015.
DOI: 10.1007/978-3-662-46675-9_22

discrete-event simulator to support dynamically analyzing the effects of different resource mixes and resource-allocation policies.

We motivate and evaluate our work through a representative example scenario based on a hospital's emergency department (ED) modeled after the Baystate Medical Center ED, in Springfield, MA, USA. An ED administrator is tasked with making the ED more efficient. The ED sees an average of 288 patients per day, employs 26 doctors, 41 nurses, 5 triage nurses, and 16 clerks, and houses 48 beds, 2 x-ray rooms, and 4 CT-scan machines. The administrator notices that the patients' average length of stay (LoS) is 297 minutes, exceeding the national average, and that patients spend, on average, 20 minutes in the waiting room before being seen. Further, the doctors and nurses are underutilized during the night and overutilized during the day. The administrator decides that the ED needs a modern software system to manage patient care, billing, supplies, and staff. Part of this system is a patient-management software component that will track each patient and allocate doctors, nurses, and other resources. This component has to make decisions about resource allocations, manage resources, such as beds and equipment, and also interact with the human resources, such as doctors, nurses, and technicians. The administrator's goals are to (1) automate the patient handling process, (2) evaluate resource allocation policies, (3) understand the constraints that impact resource utilization to develop shift schedules that balance utilization, and (4) understand the hospital's efficiency bottlenecks. To design and implement the patient-management component, the developers will need to interact with domain experts to model the process patients undergo, and the involved resources. This model will serve as documentation, enable simulation to evaluate resource-allocation policies, detect resource utilization inefficiencies, and support the administrator's decisions about how to best spend money on resources.

This paper makes the following three contributions:

1. A precise resource specification language for capabilities, interactions, allocation policies, and scheduling constraints of complex resources.
2. A process-aware discrete-event simulator JSim that respects the resource specifications and constraints.
3. A case study applying our approach to discover implications of resource allocation and scheduling policies, and thus helping guide the design of a hospital ED patient-management software system.

2 Resource Modeling

This section describes our language for specifying resources, their capabilities, and the constraints and policies governing their allocation.

2.1 Resource Characteristics

We identified six aspects of resources that must be specified to enable accurately assigning resources to tasks:

1. the resource's *capabilities*, (e.g. the tasks it may perform),
2. *attributes* (e.g., certifications and experience),

3. a *guard* qualification of those capabilities and attributes based on the dynamic state of the system (e.g., a doctor's availability is affected by the number of patients she is already caring for),
4. an *assignment policy* for enforcing resource constraints (e.g., the doctor who performed a surgery is the one who discharges the patient),
5. a *contention policy* for activity selection when multiple activities require the same resource instance (e.g., when one doctor is caring for multiple patients, the critical patients must come first), and
6. a *selection policy* for resource selection when multiple resource instances satisfy the needs of an activity (e.g., assigning a doctor to a new patient).

We designed our resource-specification language around the ability to easily and precisely specify each of the above aspects. In our language, a resource is a collection of *capabilities* and *attributes*. Each of a resource's capabilities is qualified with a *guard*, a predicate that can specify constraints and policies for allocating resources. For example, the specification of a resource with two capabilities, `triage` and `treat`, may have guards indicating that the resource's triage skills are high for all patients except gunshot-wound patients, and are particularly low late in the evening.

Each capability's *contention policy* specifies how the system decides what to do when that resource is requested by multiple activities. The contention policy is evaluated dynamically, based on an activity's priority, resource needs, and attributes. For example, a doctor may be allocated to patients on a first-come first-served basis, based on the severity of the illness, randomly, or some combination of these.

Finally, each capability's *selection policy* similarly specifies which of a set of suitable resources (e.g., a set of qualified and available doctors) is chosen. Again, this policy is evaluated dynamically.

2.2 Resource Model

Figure 1 shows the static relations among the entities that define a resource. This metamodel is an extension of an earlier, less expressive resource metamodel [14]. As already noted, a resource is composed of *capabilities*, tasks the resource can perform, and *attributes*, which describe the characteristics of the resource. Thus, for example, a `cost` attribute should be included in the specification of the doctor resource if the cost effectiveness of the doctor is to be analyzed. A particularly

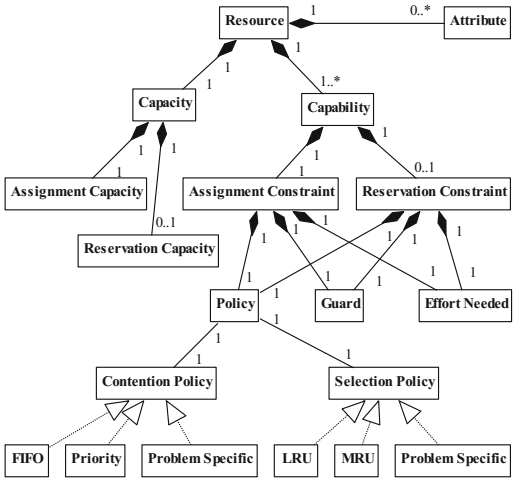


Fig. 1. The metamodel of a resource specification describes the static relations of the entities that define a resource

Attribute Declaration

```
<declare−attribute name="location" type="string" />
<declare−attribute name="working" type="boolean" />
```

Resource Model

```
<resource type="TrRN">
<attribute name="location" value="" />
<attribute name="working" value="" />
<capacity assignment_available="1" />
<capability name="Triage">
<assignment guard="working==true" effort_needed="1"
    contention_policy="FIFO" selection_policy="LeastUtilizedFirst : ProblemSpecific" />
</capability>
</resource>
```

Resource Instances

```
<instantiate type="TrRN" number="2" />
<instance type="TrRN" id="1" set_attribute="location" value="shared" />
<instance type="TrRN" id="1" set_attribute="working" value="true" />
<instance type="TrRN" id="2" set_attribute="location" value="shared" />
<instance type="TrRN" id="2" set_attribute="working" value="true" />
```

Fig. 2. The resource model for the process of handling patients specifies a triage nurse resource (`TrRN`). The attributes of the resource model (top) are used by the triage nurse model (middle). Two instances (bottom) of the resource show two sample triage nurses.

key attribute is `capacity`, comprised of `assignment capacity` and `reservation capacity`, This attribute bounds the number of activities a resource may participate in simultaneously, and is used to ensure that the simulator does not allocate to any resource more activities than that resource can handle. Both assignment and reservation capacities are needed because, for example, a doctor may care for multiple patients (up to the reservation capacity bound), but can only work on one at a time (assuming the assignment capacity is set to 1).

Figure 2 shows an example of a resource specification. It defines the attributes of the resource model (top), the resource model itself (middle), and two instances of triage nurse resources (`TrRN`).

Allocating resources to a given activity requires knowing more than the capability and capacity of the resources. It also requires knowing (1) the availability of each resource, (2) the (estimated) effort the given activity requires from a resource, and (3) constraints on reserving and assigning to each resource. Each capability consists of a reservation and an assignment constraint. Each constraint is specified in the form of a `guard`, a Boolean expression defined over the dynamic variable values of the process. For example, an assignment guard might be used to specify that only a triage nurse who is working can be assigned as a resource to an activity (`<assignment guard = "working==true">` in Figure 2). The guards can also specify what shift a nurse works,

Step	Resource request specification
Triage	triage_nurse: Triage, `blocking`
Register	clerk: Register, `blocking`
PlaceInBed	bed: PlaceInBed, `blocking`
Treat	reserved_doctor: MDTreat, 1, `replaceable`, `blocking`
Treat	reserved_nurse: RNTreat, 1, `replaceable`, `blocking`
RNAssess, RNDischarge	nurse: RNTreat, `blocking`, reserved_nurse
MDAssess, Procedure, MDDischarge	doctor: MDTreat, `blocking`, reserved_doctor

Fig. 3. Resource request specifications. Each step in Figure 4 has a resource request specification associated with it.

when the nurse takes breaks and eats meals, and when personal considerations allow the nurse to work late, or leave early.

Human resources exert effort in performing activities, whereas other resources (e.g., beds and equipment) do not. For resources that exert effort, the amount of effort can be estimated using optional `skill level` and `experience` attributes. (Our triage nurse example does not employ these optional attributes.)

Finally, constraints on resource contention (multiple activities requesting the same resource instance) and activity contention (multiple resource instances capable of providing the capability requested by a given activity) are specified by the *contention policy* and *selection policy*, respectively. First-come first-serve (`FIFO`) is an example of a built-in policy, but custom policies can be defined as arbitrary functions over the dynamic variables of the process. Least utilized resource first (`LeastUtilizedFirst`, see Figure 2) is an example of a custom policy. Other built-in policies not shown in Figure 2 include least (`LRU`) or most (`MRU`) recently used policies, and a policy based on the priority of the request (`Priority`).

2.3 Resource Request Model

The resource model includes specifications of resource requests. We separate resource requests from process activities (described in Section 3) into two types, a reservation request and an assignment request. Each activity generates a separate request for each resource it needs. Figure 3 shows several examples of resource requests:

Reservation Request: reserved-resource: capability, count, [`replaceable`,] `blocking` | `nonblocking`
Assignment Request: resource: capability, `blocking` | `nonblocking` [, reserved-resource]

Both reservation and assignment requests ask for an available resource that performs a particular capability. Which resource is returned depends on the dynamic state of the process. For example, a doctor may be assigned to drawing a patient's blood, but only when all nurses are fully assigned, and only when the blood draw task is considered to require a small amount of effort and a low skill level. Our request model supports the use of blocking requests (see the `blocking` and `nonblocking` keywords in the request definitions) to ensure that only fully qualified resources are allocated to the activity. The `replaceable` keyword means that a resource may be replaced by another, under certain situations.

3 Process Modeling

Our approach represents processes by an executable language that describes a sequences of steps. Each step specifies the need for one or more resources. We use of the Little-JIL process definition language [19], which has been used in previous work to support the definition of complex processes in the medical, election, software development, and other domains [3, 13, 15, 20, 21]. Little-JIL is not a contribution of this paper. We did, however, enhance Little-JIL by augmenting each step specification with an allocation (either reservation or assignment) request.

We now outline the Little-JIL features most relevant to our work on resource specification. We refer the reader to prior work for a complete language definition [19]. A Little-JIL activity specification is defined using hierarchically decomposed steps. A step represents an activity or a task that is part of the modeled process. Each step has a name and a set of badges to represent control flow among its substeps, its interface (a specification of its input and output artifacts and the resources it requires), and the exceptions it handles. A step with no substeps is called a leaf step and represents an activity to be performed without any explicitly defined process guidance.

Every non-leaf step has a sequencing badge (an icon embedded in the left portion of the step bar), which defines the order of substep execution, such as sequential (right arrow), in parallel (equal sign), one chosen from a set (circle with a horizontal line), or in sequence in which substeps are to be tried as alternatives (right arrow with an $\times$ on its tail). The latter two kinds of steps enable specification of certain kinds of uncertainties that might arise during process execution.

Each Little-JIL step interface specifies the types of resources the step requires. Allocating a resource instance to the step happens dynamically, during process execution by the resource manager. We use the ROMEO resource manager [12]. Every step requires at least one resource, specially designated as the step's agent. Little-JIL agents may be either humans or automated devices.

Figure 4 shows an example high-level process definition in Little-JIL that specifies the process of handling patients in an ED. When a patient arrives, she is received, which consists of being triaged, registered, and then placed in a bed. Being placed in a bed involves being treated: assessed by a nurse and then a doctor, undergoing procedures,

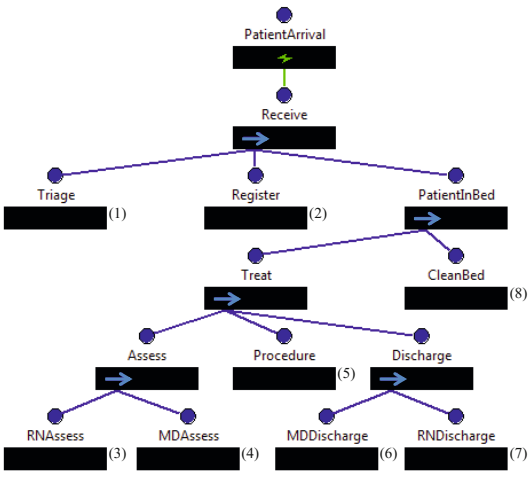


Fig. 4. A patient treatment process, specified with the Little-JIL process model language [19]. This high-level model abstracts away lower-level steps, denoted in (parentheses). The full, detailed model can be found at `http://people.cs.umass.edu/~shin/ed/`.

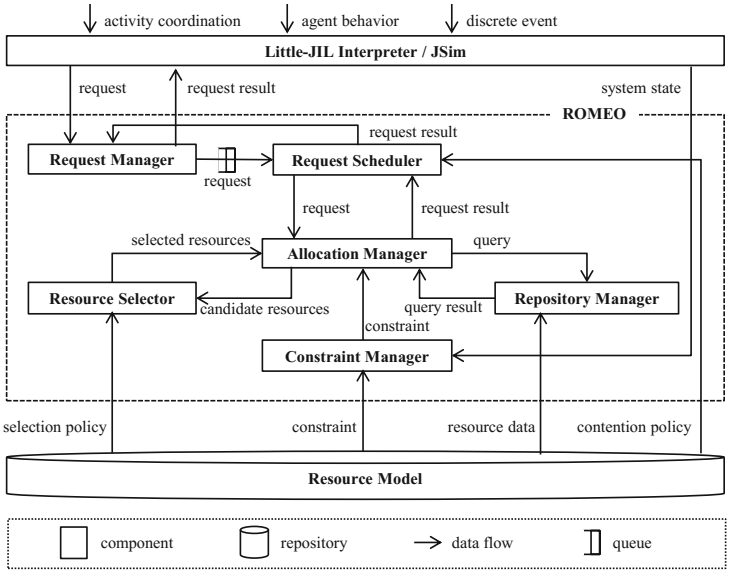


Fig. 5. A view of the resource-aware JSim discrete-event process simulator that focuses on ROMEO, the resource management component

and being discharged, first by a doctor and then by a nurse. Finally, the bed is cleaned and made available for assignment to support the treatment of another patient.

While this process activity flow is relatively simple, its execution depends on details of the resource model that may contain intricate dependencies among the doctors, nurses, beds, and other resources.

4 JSim Resource-Aware Simulator

The final piece of our approach is a discrete-event simulator that simulates processes whose activities are modeled in Little-JIL, and whose resources are modeled in ROMEO. To this end, we have extended JSim [14], an existing discrete-event simulator of Little-JIL/ROMEO processes. Our extensions include support for (1) resource instances, (2) the two-phase resource reservation and allocation, and (3) contention policy constrains to select preferred requests, selection policy constrains to select preferred resources. These extensions enable the simulator to support such scenarios as (1) allowing doctors to have varying shift constraints, (2) enforcing the same doctor always handles a given patient, unless the doctor's shift ends, at which point the patient is handed off to a new doctor, (3) enabling a variety of scheduling policies, such as handing the sickest patient first, or using the least utilized resource first. Additionally we have redesigned the simulator's architecture to improve the separation of concerns and localization of key aspects of the system.

Figure 5 shows the extended JSim architecture, focusing on the resource handling aspects of the simulator. The architecture separates the activity issues from resource

concerns, which makes it easy to keep track of resource allocations, utilizations, waiting times, and other properties that other simulators have difficulty reporting (see Section 6). In addition, extended JSIM treats resource constraints as a separate concern, which eases changing them, facilitating experimentation with different resource allocation strategies.

The Little-JIL Interpreter is an abstract representation of a number of JSim components not related to resource management. The activity coordination artifact represents the Little-JIL activity definition. The agent behavior artifact specifies details (such as speed and cost) of the way in which resource instances provide their capabilities. The discrete event clock is an event arrival stream that triggers the execution of process activities. During simulation, activities first acquire and then release resources by sending requests to ROMEO, which evaluates the dynamic guards and constraints on each resource and determines which resources satisfy the request, and responds with either a satisfactory resource, or a message that the needed resource is unavailable.

The Request Manager receives and responds to reservation, assignment, and release requests, which are queued by contention policies in the resource model. The Allocation Manager allocates resources in collaboration with the Resource Selector, Constraint Manager, and Repository Manager components. When the Allocation Manager receives an allocation request for a capability, the Repository Manager identifies candidate resources, the Constraint Manager evaluates the resources' guard constraints to remove unsuitable resources, and the Resource Selector evaluates the selection policies to select a candidate. For a release request, the Allocation Manager interacts with the Repository Manager to adjust released resources' allocated capacity values.

This architecture yields flexibility for exploring resource constraint specifications effectively. For example, the Request Manager can be instrumented to record the history of requests, and identify resource bottlenecks by finding which resources spend the most time waiting for activities, and which activities spend the most time waiting for resources. The Allocation Manager likewise can be instrumented to record the resource allocation history and compute resource utilization levels over various time granularities (e.g., hourly), even taking into account resource unavailability due to such events as lunch times and breaks.

5 Case Study: Emergency Department

We evaluated our approach by using our resource-aware discrete-event simulator to expedite the simulation and evaluation of a range of ED operations management strategies. Our aim was to show that our approach can suggest the characteristics and design of a system that has superior operational behavior, and that respects even very intricate resource characteristics and constraints. We used specifications of various ED operational practices, resource characteristics and mixes, and allocation approaches to run JSim simulations aimed at understanding the effects of these specifications on such key operational characteristics as patient waiting time, resource allocation levels, and overall costs. Our domain expert, who both helped us develop the specifications and validate the models and simulation results, has decades of experience as an emergency physician and an ED manager at the Baystate Medical Center Emergency Department, in Spring-

field, MA, USA. He further has significant experience with research in discrete-event simulation of EDs [2].

5.1 Emergency Department Characteristics

ED resources range from beds, blood, and x-ray devices, to a spectrum of human resources, from receptionists and porters, to nurses of varying kinds, to doctors with various specialties and skill levels. Resources are always scarce in order to keep down the costs of operating the ED. Therefore, wise allocation of these resources is necessary to assure timely and competent care. A full description of the ED activities, resources, constraints, and policies is beyond the scope of this paper. Instead, we describe only a few details to illustrate the complexity of the models. Some ED characteristics that posed interesting challenges for us include:

Six acuity levels: ED patients are classified into six acuity levels based on the severity of their ailments. The care process varies based on the acuity. A level-six (sickest) patient is immediately allocated a doctor (MD), a nurse (RN), and a bed. These patients also get highest priority in x-ray room and CT room allocation. In contrast, a level-one patient undergoes fewer procedures, each of which with a lower resource allocation priority.

Arrivals: Our process definition allows patients to arrive in two ways: Critical patients, by definition, always arrive by ambulance, while other patients arrive on their own. Critical patients are the sickest (acuity level six), while the others are categorized into the remaining five acuity levels. Patient arrival rates over the 24-hour period are specified by a Poisson distribution, based on actual arrival rates at the Baystate Medical Center.

Staffing: Human resources work on a shift system; the number of available humans varies over 24 hours. Typically, an MD or an RN will work one of three different 8-hour shifts, although our simulations suggest greater flexibility in the start times and durations of shifts could lead to shortening the patients' LoS.

Same MD-RN constraints: A patient assigned to a bed is cared for by the same MD and RN throughout the stay, with changes only when the MD's or RN's shifts end, or they are on a break.

Workload: Estimates of the effort required to perform activities are specified by triangular distributions, based on the Baystate Medical Center data. These estimates, along with the specifications of the skill levels of the resources, dictate the amount of time required by each of the activities.

Fast & Main tracks: The ED operates two tracks. The fast track cares for low acuity patients (levels 1–3), and the main track, high acuity (4–6). Each track has its own beds and MD and RN resources. At night, the fast track closes and its patients are transferred to the main track. During this transfer, fast track resources are deallocated and appropriate main track resources are allocated for the patients.

5.2 Emergency Department Activity Model

While we omit the full models from this paper, for exposition, we present a small subset of the ED process and resource specifications. Figure 6 illustrates the patient-testing

process for an acuity-level-four patient. The `AL4Test` step at the root is a parallel step, meaning the lab test process, `AL4LabProc`, can be done in parallel with `AL4Test`. The 70% annotation on `AL4LabProc`'s pre-requisite means that 70% of acuity-level-four patients require the lab test. For the other tests, a nurse checks a patient's ECG (`RNECG`), and then a doctor checks the ECG result (`MDCkECG`), since `AL4ECGProc` is a sequential step. After the ECG test, a nurse gives a medication to the patient (`RNMedHi`), and the patient is transferred to the CT or x-ray room. This behavior is represented by the `AL4XrayOrCTOrNothing` choice step, which means only one of its child steps will be executed, with step pre-requisites indicating the probability of each alternative. While space constraints prevent us from describing the full patient care process and resource models, they can be found at `http://people.cs.umass.edu/~shin/ed/`.

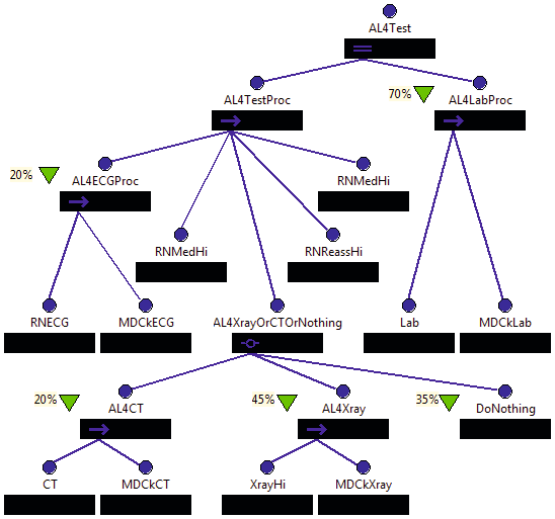


Fig. 6. The Little-JIL definition of the patient testing process, which is part of the care an acuity-level-four patient undergoes in an ED

Step	Resource request specification
RNECG, RNMedHi, RNReassHi	nurse: RNTreat, `blocking`, reserved_nurse
MDCkECG, MDCkCT, MDCkXray, MDCkLab	doctor: MDTreat, `blocking`, reserved_doctor
CT	ct_room: CTScan, `blocking`
XrayHi	x-ray_room: X-rayScan, `blocking`

Fig. 7. The resource request specifications associated with each step in Figure 6

Figure 7 shows the resource requests made by the leaf steps of the process. All requests are blocking, meaning step execution cannot begin until the resource is assigned. For each patient, the MD and RN requests can only be satisfied by the previously reserved resources.

5.3 Emergency Department Resource Model

Figure 8 describes the resource model for `MD`, and shows three example instances of `MD`s. The full model (omitted here), also defines the `RN`, `TrRN`, `clerk`, `bed`, `x-ray room`, and

Attribute Declaration

```
<declare−attribute name="location" type="string" />
<declare−attribute name="shift" type="string" />
```

Resource Model

```
<resource type="MD">
<attribute name="location" value="" />
<attribute name="shift" value="" />
<capacity assignment_available="1" reservation_available="1"/>
<capability name="MDTreat">
<reservation
    guard="location==artifact(patient.location) && time>=start(shift) && time<end(shift)"
    contention_policy="SickestFirst : ProblemSpecific"
    selection_policy="LeastUtilizedFirst : ProblemSpecific"
    effort_needed="0" />
<assignment
    guard="location==artifact(patient.location) && time>=start(shift) && time<end(shift)"
    contention_policy="SickestFirst : ProblemSpecific"
    selection_policy="LeastUtilizedFirst : ProblemSpecific"
    effort_needed="1" />
</capability>
</resource>
```

Resource Instances

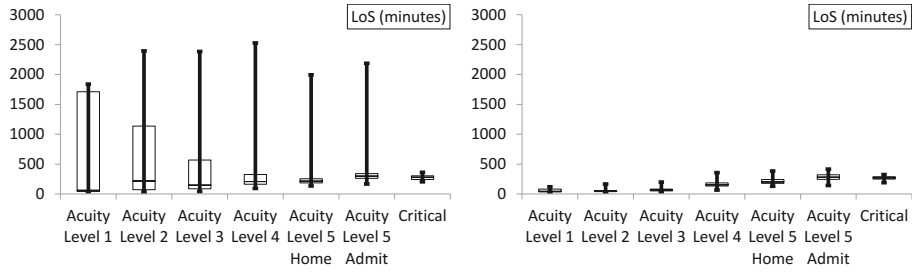
```
<instantiate type="MD" number="3" />
<instance type="MD" id="1" set_attribute="location" value="main−track" />
<instance type="MD" id="1" set_attribute="shift" value="7AM−−3PM" />
<instance type="MD" id="2" set_attribute="location" value="main−track" />
<instance type="MD" id="2" set_attribute="shift" value="3PM−−11PM" />
<instance type="MD" id="3" set_attribute="location" value="main−track" />
<instance type="MD" id="3" set_attribute="shift" value="11PM−−7AM" />
```

Fig. 8. The `MD` resource model, specifying attributes, capabilities, and allocation policies

`CT room` resources. Human resource attributes include their work shift and location (fast track or main track), and `bed` attributes include a location.

Every capability of every `MD` and `RN` resource has a `guard` that specifies when the resource is available to provide the capability. One use of this guard is specifying that fast track doctors are only available for fast track patients and only during their shifts.

In our ED model, `MD` and `RN` resources are always reserved before being assigned (whereas our language also allows for assignment without reservation). When a shift change occurs, a reserved resource becomes unavailable, prompting ROMEO to reserve a replacement. This approach enforces both same `MD`-`RN` constraints, and ED shift change policies. A similar approach allows for fast track and main track `bed` resource allocation policies.



(a) Sickest-first scheduling policy (b) Mixed-priority scheduling policy

Fig. 9. Using the mixed-priority scheduling policy greatly reduces the LoS for patients across all acuity levels

5.4 Simulation Results

Effective ED management has multiple competing goals, including decreasing patients' LoS, increasing net revenue, and increasing service quality. An ED can decrease patients' LoS by hiring more staff and building more facilities; but this increases cost and reduces net revenue. Thus, EDs seek resource allocation approaches that decrease patients' LoS without increasing cost or sacrificing quality. While LoS and revenue are straightforward to measure, there are multiple ways to measure quality of care. Because handoffs can cause miscommunication we use the number of patient handoffs due to shift changes to measure care quality.

We next present the results of three ED simulation studies that demonstrate how our approaches and tools can expedite exploring the effects of changing resource mixes and allocation strategies on LoS, resource utilization levels, and number of handoffs.

> Situation 1: The ED wishes to explore how different resource allocation policies affect the patients' LoS.

Our domain expert wanted to explore the consequences of changing the sickest-first policy to treat acuity levels 1–4 as equal. Figure 9 shows how the two policies affect the LoS. Figure 9(a) shows that with the sickest-first policy, the average LoS for all patients is 388 minutes. Low acuity patients experience high LoS variation because they are resource-starved when there are many sicker patients. Figure 9(b) shows that with the mixed-priority policy, overall LoS is decreased to 275 minutes, and the starvation problem for low acuity patients is ameliorated. Our approach expedited this study by requiring only a simple modification to the resource specification (`contention_policy = "MixedPriority : ProblemSpecific"`) and a simple definition of the policy.

> Situation 2: ED beds fill quickly and waiting time increases when patient arrival rates increase. The ED seeks guidance about what investments are likely to be most cost-effective in reducing waiting time.

EDs routinely deal with the problem of insufficient numbers of beds by creating *hallway beds*, beds placed in the ED hallways. This is a low-cost, temporary solution, but

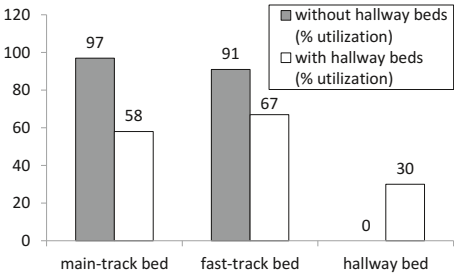


Fig. 10. Allowing the creation of hallway beds greatly reduces the bed resource utilization

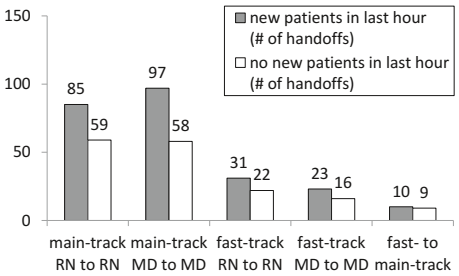


Fig. 11. The number of handoffs decreases when doctors and nurses stop accepting new patients 1 hour before their shifts end

hallway beds cause problems of privacy, noise, and poor traffic flow. We modeled the allocation of hallway beds specifying that they are only allocatable to stable patients, and only when the ED is sufficiently crowded. Figure 10 shows how bed utilization decreases dramatically when hallway beds are added. Hallway beds decrease the waiting times for both tracks, with overall patient LoS dropping from 388 to 159 minutes. Our approach expedited this study by requiring only a modest change to a guard in the specification of the hallway bed resource.

> Situation 3: The ED recognizes that handoffs are error-prone and seeks a strategy for reducing handoffs.

Our domain expert suggested that if an `MD` does not accept *new* patients in the last hour of the shift, that time might be used for more careful handoffs, leading to better quality of care, but increasing patient LoS. Figure 11 compares the number of handoffs in each track when `MD`s do and do not accept new patients in the last hour of their shifts. The main track handoffs reduce significantly, while the fast track improvement is relatively small. However, this changed working policy causes an increase in overall LoS from 159 to 170 minutes. Our approach expedited this study, again by requiring only a modest change to a guard (`reservation_guard = time >= start(shift) && time < end(shift)-1`) for the `MD` and `RN` resources.

5.5 Discussion

Complex domains like hospital EDs benefit substantially from systems that enforce careful resource management and respect resource complexity and constraints. The design of such systems can be suggested by analysis of simulations that reflect accurately and precisely the effects of resource policies on overall system characteristics.

Our work has shown that flexibility in resource specification and management, facilitated by its implementation as a separate architectural component, can expedite the rapid development and evaluation of simulations that can serve as prototypes of such systems. Our approach and tools helped us to model behavior such as shift changes and handoffs easily. We showed that easy modifications to resource specifications sufficed

to support creating simulations that compared resource scheduling strategies, and allocation criteria, and that introduced new resources types and uses. The architecture of our simulation approach facilitated these comparisons by concentrating needed changes in the resource and constraint management components.

6 Related Work

Simulating less-detailed resource models is common, exploring domains such as software development [1, 7, 11], healthcare [2, 4, 10] and other domains that employ intricate processes. Some discrete-event simulation approaches offer enhanced flexibility in defining process event flow [5, 18]. Others have built distributed discrete-event simulators [6, 17] to exploit the power of a distributed environment, but have not focused on the resources.

System dynamics approaches [7,8] incorporate resource issues more prominently, integrating representations of both discrete and continuous dynamics into discrete-event simulations. However, even these approaches fail to represent humans behavior in sufficient detail. Incorporating human behavior in software development, Hanne et al. [16] discuss how human factors influence software development productivity, but focuses on the relation between productivity and learning, time pressure, and other psychological factors. However, their work considers these human factors only as stochastic variables. Lee et al. [9] propose a simulation framework with resource management modules for resource intensive service and business modeling, but their resource models are simplistic, and do not address how system context can dynamically change a resource's capabilities and allocation constraints.

7 Contributions and Future Work

We have developed a resource specification language that allows for precise specification of the capabilities of complex resources, their interactions with one another and with processes that use them, their allocation policies, and their scheduling constraints. We evaluated our language by specifying, in considerable detail, a hospital emergency department's patient handling software component that tracks patients and assigns resources. This evaluation showed that our language is expressive enough to describe complex resources, such as humans, restrictions on their use, and constraints on their allocation policies. Our work suggests that discrete event simulations based on careful resource specification can expedite the design of complex human-intensive systems.

Encouraged by these results, we next will explore (1) developing measures of the reliability of these simulations, (2) identifying static analysis approaches to validating the correctness of these simulations, and (3) measuring flexibility and implementation speed gains deriving from separating resource management from activity management.

Acknowledgments. Alexander Wise helped in developing and supporting various Little-JIL tools. This material is based upon work supported by the National Science Foundation under grants IIS-1239334, CNS-1258588, and IIS-0705772.

References

1. Alan, M.J.S., Christie, M.: Organizational and social simulation of a software requirements development process. Software Process: Improvement and Practice 5, 103–110 (2000)
2. Beck, E.: A discrete event simulation approach to resource management, process changes and task prioritization in emergency departments. Master's thesis, Department of Mechanical and Industrial Engineering, University of Massachusetts, Amherst, MA, USA (2009)
3. Clarke, L., Gaitenby, A., Gyllstrom, D., Katsh, E., Marzilli, M., Osterweil, L.J., Sondheimer, N.K., Wing, L., Wise, A., Rainey, D.: A process-driven tool to support online dispute resolution. In: dg.o, pp. 356–357 (2006)
4. Duguay, C., Chetouane, F.: Modeling and improving emergency department systems using discrete event simulation. Simulation 83(3), 311–320 (2007)
5. Hubl, A.: Flexible model for analyzing production systems with discrete event simulation. In: WSC, pp. 1554–1565 (2011)
6. Jacobs, P.H.M., Lang, N.A., Verbraeck, A.: Web-based simulation 1: D-SOL; a distributed Java based discrete event simulation architecture. In: WSC, pp. 793–800 (2002)
7. KeungSik Choi, T.K.: Doo-Hwan Bae. An approach to a hybrid software process simulation using the devs formalism. Software Process: Improvement and Practice 11, 373–383 (2006)
8. Kofman, E.: Discrete event simulation of hybrid systems. SIAM Journal on Scientific Computing 25, 1771–1797 (2004)
9. Lee, Y.M., An, L., Bagchi, S., Connors, D., Kapoor, S., Katircioglu, K., Wang, W., Xu, J.: Discrete event simulation modeling of resource planning and service order execution for service businesses. In: WSC, pp. 2227–2233 (2007)
10. McCarthy, M.L., Ding, R., Pines, J.M., Zeger, S.L.: Comparison of methods for measuring crowding and its effects on length of stay in the emergency department. Academic Emergency Medicine 18(12), 1269–1277 (2011)
11. Podnar, I., Mikac, B.: Software maintenance process analysis using discrete-event simulation. In: CSMR (2001)
12. Raunak, M., Osterweil, L.: Resource management for complex and dynamic environments. IEEE Transactions on Software Engineering 99 (2012)
13. Raunak, M.S., Chen, B., Elssamadisy, A., Clarke, L.A., Osterweil, L.J.: Definition and analysis of election processes. In: SPW/ProSim 2006, vol. 3966, pp. 178–185 (2006)
14. Raunak, M.S., Osterweil, L.J., Wise, A., Clarke, L.A., Henneman, P.L.: Simulating patient flow through an emergency department using process-driven discrete event simulation. In: SEHC (2009)
15. Shin, S.Y., Balasubramanian, H., Brun, Y., Henneman, P.L., Osterweil, L.J.: Resource scheduling through resource-aware simulation of emergency departments. In: SEHC, pp. 64–70 (May 2013)
16. Hanne, H.N.T.: Simulating human resources in software development processes. Technical Report 64, Fraunhofer-Institut fur Techno- und Wirtschaftsmathematik (2004)
17. Vanmechelen, K., De Munck, S., Broeckhove, J.: Conservative distributed discrete event simulation on Amazon EC2. In: CCGrid, pp. 853–860 (2012)
18. Wagner, G.: Extending discrete event simulation by adding an activity concept for business process modeling and simulation. In: WSC, pp. 2951–2962 (2009)
19. Wise, A.: Little-JIL 1.5 language report. Technical Report 2006–051, Department of Computer Science, University of Massachusetts, Amherst (2006)
20. Zhao, X., Brun, Y., Osterweil, L.J.: Supporting process undo and redo in software engineering decision making. In: ICSSP, pp. 56–60 (May 2013)
21. Zhao, X., Osterweil, L.J.: An approach to modeling and supporting the rework process in refactoring. In: ICSSP, pp. 110–119 (2012)