Resource Management in Complex, Dynamic Environments

Mohammad S. Raunak[*] Leon J. Osterweil
Department of Computer Science
University of Massachusetts
Amherst, MA
{raunak, ljo}@cs.umss.edu

ABSTRACT

This paper describes an approach to the management of resources. The paper suggests that a resource should be viewed as a provider of a set of capabilities, where that set may vary over time and with circumstances. This view of resources is defined and then made the basis for the architecture of a system for storing, managing, and assigning resources. The ROMEO prototype resource management system is presented as an example of how this architecture can be instantiated. Some case studies of the use of ROMEO are presented and used to evaluate the architecture, the ROMEO prototype, and our view of the nature of resources.

1. INTRODUCTION

The systems that are of increasing importance to society are complex collaborations among such diverse kinds of resources as software systems, hardware devices, and humans. These systems are typically highly concurrent, and highly dynamic, often entailing the real-time identification and acquisition of these resources to support the performance of various system tasks. The most critical of these resources will usually be in short supply, and thus the objects of contention from the tasks comprising the system. An abundance of such resources will improve system performance, but will increase cost. Thus it is important to devise approaches to the judicious specification, management, and allocation of the resources needed by such systems.

This paper presents an overall view of resource specification, management, and assignment, and introduces tools and methods for supporting this view. Much work in many different domains has focused on various restricted resource-related problems, but has lacked the generality and power needed for adequate support of the management of the very diverse kinds of resources needed for highly complex, dynamic systems. This paper uses as an example systems needed in a hospital emergency department (ED). ED resources range from humans such as doctors, nurses, clerks, and patients, to software systems such as Electronic Healthcare Records (EHRs), equipment, such as X-ray machines, and beds, blood, and medicines. The complexity of these resources is complicated by the use of aliasing in describing them. For example, the names *doctor*, *pediatrician*, *surgeon*, *attending MD*, *director*, *primary caregiver* may all be attached to the same individual, sometimes all at the same time, and sometimes only in certain specific contexts. Further complicating the situation, these different names sometimes influence the capabilities that the resource may provide. Context may have a similar effect. For example, a physician's assistant may prescribe medication for a patient with chest pain in an extraordinary situation, whereas usually this would be done only by a doctor. ED resources may also have substitution, preemption, and priority relationships and constraints, often governed by complex policies. The approach presented here addresses all of these issues, thus suggesting how to provide key support for this growing range of societally-essential systems.

[*]The author is currently an Affiliate Assistant Professor at Loyola University Maryland

Section 2 presents our view of the nature of resources, and introduces terminology and notation needed to be precise about these ideas. Section 3 describes a conceptual framework for resource management, and the ROMEO prototype built on this framework. Section 4 describes the experimental setup used for our case studies. Section 5 presents the results of some of these case studies. Section 6 relates our work to other work on resources. Section 7 draws some conclusions and suggests some future research directions.

2. APPROACH

Systems in which resource management is an issue typically function by executing activity sequences and realizing artifact flows that define the ways that system components and capabilities are coordinated. We refer to such specifications as process definitions. Our view is that process definitions must also specify the resources that each activity needs. Thus, for example, the process of taking an X-ray in a hospital ED has a central functional component that takes X-ray film and a patient as inputs and produces the patient's X-ray as output. Clearly this will not happen without using an X-ray machine as a resource. But a precise characterization of what a resource is seems elusive. Note in particular, that it is reasonable that the X-ray film might also be considered a resource, and even the patient might be considered to be a resource to this activity.

It has been observed that, "a resource is an entity for which there is contention". This characterization seems to be useful, but is not without problems. Note, for example, that an X-ray machine may be subject to contention during a busy period, but completely idle, and subject to no contention, at other times. Thus, this definition of a resource leaves it possible that some entities could be considered resources at some times, but not at others. Suggesting that a resource is any entity for which there might ever be contention creates the possibility that even input parameters to functions might have to be considered resources.

The essence of a resource, we argue, is that it is a provider of a set of capabilities, but the set of capabilities that can be provided may be different at different times and under different circumstances. This potential to change the set of capabilities offered seems to us to be inherent in the nature of a resource. Thus, a nurse may not be able to provide the capability to authorize medication under most circumstances, but may indeed be able to do so in an emergency. A hallway wheelchair may not provide the capability of housing an ED patient under ordinary circumstances, but may in an emergency. Thus, each of the capabilities offered by a resource should be thought of as being *guarded* by a specification of when the capability can be provided. A resource is further characterized by a collection of attributes (e.g. name, job title, experience etc.) to be described more fully later. We note that zur Muehlen has a similar view of the nature of a resource, although he does not consider the possibility of dynamic change to the set of offered capabilities [33]. Russell et al. also share this view and further suggest that context may affect which capabilities a resource may offer [25]. Interestingly, we note that the term "resource" is not defined at all in the Workflow Management Coalition's Terminology and Glossary document [31]. We now provide some definitions that are more precise and specific about the intuitive ideas we have just presented.

2.1 Definitions

Let D be some domain of interest, and let P_D be a set of system processes that can be executed in D. Assume T is a finite set of all the different activities that can be carried out in performing any of the processes in P_D. Now suppose that Σ is a specific process in P_D, and let T_Σ be the set of all the different activities included in Σ. For every $t \in T_\Sigma$ there is some finite set of n_t capabilities, $CAPS_t$, required to support performing activity t. For notational simplicity (the needed generalization is not hard to devise, but can be hard to read), we assume each performance of a given activity t in a given process Σ requires the same set of capabilities. Thus, we define

$$CAPS_t = \{CAP_{t_1}, CAP_{t_2}, \cdots, CAP_{t_{n_t}}\}$$

As an example, suppose D is the hospital ED domain, then some of the activities in T_Σ might be "triage incoming patient", "assess patient condition", and "take X-ray". The capabilities needed to support "triage incoming patient" might include triaging, presumably (but perhaps not necessarily) provided by a triage nurse. The capabilities needed to support "assess patient condition" might include those provided by such resources as a doctor, a nurse, a stethoscope, and an ED bed. The capabilities needed to support "take X-ray" might include those provided by an X-ray machine, and some X-ray film. For a process Σ to be performed, it must have associated with it a pool of resources R_Σ each of which has the potential to provide one or more of the capabilities needed by at least one of the activities of the process. Thus every entity $r \in R_\Sigma$ has associated with it a set of n_r capabilities, $CAPS_r$, which is the set of all capabilities that r could ever possibly offer in support of a system process in domain D. Thus,

$$CAPS_r = \{CAP_{r_1}, CAP_{r_2}, \cdots, CAP_{r_{n_r}}\}$$

We use a capability projection function Φ, that uses Σ_{STATE}, a specification of the execution state of a system process Σ, to project $CAPS_r$ onto the subset of $CAPS_r$ that r can actually provide when Σ is in the state of Σ_{STATE} . Thus

$$CAPS_{r,\Sigma_{STATE}} \subseteq CAPS_r \textit{ such that}$$

$$CAP_{r_i} \in CAPS_{r,\Sigma_{STATE}} \textit{ if and only if}$$

$$CAP_{r_i} \in CAPS_r \textit{ and } \Phi(\Sigma_{STATE}, CAP_{r_i}) = \ TRUE$$

And we note that when a system process Σ in domain D is in state Σ_{STATE} , then a resource $r \in R_\Sigma$ can potentially be assigned to support the performance of an activity $t \in T_\Sigma$ if and only if

$$CAPS_{r,\Sigma_{STATE}} \cap CAPS_t \neq \emptyset$$

Thus, for example, "prescribe medication" might be an element of a nurse's $CAPS_r$ set, but Φ might map this capability to $TRUE$ if and only if Σ_{STATE} shows that all doctors are currently unavailable, and the condition of the nurse's patient is critical.

We now define a set of candidate resources that can provide a capability at a system state Σ_{STATE} as:

$$CAND_{t,\Sigma_{STATE}} = \{r \in R_\Sigma | CAPS_{r,\Sigma_{STATE}} \cap CAPS_t \neq \emptyset\}$$

If $CAND_{t,\Sigma_{STATE}}$ is empty, no resource is currently available for assignment, and if $CAND_{t,\Sigma_{STATE}}$ has cardinality > 1, a decision must be made about which candidate resource is to be selected. This decision might be based upon the characteristics of the alternative resources, and so each resource specification includes a set of descriptive attributes. Some example attributes might be *name*, *job title*, *education level*, and *cost*. In addition each resource specification also includes a set of *skill level* and *effort level* attributes that quantify the quality that r achieves in providing each of the capabilities in $CAPS_r$, and the effort required to do so. Most resources do not provide a limitless amount of capability, and so each resource specification also has a *capacity* attribute. If the available *capacity* of a resource is less than the *effort level* for a requested capability, then the resource cannot provide the capability.

We define a resource assignment to be a binding to an executing task of a resource selected to satisfy the task's capability request. Specifically, assume $\Sigma_{ACTIVITIES,STATE}$ is the set of activities being performed when process Σ is in state $STATE$, more precisely assume

$$\Sigma_{ACTIVITIES,STATE} = \{t_{ACT,1}, t_{ACT,2}, \cdots, t_{ACT,n}\}$$

$$\textit{where } n = \mid \Sigma_{ACTIVITIES,STATE} \mid$$

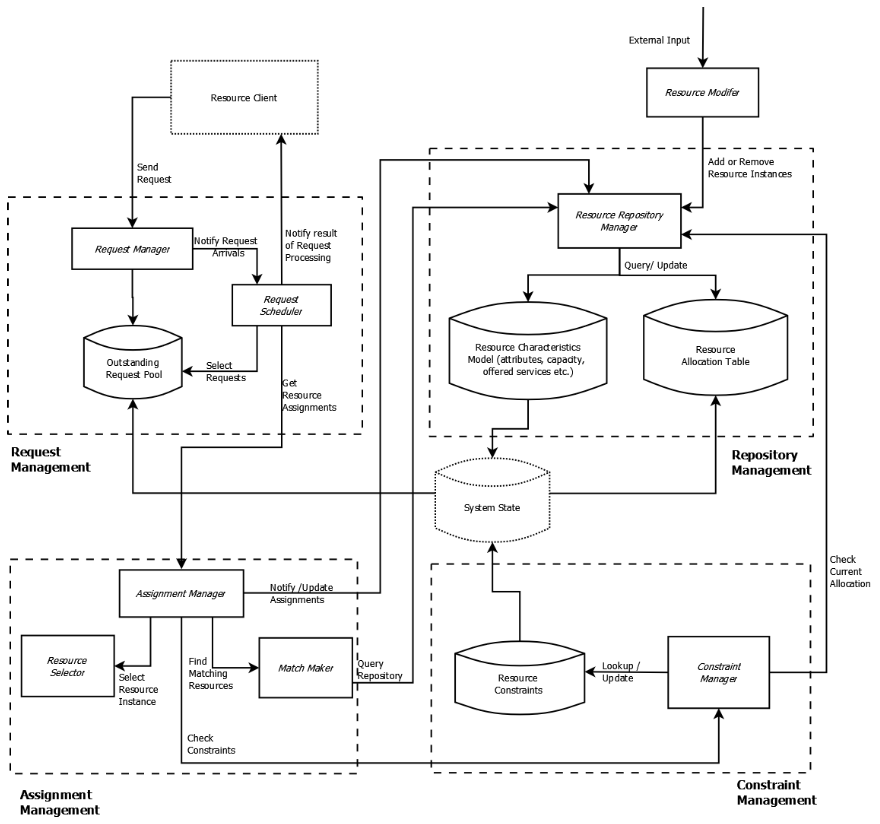

Figure 1: Resource Manager Architecture

Then $ASGN_{\Sigma_{STATE}}$ (t,c) is a resource $r \in R_\Sigma$ such that $t \in \Sigma_{ACTIVITIES,STATE}$ is an activity for which capability $c \in CAPS_t$ and such that $c \in CAPS_{r,\Sigma_{STATE}}$

We now define

$$ASGN_{\Sigma_{STATE}} = \bigcup_{t \in \Sigma_{ACTIVITIES,STATE}} \bigcup_{c \in CAPS_t} \{(t, ASGN_{\Sigma_{STATE}}(t,c))\}$$

3. THE ARCHITECTURE AND PROTOTYPE IMPLEMENTATION OF A RESOURCE MANAGEMENT SYSTEM

3.1 The Architecture

Our resource management architecture, depicted in Figure 1, is centered on four major components: Request Management, Repository Management, Assignment Management, and Constraint Management. To understand the nature of these components, we also hypothesize the existence of a System State component that represents the current state of both the executing process and the resource manager itself, and a Resource Client component that represents resource needs to be met. In this architecture a Resource Client sends requests to the Request Management component to ask for capabilities to be provided by resources managed by this system. The Resource Client shown here is an abstract representation of a task for which one or more *capabilities* are required. Clients request such services as *identification*, *reservation*, *acquisition*, or *release* of resources. The Request Manager sub-component of the Request Management processes raw requests from clients into resource queries, and places them into the *Outstanding Request Pool*. The *Request Scheduler* sub-component selects the request or requests from the *Outstanding Request Pool* to satisfy next. Selection is based on factors such as the priority of the requesting entity, and request arrival time. Requests in the *Outstanding Request Pool* can be individual requests or sets of requests that a Resource Client might need to be fulfilled atomically.

The Assignment Management component contains an *Assignment Manager*, a *Resource Selector* and a *Match Maker*. The *Assignment Manager* receives from the Request Scheduler requests that are to be fulfilled, and attempts to make assignments. The *Assignment Manager* also releases resources by unbinding them from clients, and determines the satisfiability of assignment requests (but without making any assignment). To fulfill assignment requests, the *Assignment Manager* calls the *Match Maker*, which treats requests as queries against the resource repository being managed.

The System State, Σ_{STATE} , consists of such information as the current assignments of resources, the different types of outstanding requests, past assignments made during the execution of the process, etc. System State may derive many of these information from other components. Once the *Assignment Manager* has identified candidates to fulfill a request, the *Constraint Manager* filters out resource instances that violate any current constraints. The resulting instances are then sent to the *Resource Selector*, which chooses the resource instance to be assigned, presumably influenced by the attributes of the different resources.

The Repository Management component contains four subcomponents. The *Resource Characteristics Model* defines the attributes, *capacity*, *capabilities*, *cost*, etc. of resource instances being managed. The *Resource Allocation Table* manages the set of current assignments of resource instances to requests, and assists in gathering such derived information as the state of all resource instances, including their availability at any point during system execution. The *Resource Repository Manager* provides an interface to the resource repository that enables the addition, removal, or modification of resource instances.

Assignment decisions are reported to the *Request Scheduler*, which notifies the Resource Client. When a Resource Client no longer requires a resource it must notify the Request Management component.

3.2 The ROMEO Prototype Resource Management System

ROMEO is a prototype resource management service based on the architecture just presented. Due to space limitations we now describe only a few of the less obvious features of ROMEO, omitting descriptions of such more straightforward components as the ROMEO repository manager, which is implemented around an unremarkable in-memory relational database, and the ROMEO resource assignment component, whose details are also unremarkable.

3.2.1 Resource Model

The ROMEO Resource Characteristics Model defines the static structure of the resources being managed, and a partial specification of the dynamic behavior of each resource. In ROMEO each resource is a uniquely identifiable object characterized as a set of name-value attribute pairs. To emphasize this view, we will often use the term *resource instance* to refer to a specific resource. Note in particular that ROMEO does not structure resource instances as a type hierarchy. Our experience has indicated that simple type hierarchies are not sufficient to represent clearly and completely the complex relations among resource instances in domains such a hospital ED. Multiple inheritance schemes also proved to be ungainly in dealing with such potentially conflicting issues as job titles, capability sets, and organizations. Accordingly ROMEO uses a flat structure for the resource instances it manages, making minimal assumptions about the attributes needed to describe these resource instances.

Table 1 shows an example of a resource instance specification contained in a ROMEO resource repository. The following explains the attributes used in table 1.

Name: Each resource can be “named” by a text string or other identifier. For resources that are humans, this attribute value is likely to be the name of the person. For inanimate resources, such as a bed or an X-ray machine, this attribute value may be a serial number or other distinguishing label.

Job Title: Each resource has an attribute whose value is a text string or identifier indicating the job that the resource performs. Some examples of *Job Title* attribute values are “Physician”, “Nurse”, “Clerk”, and “Bed”.

Location: Each resource has an attribute that indicates the organization or physical location where the resource works. In Table 1, the resource is specified to be located in the MainED. Usually this means that such resources can perform tasks in the main treatment area of the ED, but not tasks carried out in other ED locations.

Table 1: Example specifications of resource instances to support resource allocations for ED processes

Name:	John Smith
Job Title:	Physician
Location:	MainED
Experience Level:	10
Cost:	100
Capacity:	8
Offered Capabilities:	(MDInitialAssessment, *true*, 10, 2); (MDProcedure, *true*, 10, 5); (MDFinalAssessmentandDecision, *true*, 10, 3); (RNPaperwork, [availability(nurse) = 0 $\wedge$ crowding>100], 3, 1)

Experience Level: This attribute has a value that can be ordered either partially or totally to indicate a level of seniority or experience.

Cost: This attribute has numeric value indicating the cost of a unit of the resource’s work.

Capacity: This attribute is a numeric value that changes over time to represent the quantity of capability that a resource instance can currently make available. The capacity of a resource decreases as it is assigned to different tasks, and increases each time it completes an assigned task.

Offered Capabilities: This attribute is a set of capabilities that the resource is able to offer. These capabilities correspond to the set $CAPS_r$, defined earlier. Each such capability is characterized by a 4-tuple, consisting of Capability-name, the name of a task this resource instance can support, Capability-guard, the circumstances under which the resource instance can provide this support, Capability-skill-level, the skill level at which this resource instance supports this task, and Effort-needed, the amount of capacity that this resource instance needs to perform this task.

ROMEO provides a facility for attaching additional attribute types to individual resource instances.

3.2.2 Request Model

Modeling requests for resources is closely related to, but a separate concern from, modeling resource instances. A request model supports communicating requests for resource instances made by a given task t, namely

$$CAPS_t = \{CAP_{t_1}, CAP_{t_2}, \cdots, CAP_{t_{n_t}}\}$$

An important part of such a request is a specification of the requesting entity (i.e. the resource client, t). As noted above ROMEO resource requests are database queries. In particular, a resource request (e.g. for CAP_{t_i}), is essentially a request by a task t, for a resource r, that is currently able to provide a specific capability, c, which is the value of CAP_{t_i}. ROMEO supports three different types of such requests:

- Capability Request: A request for any resource, r, that is currently able to provide the requested capability, c.
- Characteristic Request: A request for a resource that can provide capability, c, but also has all of a list of

additional desired characteristics, and a list of constraints that must be satisfied. ROMEO attempts to satisfy such a request, but if there are no such resources available, then any resource that can simply provide the required capability is selected.

- Query-Based Request: A request for a resource that satisfies all of a specified set of predefined queries that are prespecified combinations of characteristics. In the ED domain, for example, we have defined a query, *attending physician*, that specifies a resource having "MD" as the value of its *Education* attribute, "ED permanent staff" as the value of its *Employment Status* attribute, and ">5" as the value of its *Experience* attribute. Such queries seem to provide some of the advantages of a formal type system by providing a primitive vehicle for characterizing properties shared by a subset of the resources.

3.3 Constraint Management

As noted earlier, constraints can be particularly useful in representing domain-specific policies, some of which can be quite complex. ROMEO supports the creation and application of many kinds of request constraints. Two types are of particular interest, a Resource-Collection constraint, and a Resource-Iterator constraint. A Resource-Collection constraint is a query specification that consists of one or more query names separated by commas. For example, a Resource-Collection constraint named `caregiver` may declare "doctor, nurse" as its query specification. ROMEO instantiates this type of constraint into a union collection of those resource instances that satisfy either the query corresponding to doctor or the query corresponding to nurse. A Resource-Collection constraint can optionally include a maximum cardinality specification. For example, the declaration "doctor, nurse, 5" specifies that the resulting collection's cardinality may not exceed 5. Similarly, Resource-Iterator constraint can also consist of a single query name or a list of query names separated by commas. ROMEO also supports specification of a maximum cardinality in a Resource-Iterator constraint.

ROMEO also supports the ability to define and name resource request constraints hierarchically with higher level constraints being defined in terms of lower level sub-constraints. ROMEO stores such queries in the resource repository. The following example shows how this capability can be used to specify a complex, yet realistic, ED resource assignment policy. Many hospital EDs are divided into a MainED where all patients can be treated and a FastTrackED where only patients with low acuity levels are treated. A query named *attendingMD* can be defined to select only resource instances whose *Job Description* attribute is "AttendingMD" and another query named *resident* to select only whose *Job Description* attribute is "Resident". A resource constraint named *fast-track-resources*, for example, can be defined to select only resource instances located in the ED fast track area. Still another resource constraint can be specified as *(attending, resident)* to describe resource instances that satisfy either of the two queries. Let us suppose we label this constraint as *doctor*. With these definitions in place, if we now specify a request for a *doctor*, which is spcified as constrained by *fast-track-resources*, this constrained query will return only resource instances who are either an attending MD or a resident, and who are currently working in the ED fast track section.

3.4 ROMEO Client Model

We assume that a typical ROMEO client is a task that is part of a process or workflow model that specifies temporal relationships, artifact flows, and resource requirements. We assume that sequencing and coordination are accomplished by a task interpreter facility that instantiates tasks according to the task coordination model specification. We assume further that each task, t, requires the services of a resource, r, that has the capability to perform the task, which is evidenced by the inclusion of t among the capability set of r, $CAPS_r$. We refer to this resource as the task's agent. ROMEO also allows for the possibility that performance of the task may require additional capabilities that are also specified as part of the task specification. These additional capabilities, along with the agent capability thus comprise what we have earlier denoted by

$$CAPS_t = \{CAP_{t_1}, CAP_{t_2}, \cdots, CAP_{t_{n_t}}\}$$

Supporting the assignment of these additional, non-agent, resources to tasks adds a considerable amount of power and complexity to ROMEO. But space limitations prevent our describing this facility in detail in this paper. Thus subsequent evaluation and examples focus only on the use of ROMEO in supporting the assignment of agent resources to tasks. A more detailed discussion of ROMEO is available at [20].

For this client model to work, all candidate agent resources are expected to have registered with ROMEO each indicating that it is available for providing a list of guarded capabilities. Further it is assumed that each such agent resource agrees to consult a to-do-list (i.e. an agenda) where tasks are placed once ROMEO has assigned the agent resource as the provider of a capability to a task.

4. EVALUATION SETUP

We used the Little-JIL process definition language [32], its execution environment, Juliette [8], and a discrete event simulation system, JSim [21], to construct an evaluation platform for ROMEO. Little-JIL supports the precise definition of processes involving different agent and non-agent resources. A Little-JIL process definition is comprised of four orthogonal components: 1) a coordination specification, 2) a resource specification that includes constraints, 3) a specification of artifacts (entities such as data items, files, or access mechanisms) and their flow and 4) a specification of the behaviors of those resources that can be assigned as agents. Juliette, which supports the execution of processes defined in Little-JIL, requires a capability for specifying and managing resources, and for responding to requests for resources to execute the steps of the process it is executing. ROMEO is suitable for this purpose. Thus Juliette is a vehicle for evaluating ROMEO's capabilities. On the other hand, as the execution of processes (particularly human-intensive processes) can take a considerable amount of time, we sought ways to expedite the evaluation of ROMEO. The JSim system, which generates discrete event simulations from Little-JIL process definitions, proved to be useful in that regard, as JSim supports the rapid generation of simulations that make intensive use of ROMEO's capabilities. As an added

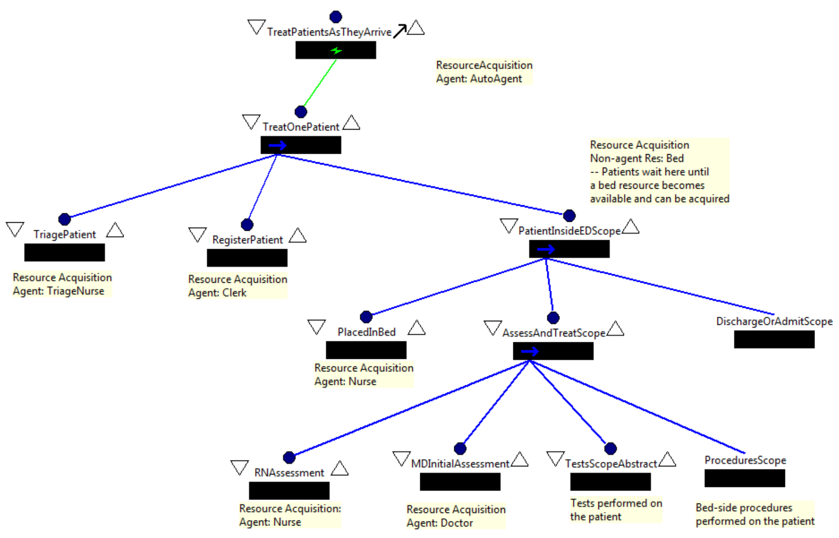

Figure 2: A Simple ED process in Little-JIL

benefit, the simulations, most of which were based upon realistic ED process scenarios, also generated results that were of interest to ED domain experts. These ED experts were thus highly motivated to participate in the close scrutiny of the resource assignments made by ROMEO. This helped our efforts to validate ROMEO and its effectiveness in supporting the modeling of ED resources and management policies governing them.

4.1 Modeling an ED process using Little-JIL

Figure 2 shows the Little-JIL process definition, called SimpleED, that describes how care is provided to patients at a typical hospital ED.

In SimpleED, a patient is first seen by a triage-nurse (TriagePatient step) who assigns a triage acuity level. The patient then goes to the registration clerk (RegisterPatient step), who collects information about insurance and other details and stores it into the patient's record. The registration clerk also generates an id-band and places it on the patient. The patient then goes inside the treatment area of the ED if a bed is available, or waits in the waiting room until a bed becomes available. This is modeled by a blocking acquisition request for a bed resource in the PatientInsideEDScope step. Once a bed is successfully acquired, the patient is placed in it (PlacedInBed step). A nurse resource is specified as the agent for the bed placement step. The patient is then assessed by a nurse in the RNAssessment step, and then by the attending doctor (MDInitialAssessment step). The doctor assessment may result in ordering tests. The resulting test related activities have been represented as a single abstract step named TestsScopeAbstract, which will be elaborated upon subsequently. Some bedside procedures may also be performed on the patient in the ProceduresScope step. Figure 3 shows the elaboration of DischargeOrAdmitScope. After all of this, the doctor makes a final assessment and decides whether to admit or discharge the patient (the MDFinalAssessmentAndDecision step), which is not elaborated here. Finally, the RNPaperWork step is performed by a resource whose job title is Nurse.

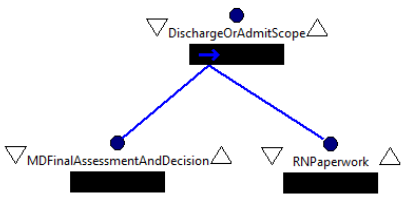

Figure 3: Discharge part of 'SimpleED' process

In addition to undergoing tests, the patient may also be treated with additional bedside procedures that may include suturing, casting, or intubation. Some of these procedures could be done by a nurse (RN) and others must be done by a doctor (MD). Predefined queries specify the constraints on the resources required for these steps. Throughout this process, a parameter named patientInfo (not explicitly shown in Figure 2) is passed into and out of each step. This parameter carries information related to the current state of the patient. As agents carry out different steps, they may use this information and insert additional information for use by subsequent steps and their agents. This information may be used to determine the sequence of process steps executed.

4.2 JSim: Generating Discrete Event Simulations from Little-JIL Process Definitions

JSim discrete event simulations are generated from Little-JIL process definitions [21]. A JSim simulation proceeds as an iteration through the steps of a process definition. Simulation of the SimpleED process is begun by an initial Patient Arrival event, but then continued by subsequent patient arrivals, and by the fact that the simulation of each step creates one or more new events, each representing something to be simulated, such as step completion, spawning of substeps, etc. Of particular interest for evaluation of ROMEO, the Resource Manager is consulted during each step simulation to obtain the agent resource needed to determine how step performance is to be modeled. The JSim Agent Behavior Specification (JABS) language is used to specify such behavior as how long a given agent will require to perform a given step, which additional simulation events are to be scheduled, how likely it is that a particular choice will be made, or that a particular exception will be thrown. In simulating the ED patient care process, these estimates were based on interviews with ED professionals, and analysis of statistical data. Space does not allow a detailed description of JABS, but full details can be found in [20].

JSim execution produces a trace file that holds such information as which agent resource instance was assigned to which task at what time, when the agent resource instance started working on that task, and when the agent resource instance completed the task. ROMEO also produces an allocation file that lists the resource that was assigned in response to every request, and the capacity that the resource had at the time of its assignment. These outputs were the basis for both validating the simulation, and for evaluating the effectiveness of ROMEO in managing resources in ways consistent with defined policies, preferences, and constraints.

5. CASE STUDIES AND EXPERIENCES

Space limitations allow for the description of only a very small number of the simulations used to evaluate ROMEO. We describe them here both to show the use of features of ROMEO, and to demonstrate how those features were effective in supporting the evaluation of some potential resource changes in rather complex hospital ED systems.

5.1 Dynamic Changes in the Capabilities Offered by Resources

One key hypothesis of this research is that it is important to model the way in which resources change the set of capabilities that they offer depending upon the state of the execution of a system. To evaluate this, we considered the SimpleED process shown in Figure 2, and set up a simulation experiment with the following capability modification scenario. We hypothesized that the steps PlacePatientInBed and RNPaperwork do not always have to be done by a registered nurse (RN), but that triage nurses who ordinarily perform only triage operations might substitute an RN in placement of a patient in a bed or in discharge paperwork when all of the following conditions hold: 1) the ED is overcrowded, 2) all RNs are busy, and 3) a resource instance of type `TriageNurse` is available. We further suggested that the step RNPaperwork might also be performed by a registration clerk when all of the following conditions hold: 1) the ED is overcrowded, 2) the clerk is idle, and 3) there is no nurse available for performing the discharge paperwork. We measured the crowdedness of the ED based on the number of patients who have gone through the TriagePatient and RegisterPatient steps but are currently waiting for a bed to become available. To simulate this scenario, we changed the agent resource request for PlacePatientInBed and RNPaperwork from `Nurse` to `default` (a request for any resource instance that is capable of providing the capability). Here the capability required is specified implicitly as being the activity itself, i.e. PlacePatientInBed or RNPaperwork.

Table 2 shows the guard functions on the capabilities for nurses and registration clerks that support the scenario just described. Other experimentation, and consultation with the ED domain expert, had suggested that a realistic mix of ED resources would consist of 13 beds, 4 doctors, 4 nurses, 2 triage nurses and 2 clerks. Once this resource mix and these resource capability modification policies were decided, the work required to implement them as ROMEO resource management policies took less than thirty (30) minutes. It is interesting to note, however, that we were not immediately able to define either the specific resource mix or the set of resource substitution rules that were very effective in reducing ED Length of Stay (LOS). This required trying many different combinations of parameters. It underscores the value of a powerful and flexible system for specifying resource capability modification policies, as arriving at effective policies may require considerable experimentation. The simulations suggested that the policies described here can indeed support a significant reduction in the average LOS. For example, we specified that the times taken to perform each step be provided by a triangular distribution, and that patient arrivals be simulated with a Poisson distribution with a mean inter-arrival time of nine (9) minutes. We then ran the simulation five times, each with 300 patients. The data, plotted in Figure 4, shows the reduced patient LOS achieved using the dynamic capability modification rules we specified.

Table 2: Partial definition of guard function conditions under which resources can offer capabilities

CAPABILITY-NAME	RESOURCE-GROUP	CONDITION
PlacePatient-InBed	Nurse	true
PlacePatient-InBed	TriageNurse	StateServer.-Pending-Requests-(“bed”)>N
RNPaperwork	Nurse	true
RNPaperwork	TriageNurse	StateServer.-Pending-Requests-(“bed”)>N
RNPaperwork	Clerk	StateServer.-Pending-Requests-(“bed”)>N

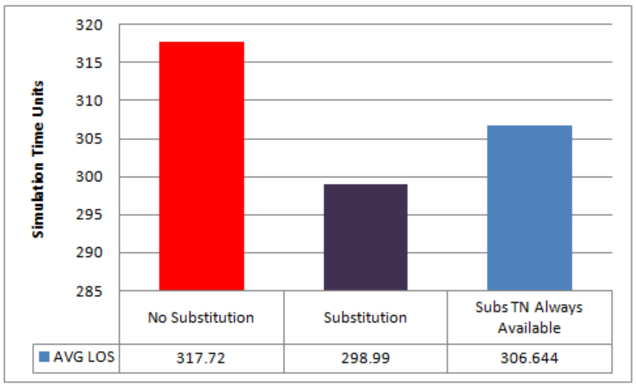

Figure 4: The impact of dynamic substitution

We then modeled a more complex, but more realistic, domain policy stating that a triage nurse is allowed to substitute for a regular nurse only when there is at least one triage nurse left available to attend to a newly arrived patient. To model this, we took less than five minutes to add the capability guard function shown in in Table 3.

Table 3: Elaboration of substitution condition for triage nurses

CAPABILITY-NAME	RESOURCE-GROUP	CONDITION
PlacePatient-InBed	TriageNurse	StateServer.-Pending-Requests-("bed")>N && StateServer.-Available-("TriageNurse")>1

This change still reduced the average patient LOS, but not by as much as the previously described policy. Scrutiny of ROMEO's behavior confirmed that it was faithfully enforcing the new policy. The results obtained, however, suggested that more experimentation might be desirable to assure that the results were not an artifact of the particular experimental scenario.

5.2 Constraining Resource Choice

In another case study we used ROMEO and JSim to model the following constraint:

> In a hospital ED, the doctor who performs InitialAssessment on a patient must be the same as the doctor who performs the FinalAssessment and makes the decision regarding discharging or admitting the patient.

We used the Resource-Collection constraint, described previously, to implement this policy. A Little-JIL resource-collection constraint was used to specify in the SimpleED process that the steps MDInitialAssessment, MDProcedure, and MDFinalAssessmentAndDecision not only require a doctor, but that the doctor required for each of these steps has to be the same as for the other steps. We specified this by making use of the fact that Resource-Collection (and Resource-Iterator) constraints can be copied from a Little-JIL parent step to a child step. Thus, by using a parameter named Doctor-Constraint as a Resource-Collection constraint for the PatientInsideEDScope non-leaf step, this constraint was copied down the Little-JIL step hierarchical decomposition tree from this parent step to its children steps. ROMEO ensured that all resources returned actually did satisfy the constraint because the constraining collection was defined to have maximum cardinality of 1, assuring that the same resource instance was assigned to all these steps. The effort needed to specify this case study took about 35 minutes of modeling time. It took only a few minutes to set up a comparison simulation in which there were no constraints on which doctor was assigned to any of the steps. Space limitations prevent the presentation of the full details of these simulation runs. They can be found in [20]. But, as intuitively expected, the actual simulations showed that adding the constraint increases the average LOS. In the specific simulations we ran, having the constraint maintained all the time adds an average of 25.85% to overall patient LOS.

5.3 Other simulation case studies

Additional case studies demonstrated how ROMEO could help determine the effect of separating ED resources into separate areas and using different policies to govern when resources from one area could be used in another area. ROMEO constraint specifications were readily used to model a range of such policies. In other case studies, we showed that it was relatively easy to model how the actual execution of a process can be changed based upon such context conditions as the availability of resources [20]. This capability was facilitated by the maintenance of process execution state information for use by ROMEO, and also by ROMEO's own internal resource utilization state management facilities. Other case studies showed that it was relatively straightforward to specify different priorities for different steps and different patients, and to then use that priority information to affect the decisions ROMEO made about allocation of resources. Still other simulations exploited ROMEO's facilities for supporting the assignment of multiple resources to individual tasks, for allowing the possibility that such requests may need to be atomic, and for supporting blocking requests. Finally we note that ROMEO has also been used to support the execution of processes in domains other than hospital EDs. In particular ROMEO was used to manage the resources needed by processes in the domain of labor-management dispute resolution.

5.4 Validation activities

Extensive efforts were made to assure that ROMEO's behavior was consistent with its specifications. Some of these efforts amounted simply to careful visual scrutiny of the resource assignment decisions ROMEO made. Both ROMEO's developers and our ED domain expert spent considerable amounts of time looking at simulation results to verify ROMEO's decisions. In addition we compared the simulation results obtained from ROMEO-JSim to the results of simulations supported by other simulation systems. We ran several batteries of simulations using the popular Arena [4] discrete event simulation system, and compared the results obtained to the results provided by ROMEO-JSim. The results obtained were the same for both systems, but these comparisons were on simulations for which the resource assignment choices were relatively straightforward, as the resource management capabilities of Arena are far more restrictive than those provided by ROMEO-JSim. Additional comparisons of more complex simulations should be carried out to increase our confidence in ROMEO's behavior. In addition we compared the results obtained from ROMEO-JSim to those predicted by Little's Law [6, 20], a well-known "rule of thumb" used to estimate and validate the overall behaviors of queue-based simulations. We examined the qualitative outputs of our simulation runs and assured that the results obtained were consistent with expectations and with the predictions of Little's Law.

6. RELATED WORK

There has been a great deal of investigation of different aspects of managing resources in a wide variety areas. In this section, we present a necessarily brief summary of that work.

Managing resources is at the heart of much operating systems and networking research. Most of this work, however, is concerned primarily with allocation strategies and scheduling, and has dealt with relatively homogeneous types of resources such as processor time, memory/disk space, network bandwidth, Internet hosting servers etc.[9, 22, 30]. Distributed computational platforms like grid computing and server clusters are concerned with managing distributed and somewhat more heterogeneous types of resources [19, 16].

Workflow and process languages provide various mechanisms for resource specification and utilization [3, 1, 2]. Some of the workflow and process languages that address resource management issues include APEL [11], MVP-L [23], APPL/A [29], Process Weaver [7], and BPEL4WS [1]. However, the modeling capabilities in these languages are restrictive, and their support for describing resource relationships, constraints, request specification and resource allocation are minimal. In particular, BPEL4WS focuses mainly on web services as resource objects, and BPEL4People [15] focuses on human participation in web services. [25] introduce the concept of "workflow resource patterns", which seem close to the resource request specification approach presented here. But their work does not address the need for resource assignment.

Artificial Intelligence research has primarily been concerned with scheduling resource objects [28], but much of that work treats scheduling as a static, well-defined optimization task, where our approach addresses the inherent need for dynamism. The operations research (OR) community has historically explored solutions to resource management problems using a combination of dynamic programming and combinatorics [27]. Ontology research is relevant as it supports creating knowledge structures that could be used to model resources [13]. Most ontological frameworks use some sort of logic language to express these concepts and their relationships. The semantic web [5] makes heavy use of ontologies. Languages like DAML [14], DAML+OIL and OWL [17] are good examples of this approach. Such languages can be useful in describing structures and relationships of resource objects, but they fall short in describing resource constraints, which we have found to be essential to modeling many real-world resource utilization policies.

There has been considerable work in modeling and simulating hospital processes. Connelly and Bair [10] presents a discrete event simulation system that predicts actual patient care times using simulation. Their model does not allow for dynamically changing the capabilities offered by resources, however, and their predictions of patient service times were correct to within less than one hour for only 28% of their simulated patients. Draeger [12] developed simulations to assess nurse staffing concerns and alternatives for improvements. McGuire [18] discusses the use of simulation to test process improvement alternatives aimed at reducing the length of stay for ED patients. Rossetti [24] looks 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. Samaha [26] uses ED simulation studies to perform 'what-if' analysis regarding the effect of process change and staff level change on LOS. All these simulation studies have taken a factory view of the ED, where patients come in like orders on a factory floor 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.

7. CONCLUSIONS AND FUTURE DIRECTIONS

Modeling and managing resources is a problem for systems in many domains. This paper identified key problems in defining, managing, and assigning heterogeneous resources in environments that are very complex and highly dynamic. We proposed a novel characterization of the nature of resources, suggested a generic resource management service architecture, built a prototype, and evaluated our characterization and approach primarily by driving discrete event simulations of patient care in a hospital ED. The results obtained support our suggestion that resources should be characterized as having dynamically changing capability sets. Further experience demonstrated the value of focusing on constraint management as a key issue in implementing complex resource management policies. These approaches greatly facilitated the work of contriving simulations of different resource assignment strategies, leading to suggestions for potentially significant improvements in the functioning of a hospital ED. More case studies in the ED and additional domains are needed to provide a clearer view of which features of our approach seem most effective. Finally, we note that the resource assignment described here is done on a first-come-first-served basis. It seems clear that scheduling resources in advance could help to avoid resource bottlenecks and other inefficiencies. Integrating a resource manager with an explicitly defined process, such as we have done in our work with JSim, creates the possibility of obtaining and exploiting look ahead information that could lead to these kinds of improvements in resource assignment.

8. ACKNOWLEDGMENTS

The authors gratefully acknowledge the important contributions made to this work by many conversations and discussions with Sandy Wise, who made many valuable suggestions about the architecture of ROMEO. Guillaume Viguier-Just and Tiffany Chao provided very important support for the construction of case studies and other evaluative activities. Dr. Philip L. Henneman, MD, was an indispensable source of information about the nature of hospital EDs and the processes carried out there. Hari Balasubramanian, Barbara Lerner, Bobby Simidchieva, Lori Clarke, Matt Marzilli, and George Avrunin provided a great deal of help and support through their continuing interaction about this work.

This work was supported by the US National Science Foundation under Award Nos. CCR-0427071, CCR-0204321 and CCR-0205575. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of The National Science Foundation, or the U.S. Government.

9. REFERENCES

[1] *BPEL4WS Specification, Version 1.1.* IBM, 2005.

[2] *BPMN 1.0: OMG Final Adopted Specification.* Object Management Group, 2006.

[3] W. M. P. v. d. Aalst and A. H. M. t. Hofstede. Yawl: Yet another workflow language. Technical report, Eindhoven University of Technology, 2002.

[4] V. Bapat and D. T. Sturrock. The arena product family: enterprise modeling solutions: the arena

product family: enterprise modeling solutions. In *Proceedings of the 35th conference on Winter simulation*, pages 210–217, 2003.

[5] T. Berners-Lee, J. Wendler, and O. Lassila. The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 284:34–43, 2001.

[6] D. Bertsimas and D. Nakazato. The distributional little's law and its applications. *Operations Research*, 43:298–310, 1995.

[7] M. L. Brasseurl and G. Perdreaul. Process weaver: from case to workflow applications. In *IEEE colloquium on CSCW and Software Process*, 1995.

[8] A. G. Cass, B. S. Lerner, S. M. Sutton, E. K. McCall, A. E. Wise, and L. J. Osterweil. Little-jil/juliette: a process definition language and interpreter. In *International Conference on Software Engineering (ICSE)*, pages 754–757, 2000.

[9] A. Chandra. *Allocation for Self-managing Servers.* PhD dissertation, University of Massachusetts Amherst, Department of Computer Science, 2005.

[10] L. G. Connelly and A. E. Bair. Discrete event simulation of ED activity: A platform for system-level operations research. *Academic Emergency Medicine*, 11(11):1177–1185, 2004.

[11] S. Dami, J. Estublier, and M. Amiour. Apel: A graphical yet executable formalism for process modeling. *Automated Software Engineering: An International Journal*, 5(1):61–96, 1998.

[12] M. A. Draeger. An emergency department simulation model used to evaluate alternative nurse staffing and patient population scenarios. In *Proceedings of the 24th Conference on Winter Simulation*, pages 1057–1064, Arlington, VA, USA, 1992.

[13] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Intl. Journal of Human-Computer Studies*, 43:907–928, 1995.

[14] J. Hendler and D. L. McGuinness. The darpa agent markup language. *IEEE Intelligent Systems*, 15(6):67–73, 2000.

[15] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. Riegen, P. Schmidt, and I. Trickovic. WS-BPEL extension for people. Technical report, IBM, August 25 2005.

[16] C. Liu and I. Foster. A constraint language approach to grid resource allocation. In *Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, 2003.

[17] D. L. McGuinness and F. v. Harmelen. Owl web ontology language. http://www.w3.org/TR/owl-features/, 2004.

[18] F. McGuire. Using simulation to reduce length of stay in emergency departments. In J. T. Seila, S. Manivannan, D. Sadowski, and A.F., editors, *IEEE Winter Simulation Conference*, pages 861–867, 1994.

[19] R. Raman, M. Livny, and M. Solomon. Matchmaking: An extensible framework for distributed resource management. *Cluster Computing*, 2(2):129–138, 1999.

[20] M. S. Raunak. *Resource Management in Complex Dynamic Environments.* PhD thesis, University of Massachusetts Amherst, Department of Computer Science, 2009.

[21] 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 *Proceedings of the Software Engineering in Health Care (SEHC) 2009*, Vancouver, Canada, May 2009.

[22] M. S. Raunak, P. Shenoy, P. Goyal, and K. Ramamritham. Implications of proxy caching for provisioning networks and servers. In *Proceedings of the ACM SIGMETRICS*, pages 66–77. ACM, 2000.

[23] H. D. Rombach. MVP-L: a language for process modeling in-the-large. Technical report, University of Maryland, College Park, MD, USA, 1991.

[24] M. D. Rossetti, G. F. Trzcinski, and S. A. Syverud. Emergency department simulation and determination of optimal attending physician staffing schedules. In *1999 Winter Simulation Conference*, 1999.

[25] N. Russell, W. v. d. Aalst, A. Hofstede, and D. Edmond. Workflow resource patterns. In *17th International Conference on Advanced Information Systems Engineering (CAISE '05)*, volume 3520 of *Lecture Notes in Computer Science*, pages 216–232, Portugal, 2005. Springer Verlag.

[26] S. Samaha, W. S. Armel, and D. W. Starks. The use of simulation to reduce the length of stay in an ed. In *Proceedings of the 35th conference on Winter simulation*, pages 1907–1911, 2003.

[27] D. W. Sellers. A survey of approaches to the job shop scheduling problem. In *Proceedings of the 28th Southeastern Symposium on System Theory*, 1996.

[28] S. Smith. Is scheduling a solved problem? In P. C. Kendall and Graham, editors, *The Next Ten Years of Scheduling Research*, pages 116–120, 2003.

[29] S. M. Sutton, Jr., D. Heimbigner, and L. J. Osterweil. Appl/a: a language for software process programming. *ACM Transactions on Software Engineering and Methodology*, 4(3):221–286, 1995.

[30] B. Urgaonkar. *Dynamic Resource Management in Internet Hosting Platforms.* PhD dissertation, University of Massachusetts Amherst, Department of Computer Science, 2005.

[31] WFMC. Workflow management coalition terminologh & glossary. Technical Report WFMC-TC-1011, Workflow Management Coation, February 1999.

[32] A. Wise. Little-jil 1.5 language report. Technical Report 2006-051, University of Massachusetts Amherst, October 2006.

[33] M. zur Muehlen. Resource modeling in workflow applications. In *Proccedings of Workflow Management Conference (WFM99)*, pages 137–153, 1999.