Specification and Analysis of Human-Intensive System Resource-Utilization Policies

Seung Yeob Shin Yuriy Brun Leon J. Osterweil
College of Information and Computer Sciences
University of Massachusetts
Amherst, MA, 01003, USA
{shin, brun, ljo}@cs.umass.edu

ABSTRACT

Complex, human-intensive systems, such as those used in hospital Emergency Departments, typically require the effective support of many types of resources, each governed by potentially complex utilization policies. Resource utilization policies range from simple, e.g., sickest patient first, to extremely complex, responding to changes in system environment, state, and stimuli. Further, policies may at times conflict with each other, requiring conflict resolution strategies that further increase the complexity. Sound policies for the management of these resources are crucial in assuring that these systems achieve their key goals.

To help system developers make sound resource management decisions, this paper presents a resource utilization policy specification and analysis framework for complex human-intensive systems. We provide (1) a precise specification language to describe very diverse and potentially complex resource utilization policies, (2) a process- and resource-aware discrete-event simulation engine that executes simulations to dynamically evaluate the policies' effects on the outcomes achieved by systems that use the resources, and (3) a process- and resource-aware finite state verification framework that supports formal verification that resource management policies are correctly implemented by these simulations.

CCS Concepts

•Computing methodologies → Model verification and validation; •Software and its engineering → System modeling languages;

Keywords

Resource specification, Discrete-event simulation, Finite state verification

1. INTRODUCTION

Human-intensive systems, where human, software, and hardware resources must be synergistically integrated to perform key functions, play an important role in our society. Because access to these resources is usually limited both by their small quantity and by restrictions on their availability, contention for them is often a serious problem. The problem is addressed by creating policies that are driven by system goals, regulations, or the need to satisfy the interests of different stakeholders. Resource utilization policies usually significantly impact system behaviors and results. Thus, for example, a hospital's staffing policy influences staff workload, costs, quality of patient care, etc. Therefore, resource utilization policies should be thoroughly evaluated and rigorously analyzed.

In addition to the inherent complexity of some policies, an additional major challenge in analyzing resource utilization policies is that policies can conflict with each other. When such conflicts arise in a hospital system progress may stall, and life-threatening situations may arise. Ideally such conflicts should be anticipated so that if they arise smooth operation of the system can nevertheless continue. However, as system size and complexity grow policies tend to grow as well, both in number and in complexity. This increases the likelihood of conflicts and the difficulty of anticipating and resolving all of them.

This paper presents a framework for resource utilization policy specification and analysis for human-intensive systems. This framework extends an iterative process improvement framework [1] by focusing on resource utilization improvement. The specification approach separates resource utilization policy concerns from other system issues such as activity and artifact specification. We propose three types of policy specifications: permission constraint policies that restrict the activities that a given resource can support, scheduling policies to define precedence among both requests and resources, and conflict resolution policies to resolve conflicts between multiple resource utilization policies. In previous work [10,11], we described a process- and resource-aware discrete-event simulator that adheres to resource utilization policy specifications to support dynamic analysis of the effects of diverse resource utilization policies. That work incorporated rudimentary facilities for specifying policy conflicts and their resolution. The contribution of this paper is that it presents a comprehensive approach to policy conflict specification and resolution and also present a process- and resource-aware finite state verification technique that supports static analysis of resource utilization policies and their potential conflicts. Specifically, our static resource analyzer is able to provide guarantees of the absence of violations of resource utilization policies as well as other general resource utilization properties related to resource capacity, deadlock, and starvation.

The rest of this paper is structured as follows. Section 2 motivates the need for our resource utilization policy specification and analysis framework in a hospital's patient care system. Section 3 describes our approach. Section 4 places our work in the context of related research. Section 5 summarizes our contributions and future research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SEHS'16, May 14–15 2016, Austin, TX, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 978-1-4503-4168-4/16/05. . . $15.00

DOI: http://dx.doi.org/10.1145/2897683.2897688

2. RESOURCES IN A HOSPITAL

Assuring sound management of the diverse kinds of human and other resources in a hospital Emergency Department (ED) is complex, and made more so by the need for policies that must be flexible enough to deal with many different kinds of circumstances. Thus, for example, nurses may perform technicians' activities such as Electrocardiogram (EKG) testing in crowded situations to alleviate the heavy workload on technicians, whereas these activities are only performed by technicians under normal circumstances. In addition, hospital staff can care for patients only during their shift hours. However, their shift hours may be changed if the hospital decides to make scheduling policy changes, for example to prepare for increased incidence of accidents during a vacation season. Therefore, these dynamic changes in resources' system participation must be carefully considered to assure efficient and effective resource management.

In addition, the participation in patient care of hospital staff is restricted by various resource utilization policies that often conflict each other. For instance, a patient in a hospital ED should be cared for by the same doctor and nurse while the patient stays in the ED. Under unusual circumstances, however, a hospital may allow violation of this policy to improve efficiency in patient care. For instance, if a high-skilled nurse is too busy to care for his non-severe patients, he may be allowed to delegate his work (e.g., monitoring or discharge) to another nurse even though this delegation policy conflicts with the same nurse policy. Similarly, policies that define work hour shifts may also lead to conflicts with the same doctor policy, for example when a patient must be discharged after the end of the shift of the admitting doctor. To address this policy conflict, hospitals necessarily incorporate a handoff policy requiring that a departing doctor hands over her patients to an incoming doctor. This handoff policy becomes even more complex when taking into account a doctor's workload, length of patient stay (LOS), and other patient care quality measurements.

Suboptimal policies for resolving hospital resource management conflicts can result in such problems as overcrowding, inefficient staff utilization, very long LOS, and other medical or financial problems. To address these problems, hospitals are always seeking better resource utilization policies. We believe our approach supports such efforts by facilitating (1) the exploration of various changes in resource utilization policies through the concentration of resource utilization policy issues into a separate component containing all resource allocation and conflict resolution specifications, (2) the use of discrete-event simulation to evaluate the effects of diverse alternative resource utilization policies, and (3) the verification of the adherence of these simulations to specified resource utilization and conflict resolution policies.

3. RESOURCE UTILIZATION POLICY SPECIFICATION AND ANALYSIS

We now describe our approach to helping system developers devise sound resource utilization policies. We provide a language that addresses the challenges of creating precise and detailed specifications of resource utilization policies. Given the language, we present tools that use both discrete-event simulation and finite state verification to support using both dynamic and static analysis to determine the effects of the specified policies.

3.1 Resource Utilization Policy Specification

Our resource utilization policy specifications are built atop specifications of the resources to be governed by these policy specifications. Our resource specification language has been described in earlier work, and so in this paper we only suggest its specification capabilities through a small pictorial example. Full details of this language can be found in [11]. Thus, note that Figure 1(a) contains a representation of our definition of the `MD` resource type (a doctor working in the ED). Figure 1(b) also indicates how this resource type can be instantiated into three doctor resource instances, namely `md0`, `md1` and `md2` representing the three doctors all working in the same ED. The attributes `shiftStart` and `shiftEnd` represent the `MD`'s shift work hours (e.g., `md0` works from 0AM to 8AM). `Capacity` 1 restricts an `MD` to care for only one patient at a time. The specification further indicates that `MD` can provide the `Treat Patient` capability to care for a patient, but note that the `Treat Patient` specification could be decomposed further into such more specific patient care activities as assess, prescribe, and discharge, as needed by the specific details of more detailed patient processes that may need to be modeled and simulated.

Resource	Attribute	Capacity	Capability
MD	shiftStart shiftEnd	1	Treat Patient

(a) Resource characteristics

MD instance	Attribute value
md0	shiftStart:0AM, shiftEnd:8AM
md1	shiftStart:8AM, shiftEnd:4PM
md2	shiftStart:4PM, shiftEnd:0AM

(b) Resource instances

Permission	Request	Resource	Guard
SameMD	Treat Patient	MD	SameGuard
ShiftMD	Treat Patient	MD	ShiftGuard

(c) Permission policies

Figure 1: Examples of resource characteristics (a), resource instances (b), and permission constraints (c) in a hospital ED. For exposition, we omit full specifications, and include only the key features specifications.

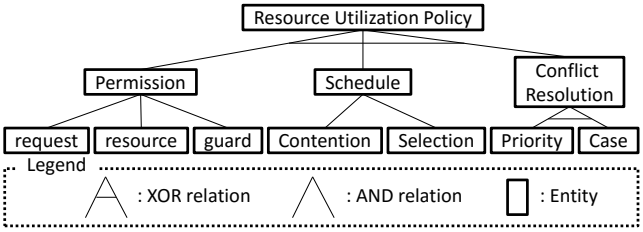

Figure 2: The structure of the resource utilization policy specification describes the static relations between the policy's entities.

Given the existence of such resource definitions, we now present some example resource utilization policies. Figure 2, shows that we currently provide facilities for the specification of three different kinds of resource utilization policies, namely permission constraint policies, scheduling policies, and conflict resolution policies, which we now describe in turn.

3.1.1 Permission Policies

A `Permission` policy specifies the permissibility of a `Resource`

to handle a Request as restricted by a specified Guard condition. Thus, for example, the SameMD and ShiftMD permission policies can be specified as shown in Figure 1(c). The SameMD permission policy is used to determine which MD resource instance is available to handle a request requiring the Treat Patient capability. Similarly, the ShiftMD permission policy is used to define the physician's shift, namely the hours during which an MD resource is allowed to handle Treat Patient requests. Guard is represented as a boolean expression that is evaluated to be true only when a resource instance can handle a request. For instance, we specify ShiftGuard as $t >= r.shiftStart \;\&\&\; t < r.shiftEnd$ where r is one of the MD resource instances, md0, md1, or md2. The boolean expression is evaluated to be true only when an MD resource instance r is working on her shift when a request Treat Patient happens at time t.

3.1.2 Schedule Policies

The next policy specification approach, the Schedule policy, supports the specification of Contention and Selection policies. Contention policies specify precedence among requests. Specifically, we support specification of built-in policies such as random, first-in first-out (FIFO), last-in first-out (LIFO), and priority-based; and custom-built policies that can be built based on the use of dynamic system state variables, such as patient load. In a hospital ED, for instance, when multiple patient care activities for different patients require the service of more doctors than are currently available for allocation, an appropriate scheduling policy is necessary to resolve the contention problem among the requests for the services of all doctor resources. As can be seen in the following Contention policy example, the sickest patient first policy is a contention policy that is often used in a hospital ED, specifying that a doctor cares for patients according to the urgency of their need, patient.severityLevel. The effects of the contention policy should be carefully evaluated, however, as enforcing it rigorously can result in the needs of very sick patents to cause resource starvation of non-acutely ill patients who might then fail to receive timely treatment. Simulation studies should facilitate such evaluations.

Contention policy: SickestPatientFirst
Request = {Treat Patient},
Contention Policy = (Priority, patient.severityLevel)

The Selection policy complements the Contention policy, supporting specification of precedence among the resources that are able to handle a resource request. If there are many candidate resources able to handle a given request, an effective selection policy can lead to more efficient and effective utilization of those resources. For instance, when a new patient arrives in an ED, there are usually more than two doctors who can assess the patient. An appropriate workload policy can do much to balance the workloads of the different doctors. We provide built-in selection policies such as random, least recently used (LRU) and most recently used (MRU); and custom policies that can be defined based on the use of system dynamic state variables. Thus, for example, a least utilized resource first policy might be used to balance the workloads of hospital staff members. As can be seen in the following example, LeastUtilizedFirst is a custom policy that uses such system state variables as the system's execution duration and resource allocation periods, to calculate a level of resource utilization that might then be used as the basis for allocating the least utilized resource first.

Selection policy: LeastUtilizedFirst
Request = {Treat Patient},
SelectionPolicy = LeastUtilizedFirst::CustomBuilt

3.1.3 Conflict Resolution Policies

Conflict resolution policies specify how to deal with other policies that may come into conflict with each other. For the circumstances under which two or more policies cannot be enforced at the same time we provide capabilities for specifying Conflict Resolution policies. We provide two kinds of conflict resolution policies: a Priority conflict resolution policy and a Case conflict resolution policy. We now describe each of them.

3.1.3.1 Priority Conflict Resolution.

The ED policy that we described above, specifying how doctors hand off patients at the end of their shifts, can be defined using the following Priority conflict resolution policy:

Priority conflict resolution policy: HandoffMD
{SameMD, ShiftMD} > {ShiftMD}

This conflict resolution policy, {SameMD, ShiftMD} > {ShiftMD}, specifies that enforcing both the SameMD and ShiftMD permission policies has higher priority than enforcing only the ShiftMD permission policy. In other words, this states that the need to satisfy both SameMD and ShiftMD is considered first in doing resource allocation. However, if no resource is able to satisfy all the permission policies, the permission policy ShiftMD is applied next in considering possible resource allocations. For instance, if md0 ends her shift at 8PM, then md0 is not able to care for her patients at 8PM because that would violate the ShiftMD policy. According to the above priority specification, however, another doctor, either md1 or md2 is then permitted to provide care for the patients of md0, assuming these other doctors are on shift in the ED (i.e. assigning them must still satisfy the ShiftMD policy). After resolving such a conflict condition, the system returns to full enforcement of all policies for subsequent resource allocation requests. Thus, for example, either md1 or md2 (whichever has taken the handoff) is obliged to satisfy the same doctor policy until her shift ends.

This kind of conflict resolution specification seems to have broad applicability. We note, for example, that the use of more highly skilled medical providers is generally preferred because it typically results in the provision of higher quality patient care. But less skilled providers may be allocated to these tasks instead in order to improve the overall efficiency of the healthcare system in providing patient care in various kinds of complex situations. Moreover, as described in Section 2, in a particularly overcrowded condition, nurses may perform technician tasks such as taking EKGs. Conversely, even though nurses may be the preferred providers of certain kinds of patient care, a technician-use policy might dictate that technicians be allowed to provide these kinds of care to certain classes of patients when nurses are not available, perhaps due to nurse workload limit policies. Similarly, the above priority policy allows the delegation of a doctor's work to another doctor when the first doctor's shift ends.

3.1.3.2 Case Conflict Resolution.

In addition to Priority, we provide another form of specification for conflict resolution, namely the Case conflict resolution policy. While Priority seems more suitable for specifying relatively straightforward conflict resolution, Case seems more suitable for complex conflict resolution situations. Further experimentation is underway to validate this hypothesis.

A Case conflict resolution policy specification is defined as an ordered pair, (conflict situation, resolution action). Thus, as an example, consider the FastTrackBedUse policy.

Case conflict resolution policy: FastTrackBedUse
Conflict situation:
`EmptyBed(√), InFastTrack(×), FastTrackOpen(×)`
Resolution action:
`enforce {EmptyBed}`

Many hospital EDs offer their patients care in two separate locations, with separate facilities, and using somewhat different processes. These are often referred to as the main-track (for seriously ill patients) and the fast-track (for all other patients), and they are operated separately in order to improve their overall efficiency and quality in providing ED patient care. While the main-track operates 24 hours of every day, the fast-track is often closed during periods of slack demand, such as late at night, in order to save money. If the locations of the two tracks are not far from each other, hospitals often, especially during periods of heavy load, find it useful to utilize fast-track beds, even if the fast-track is closed. Although fast-track beds may be used in this way, hospitals will not use any other fast-track resources when the fast-track is closed. This policy can be be defined clearly and concisely using the `Case` policy specification, `FastTrackBedUse`. We do this by building upon some lower-level policies and combining them using the `Case` conflict resolution specification approach. Thus, we use the `EmptyBed` policy, which enforces that a bed must not already be occupied in order for it to be used to care for a new patient. We use the `InFastTrack` policy, which specifies that only fast-track patients are allowed to use beds in the fast-track. And we use the `FastTrackOpen` policy, which specifies the times during which the fast-track is open and closed. Given these three policies, we can specify the typical use of a fast-track ED bed, when it is allocated to care for a new fast-track patient in an open fast-track to be: only when a bed is not occupied (`EmptyBed(√)`), the bed and patient are located in the fast-track (`InFastTrack(√)`), and the fast-track is open `FastTrackOpen(√)`.

But a `Case` conflict resolution policy specification can, in addition, support the specification of the previously described conditions under which a fast-track bed can be allocated for main-track use even when the fast-track is closed. This `Conflict situation` as described above, is defined as `Conflict situation: EmptyBed(√), InFastTrack(×), FastTrackOpen(×)`, which defines the situation in which a bed is not occupied, a patient is not located in the fast-track, and the fast-track is closed. It then specifies the `Resolution action` as described by, `enforce {EmptyBed}`, which specifies that it is allowable to enforce only the `EmptyBed` policy to resolve the conflict situation (note that another conflict resolution policy might be defined to enforce a preference for using a main-track bed over a fast-track bed, when both are available). Therefore, this conflict resolution policy specifies a flexible management of hospital bed resources by utilizing fast-track beds when the fast-track is closed. Note in addition that there is no specified resolution for conflicts over non-bed resources when enforcing the three policies: `EmptyBed`, `InFastTrack`, and `FastTrackOpen`. Thus this conflict resolution, as desired by ED policy, does not allow the allocation of fast-track resources other than beds.

3.2 Discrete-Event Simulation

As noted above, our interest in the specification of complex constraints on diverse resources has arisen from our earlier interest in the discrete-event simulation of hospital EDs [3, 10, 11]. This paper only summarizes our previous work briefly to provide motivation for our static analysis research. But it is certainly the case that this simulation work, in addition to motivating our work on resource policy, has also served as an excellent vehicle for demonstrating the effectiveness of this work. In order to support this resource- and

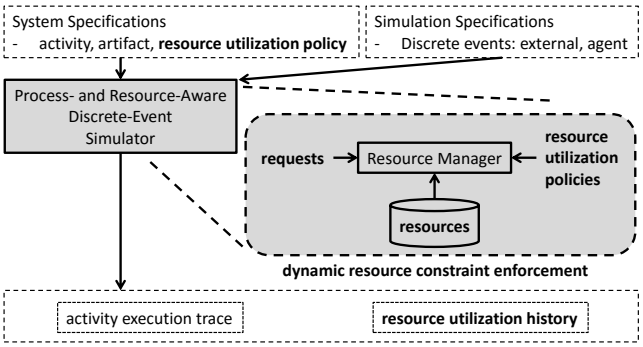

Figure 3: Process- and resource-aware simulator separates the resource manager for dynamically enforcing resource constraints. The simulator extends a prior resource manager [11] to incorporate resource utilization policies.

resource-conflict- sensitive discrete-event simulation, we require a specification of a process within which our specified resources are to be allocated and utilized. Thus our work assumes the existence of a process definition that defines system aspects such as activity coordination and data flow as well as specifications of the kinds of resources that are required at different points in the performance of the process. In our work we have used the Little-JIL process definition language [14] to specify patient care processes in a hospital ED. A Little-JIL specification is a graphical, hierarchical decomposition of activities that provide rich semantics that support diverse sequencing of activity executions and data flows. The rich semantics of Little-JIL have proven to be very useful in supporting the capture of the complex nature of ED processes. We assume that the execution, either real or simulated, of a Little-JIL-defined process generates a stream of requests for the allocation of the resources needed to perform specified activities. We built our ED process models based on real-world data and the knowledge of a domain expert who has extensive experience as an emergency department physician and ED manager at the Baystate Medical Center, in Springfield, MA, USA. The full ED process model that we used in our work is publicly available at http://people.cs.umass.edu/~shin/ed/.

We used the JSim [8] simulator as a discrete-event simulation engine, augmenting it beyond the capabilities described in [11] to additionally incorporate the resource utilization policies and policy conflict specifications that we have just described. As can be seen in Figure 3, our simulator took `System` and `Simulation` specifications that included some `resource utilization policy` specifications as well. In our simulator, the enforcement of the resource utilization policies is concentrated into the `Resource Manager` component, which facilitates keeping track of resource utilization history such as resource allocation traces, counts of policy conflicts, wait times to allocate resources, and other resource related simulation results.

3.2.1 Dynamic Verification

Our prior work [3, 10, 11] described a variety of simulation studies that we carried out using the earlier version of our simulator. Here we present only one, again presenting this example study largely to provide the basis for supporting the presentation of our work on static analysis of resource specifications.

In many studies of patient care processes [3, 8, 12], bed time, the total time that a patient spends in a bed during an ED stay, is an important measurement of efficiency and effectiveness in patient care. Bed time is influenced by many factors, such as the total

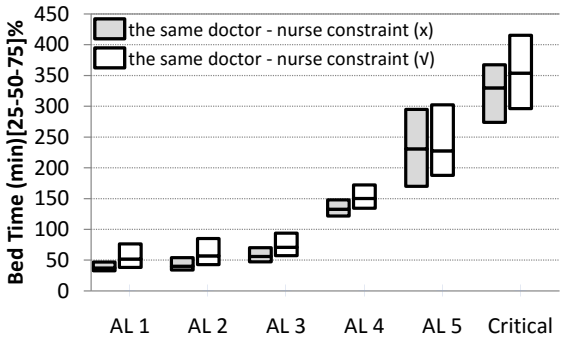

Figure 4: Two simulations showing bed-time results. The policies for the gray box plot simulation fail to adhere to the same doctor-nurse constraint, while the white box plot simulation adheres to that constraint.

number of ED beds, the arrival distribution of patients at an ED at various times, the nature of the patient care processes themselves, and the ways that resources are constrained by their utilization policies. To study bed time, our ED model includes the following important ED characteristics: (1) varying patient arrivals over a 24 hour day; (2) six different patient care processes, one for each ED patient acuity level; (3) diverse kinds of resources such as beds, medical devices (e.g., x-ray and CT machines), and human resources (e.g., doctors and nurses); and (4) resource utilization policies such as the same doctor-nurse, shift, and handoff policies.

Figure 4 compares the results of two simulation studies, showing simulated bed times for patients in each of the EDs six acuity levels. The gray box plot shows the bed times obtained from simulations that were not required to adhere to the same doctor-nurse policy. In contrast, the white box plot shows the bed times for simulations that did adhere to the same doctor-nurse policy. As can be seen, the two simulations provide significantly different bed time results, especially for the `Critical` patients. As might be expected, one key source of observed differences was the amount of time that patients had to spend waiting to be cared for by doctors and nurses. In particular, when a patient is ready to be discharged, if her doctor is busy caring for another patient, the patient must wait in her bed because of the same doctor policy. This type of result is but one of many that provide a variety of suggestions about policies that ED administrators should consider, based upon studies such as ours. However, hospital administrators can be quite skeptical about simulation results, partly because it is difficult to be sure that simulation results are correct. Discrete-event simulations are complex programs that pose the usual difficulties for testing. Thus, for instance, if a violation of the same doctor-nurse policy happens in only one specific patient care trace out of a plethora of possible simulation traces, it requires herculean effort to search for and identify this rare trace. Indeed, such a problem trace may not even be executed in some simulations. More fundamentally, because simulations are intended to produce results that cannot be obtained any other way, the correct results of a simulation run are not known in advance (there is no oracle for these programs), making it difficult to verify the quantitative results of a discrete event simulation run with traditional testing approaches. These difficulties in the dynamic testing of simulations, and especially in verifying that simulations always correctly enforce resource utilization policies, and policy conflict specifications, has caused us to explore the use of static analysis approaches to support such verifications.

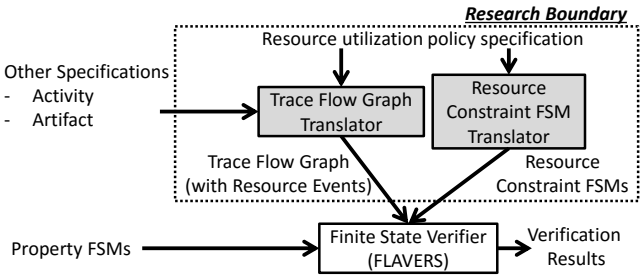

Figure 5: Our static resource analysis framework relies on FLAVERS, a finite state verification tool. The approach automatically generates the FLAVERS inputs that encode resource utilization policies.

3.3 Finite State Verification

Finite State Verification is a static analysis technology that has long been used in order to either demonstrate that all possible executions of a program must always adhere to specified properties, or to identify one or more paths on which a property can be violated. In this past work the properties were typically event sequence specifications characterizing program functionality. In this current work we have used this technology to statically verify that all simulation executions must always adhere to resource utilization specifications defined using the policy specifications presented above. Our belief is that such verifications should increase the credibility of the results of our simulation studies.

Our static analysis of resource utilization relies on the use of FLAVERS [2], a static analysis tool that can verify that all executions of a system satisfy a given property, or will report a counterexample scenario that violates the property. FLAVERS requires three kinds of inputs: a trace flow graph (TFG), a property finite state machine (FSM), and a set of constraint FSMs. A TFG is a conservative representation of all syntactically-feasible event execution sequences of a system, automatically derived from our Little-JIL ED process specification. A property FSM is a representation of a mechanism for accepting only desired system event execution sequences. A property FSM has special accepting states that define what is meant by successful satisfaction of a property. A constraint FSM is a representation of a mechanism for identifying when a system event execution sequence has become infeasible. The constraint FSM has special trap states to eliminate infeasible execution paths from the analysis search space.

As can be seen in Figure 5, our static resource analysis approach extends the `Trace Flow Graph Translator` and `Resource Constraint FSM Translator` components that generate the inputs to the FLAVERS `Finite State Verifier`. The TFG extensions have the effect of augmenting this graph with representations of resource allocation events and resource utilization policies in addition to the TFG's existing specifications of activities and artifact flows. Augmentations to the `Resource Constraint FSM Translator` convert the various resource policy specifications into constraint FSMs that are used to exclude infeasible resource allocation behaviors from the analysis search space.

As an example, we demonstrate how we use this approach to verify that simulations always adhere to the same doctor property. First note that Figure 6 shows a part of generated TFG that represents the allocation and deallocation of a doctor resource. We assume that there are two doctors that might be allocated, and thus show that either one or the other is allocated prior to execution of the `assess` patient care task, and either one or the other is deallocated

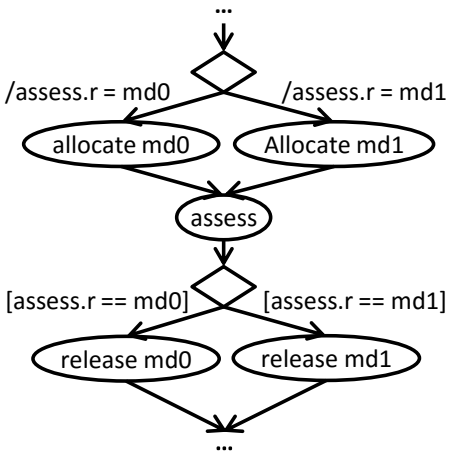

Figure 6: A part of the generated trace flow graph, including resource allocation and resource release events for the md0 and md1 resource instances.

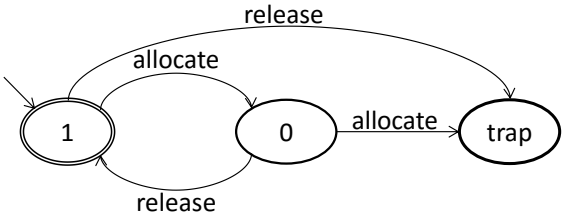

Figure 7: This resource constraint FSM encodes the capacity limit of resource allocations. This FSM allows a resource to be allocated at most once.

afterwards. Accordingly, two kinds of resource events (`allocate` and `release`) for two MD resource instances (`md0` and `md1`) are included in the TFG, before and after the `assess` event node representing the choices between the two `MD`'s. Because the TFG is a conservative representation it includes infeasible paths such as `allocate md0` → `assess` → `release md1`. These infeasible resource allocation sequences are eliminated by associating variable update actions or guards to transition edges in the TFG. For instance, if the `allocate md0` event happens on an execution path, the action `/assess.r = md0` updates the `assess.r` variable. Then, the the guard `[assess.r == md0]` causes only the following event trace to be feasible: `allocate md0` → `assess` → `release md0`.

With resource allocation events embedded in the TFG as just shown (and with the additional assurance that only allocated resources will be deallocated) it is now possible to create the basis for verifying that no resource can ever be allocated to more than one task at a time (as mandated by the specification in Figure 1(a) in Section 3.1, which shows that an `MD` resource is only allowed to care for one patient at a time (see `Capacity` 1)). Given this resource capacity policy specification, `Resource Constraint FSM Translator` creates the appropriate resource constraint FSMs. Figure 7 shows such a constraint FSM encoding the capacity limit constraint for the `MD`. This FSM specifies that if an `MD` resource is allocated twice, without an intervening deallocation, the FSM will be driven into the `trap` state, indicating the presence of a path along which the capacity constraint has been violated. If the trap state is entered, then FLAVERS excludes the path causing this infeasible resource allocation behavior from the analysis search space.

The last input component of FLAVERS is a property FSM. Figure 8 shows the same doctor property. For FLAVERS verification, the events in the property FSM (e.g., `treat patient`) are matched to the nodes in the TFG that represent patient care activities. For instance, the `reserved` abstract event is matched to activity of having a doctor allocated to begin treating a new patient. For this property we also create `treat patient`, an abstract event consisting of the set of all concrete events, that cause the allocated doctor to perform patient care activities, and `treat by others` an abstract event consisting of the set of events that cause other doctors to deliver care to the patient. The property FSM contains the states that a given `MD` resource can be in. The `idle` state means that the doctor is not in charge of caring for the patient `p`. The `reserved` event indicates that doctor is reserved to care for a patient `p`. After the doctor is reserved to care for the patient `p`, the property FSM allows (`treat patient`) by only the same doctor. If the patient `p` is cared for by another MD, the property FSM reaches the `trap` state which indicates a violation of the same doctor property.

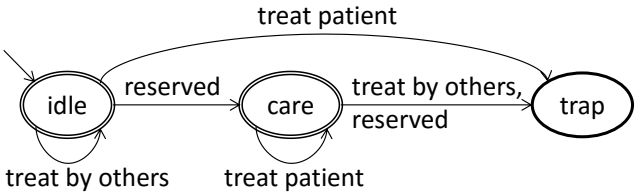

Figure 8: This FSM encodes the same doctor property. The `idle` and `care` states are accepting states. The `trap` state is a violation state.

Given the three kinds of inputs, FLAVERS uses its state propagation algorithm to verify whether all the resource allocation paths of a given TFG always satisfy a property FSM (see Figure 8). The analysis algorithm associates a set of tuples of states in the property FSM and constraint FSMs to each event node in the TFG. While propagating the tuples based on events in the TFG, if a constraint FSM reaches a trap state, the corresponding execution path is excluded from the analysis search space (e.g., the deallocation after a task of a doctor other than the one who had been allocated to do that task) If the property FSM has reached an accepting state at the end of state propagation, the analysis has thus guaranteed that all possible simulation executions must always obey the same doctor constraint.

3.3.1 Preliminary Evaluation

Given the same doctor property in Figure 8, we verified whether the specified patient care processes in an ED satisfy the property or not. For this verification, we used three doctors (`md0`, `md1`, and `md2` all working on the same shift) and one patient. We verified the ED model, incorporating the same doctor policy (see the simulations of gray box plot in Figure 4). The verification was run on a computer with four 3.7GHz processors and 4GB memory. The ED process definition incorporated 178 Little-JIL activity nodes, which in conjunction with our process- and resource-aware policy specifications, resulted in the generation of a TFG with 1517 nodes, 6635 edges, and 11 tasks. 28 constraint FSMs were also generated, aggregating a total of 1601 states and 119714 transitions. This verification required approximately 10 seconds.

The following counterexample (`md2` is caring for a patient until `md1` discharges the patient), was uncovered by our initial analysis, which thus showed that the original ED simulation did contain a doctor allocation trace that violated the same doctor property. After appropriate modifications the ED specifications were then verified

to adhere to the same doctor policy.

Counter example (resource allocations):

```
(allocate md2 to prepare patient) →
(allocate md2 to assess) →
(allocate md2 to reassess) →
(allocate md2 to check x-ray result) →
... →
(allocate md2 to reassess) →
(allocate md1 to discharge)
```

Our evaluation shows that the formal resource utilization policy specifications allow verifying properties related to resource utilization policies, although the initial verifications were for relative small examples. Indeed, verification of processes involving the concurrent care for multiple patients causes search space explosion. As we address this problem we hope to find that this verification capability will help domain experts gain additional confidence in the correctness of the results produced by simulations such as ours.

4. RELATED WORK

Systems enforce policies to guide behavioral decisions of participants to help them achieve system goals. Much prior work has suggested security policy or access control policy languages to specify that only authorized entities are permitted to access entities such as services or resources. Sandhu et al. [9] provide role-based access control models. XACML [7] defines information access control policies for securely browsing documents over the Internet. The Rei [4] policy specification language is based on deontic concepts and provides constructs for rights, prohibitions, obligations and dispensations. However, we have found that most of this prior work has difficulty in specifying diverse and complex resource utilization policies in a flexible manner.

Simulations are widely used to dynamically analyze complex systems in software development, healthcare, manufacturing, and other domains. For instance, SimEvents [6] is a discrete-event simulator built based on Matlab functions. Wang et al. [12] use a simulation model to identify potential changes in operational policies to reduce patients' length of stay. Despite the large number of these previous studies, in our view, none of them has supported the evaluation of complex resource utilization policies in a flexible manner because most of them exclude resources as entities in their activity coordination models.

Static verification techniques are also widely used to complement the inherent limitations of dynamic verification such as the impossibility of exhaustive executions of a system. Li et al. [5] provide a resource constraint analysis approach for workflow specifications. Wang et al. [13] introduce a resource-constrained workflow model as well as a resource requirement analysis approach. In contrast to our static resource analysis, this prior work has focused on only a specific resource utilization problem such as resource contention and capacity limits based on relatively simplified resource models.

5. CONCLUSIONS AND FUTURE WORK

We have presented a framework of resource utilization policy specification and analysis. To incorporate the diverse, complex characteristics of resource utilization policies in a manageable manner, we separate resource utilization policy concerns into permission constraints, schedule, and conflict resolution policies. Given these formal resource utilization policy specifications, we support dynamic analysis of the effects of different resource utilization policies through process- and resource-aware discrete-event simulation. In addition, a process- and resource-aware finite state verification system verifies properties and adherence to resource utilization policies.

Our preliminary results show promise that our framework helps system developers evaluate, validate and verify diverse and complex resource utilization policies in a system. Encouraged by this, we are continuing our research on (1) evaluating our framework by applying it to a complex hospital ED patient care system, (2) validating our simulation models through close comparison between real-world data and simulation outputs, and (3) identifying key system properties in a hospital that needed to be verified. We will continue to investigate the relative efficacies of our different conflict resolution specification approaches. We will continue to investigate the scalability of our static analysis approach, as we have recognized that scaling up to multiple patients, for example, leads to serious state space explosion problems, which we are now addressing.

6. ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under grants no. IIS-1239334, CMMI-1234070, CNS-1513055, and CCF-1453474.

7. REFERENCES

[1] L. A. Clarke, L. J. Osterweil, and G. S. Avrunin. Supporting human-intensive systems. In *FoSER*, 2010.

[2] M. B. Dwyer, L. A. Clarke, J. M. Cobleigh, and G. Naumovich. Flow analysis for verifying properties of concurrent software systems. *TOSEM*, 13(4), 2004.

[3] P. L. Henneman, S. Y. Shin, Y. Brun, H. Balasubramanian, F. Blank, and L. J. Osterweil. Using computer simulation to study nurse-to-patient ratios in an emergency department. *JoNA*, 45(11), 2015.

[4] L. Kagal, T. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *POLICY*, 2003.

[5] H. Li, Y. Yang, and T. Y. Chen. Resource constraints analysis of workflow specifications. *Journal of Systems and Software*, 73(2), 2004.

[6] MathWorks. SimEvents. http://www.mathworks.com/products/simevents.

[7] OASIS Standard. extensible access control markup language (xacml) version 3.0. Technical report, (OASIS), 2013.

[8] M. S. Raunak, L. J. Osterweil, A. Wise, L. A. Clarke, and P. L. Henneman. Simulating patient flow through an emergency department using process-driven discrete event simulation. In *SEHC*, 2009.

[9] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2), 1996.

[10] S. Y. Shin, H. Balasubramanian, Y. Brun, P. L. Henneman, and L. J. Osterweil. Resource scheduling through resource-aware simulation of emergency departments. In *SEHC*, 2013.

[11] S. Y. Shin, Y. Brun, L. J. Osterweil, H. Balasubramanian, and P. L. Henneman. Resource specification for prototyping human-intensive systems. In *FASE*, 2015.

[12] J. Wang, J. Li, K. Tussey, and K. Ross. Reducing length of stay in emergency department: A simulation study at a community hospital. *IEEE Transactions on SMC, Part A: Systems and Humans*, 42(6), 2012.

[13] J. Wang, W. Tepfenhart, D. Rosca, and A. Tsai. Workflow resource requirement modeling and analysist. In *ICNSC*, 2008.

[14] A. Wise. Little-JIL 1.5 language report. Technical Report 2006–051, Department of Computer Science, University of Massachusetts, Amherst, 2006.