Modeling and Managing Resource Utilization in Process, Workflow, and Activity Coordination

Barbara Staudt Lerner*, **Anoop George Ninan, Leon J. Osterweil, Rodion M. Podorozhny**
Laboratory for Advanced Software Engineering Research
Department of Computer Science
University of Massachusetts
Amherst, MA 01003, USA
+1 413 545 2013
lerner@cs.williams.edu {agn, ljo, podorozh}@cs.umass.edu

ABSTRACT
Specifications of workflow, process, and activity coordination systems focus on the assignment of work to human and software agents and the dataflow required to support those activities. The runtime behaviors of these systems vary widely depending upon the availability of agents and other needed resources. Thus the precise specification of resources needed by, and available to, a system is an important basis for reasoning about and optimizing system behavior. Previous resource models used in management and workflow have lacked the rigor to support powerful reasoning and optimization. Some resource models for operating systems have been quite rigorous, but overly narrow in scope. This paper presents a meta-model for creating precise models of such resource types as humans, tools, computation platforms, and data, and the various associations between these types that are needed to support intelligent allocation of these resources. We also present examples of the use of this meta-model. This paper also describes a prototype resource allocation and management system that implements these approaches. This prototype is designed to be a separable, orthogonal component of a system for execution of processes defined as hierarchies of steps, each of which incorporates a specification of resource requirements.

Keywords
Resources, Resource Management, Workflow, Activity coordination, Agent coordination.

1 INTRODUCTION

Much research in such areas as software process, workflow, CSCW and multi-agent systems focuses on devising mechanisms for specifying coordination of diverse activities to accomplish complex tasks. Most approaches in these domains represent the overall task to be accomplished as a synthesis of lower-level tasks. In addition, different approaches incorporate, to differing degrees of formality, the specification of the artifacts that these various tasks use and produce. Some also have mechanisms to specify the agents and resources to be used to support task execution.

The differences in emphasis on these different components of activity coordination specification are often clearly due to differences in goals. Some specifications are often intended to be largely illustrative and advisory, being intended for use in helping humans come to common understandings. But many specifications are intended to be sufficiently rigorous that they can be used as prescriptions for computer support of resource allocation and scheduling through the application of tools.

Our past work has this latter goal. We have concentrated on developing and evaluating resource specification formalisms that are sufficiently rigorous that they can be used to reason about such complex activities as the processes used to develop large software systems and how to support these processes most efficiently. This has demonstrated the need for structuring the tasks that comprise the larger overall processes and the need for being articulate and complete in specifying the artifacts that these tasks use and produce.

Operating systems research long ago demonstrated the need for reasoning about resources in parallelizing, coordinating and optimizing the execution of system processes. Clearly, the abundance of resources can enable execution of tasks in parallel, thereby speeding up accomplishment of larger goals. This same phenomenon is also clearly observable in broader classes of activity coordination. If a software design activity requires the use of a specific design tool, then the various members of a design team can work in parallel only if multiple licenses of the tool are available. Similarly, if the activity is decomposed into several parallel design subtasks, the design process may proceed faster, but only if more than one qualified designer is available to work on the project.

* Currently at Department of Computer Science, Thompson Computing Laboratory, Williams College, Williamstown, MA 01267, USA. +1-413-597-4215.

The lack of resources causes contention, occasions the need for some tasks to wait for others to complete, and generally slows down accomplishment of larger goals. Often potential delays can be avoided or reduced by using resource analysis to identify ways in which tasks can be made to execute in parallel that avoid resource contention.

Problems in operating systems like deadlocks and livelocks are very real for the larger class of activity coordination problems as well. It is not hard to devise a process in which a requirements analyst awaits the results of a prototype activity in order to complete requirements specification, while a prototyper awaits the completion of the requirements specification to complete the prototype.

In addition to the scheduling concerns outlined above, reasoning about resources requires an understanding of the similarities and differences among resources as well as the precise needs of the activities being coordinated. With this understanding it is possible to identify in which situations only a particular resource will suffice and in which situations any of a class of resources may be acceptable for the task. This identification of critical resource needs can also facilitate the identification of likely bottlenecks in the execution of a process or workflow.

Having the ability to describe resource classes and their membership also facilitates improvements in the execution of coordinated activities without requiring change to the activity specification itself. For example, by adding a second resource compatible with some critical resource, the throughput of the coordinated activity should improve. A superior resource may also result in qualitative improvements to the output of the activity, again without changing the activity specification itself.

In our process work, we would like to create processes that use a wide variety of resources. In addition, we would like to be able to treat a group of resources, say a team of humans, as a single allocatable unit and support effective resource sharing among tasks through fractional allocation policies. We would like to be able to identify where deadlocks, race conditions and starvation are possible, so that we can assure that they do not occur. We would also like to be able to infer when additional resources can speed up execution of the overall activity, and when there are extra resources that may be reassigned. We would like to be able to manage contention for popular resources and those that require mutual exclusion.

The preceding sorts of reasoning and control seem to us to be impractical unless the resources needed by tasks are specified. As with most software analyses, these sorts of reasoning can be more powerful and precise when the rigor with which needed resources are specified is more powerful and precise. Thus in this work, we propose a powerful and precise resource specification formalism to provide a basis for the kinds of analyses indicated above.

2 MOTIVATION

As an example to motivate our work, we show how management of resources specified by our modeling formalism can improve the effectiveness of a small software development team. Consider a team whose members have a variety of skills. Besides the people, other resources include licenses for tools like Rational Rose and Visual C++. Assume that the team is currently in the later stages of development of a product, and is both completing implementation and also carrying out some redesign that is required in order to fix bugs that have been detected in earlier work. The progress of the team may be hampered due to lack of resources. In particular there are probably times when having more qualified designers or coders would help. More likely it could be that the progress of the team could be expedited if the team had available an additional Rose or VC++ license.

A reasonable scenario here might be that the organization has money to purchase only one additional license, and it would be important to decide in some way, the best choice, a Rose license or a VC++ license. Our resource specification capability is intended to be a facility for supporting informed decisions of precisely this sort. It might be used as the basis of schedule reduction achievable through one purchase or the other. It might be used to study increases in human resource utilization, or it might be used to suggest ways in which the process using this model itself might be altered to reduce the schedule without having to purchase new licenses at all. Of course, a small example like this can easily be analyzed without automated tools. As the number of resources increases, the complexity of their interactions, and the complexity of processes being supported increases, these analyses become more difficult to do without automated support.

We now introduce our resource modeling and management approach and indicate why it seems to us to be effective in dealing with problems such as these.

3 RESOURCE MODEL ABSTRACTIONS

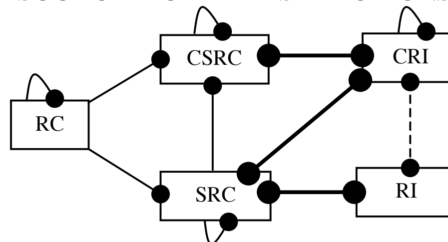

Fig 1: Abstract representation of our Resource Model

Our resource manager is a component of a larger system that is used to program and execute processes [10], support reasoning about real-time systems, or be integrated into a planning system, for example. Since we believe that this model may be used to specify resources in any domain, we have designed the meta-model to support a wide range of

resource types, which may be physical entities like robots, electronic entities like programs or data, people etc.

In *Fig 1* above, we depict a representation of our resource meta-model. The boxes indicate the various types of resources we support. The thin solid edges are is-a links connecting a resource class with a more specific resource class. The thick solid edges indicate type-instance relationships between schedulable resource classes and instances. The dashed edge is a whole-part relation, discussed later in this section. The ball at the end of an edge denotes cardinality greater than 1.

The elements of this meta-model are used to create resource models that represent those entities of an environment that may be required, but for which an unlimited supply cannot be assumed. A resource model is defined as a collection of **Resource Classes (RC)** and **Resource Instances (RI)**. A *Resource Instance* represents a unique entity in the environment, such as a specific person, printer or document. A *Resource Class* represents a set of resources (or classes) that have some common properties. Some resource classes may be **Schedulable**. A **Schedulable Resource Class (SRC)** is one whose instances are similar enough that the user of the resource model might not care which instance is chosen. For

LEGEND

is-a edges
type-instance edges
Whole-part edges (Fig 2b)
Resource Class
Schedulable Resource Class
Composite Schedulable Resource Class
Resource Instance
Composite Resource Instance

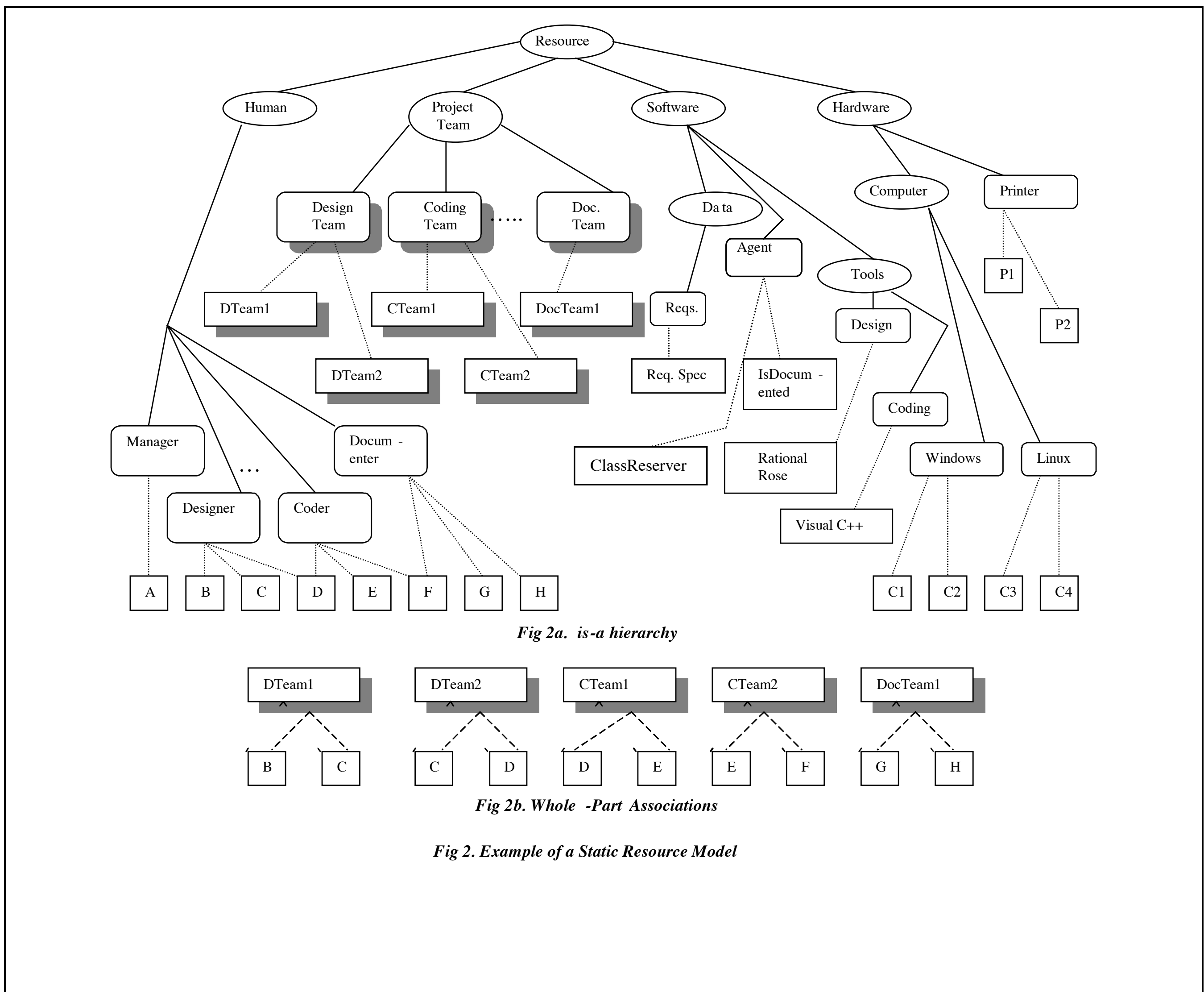

Fig 2a. is-a hierarchy

Fig 2b. Whole -Part Associations

Fig 2. Example of a Static Resource Model

example, *Printer* could be a *Schedulable Resource Class*. Other resource classes are **unschedulable** (RC) and are more abstract. They are intended to be more for an organizational convenience when defining resource models. Both resource classes and instances may be collected to form groups or **composite** objects. Thus a **Composite Resource Instance (CRI)** is one that forms a group of other *Resource Instances*, composite or otherwise. A **Composite Schedulable Resource Class (CSRC)** can have other *Composite Schedulable Resource Classes* and/or *Composite Resource Instances*. An example of a *Composite Resource Instance* is a specific team of humans or a cluster of printers whereas an example of a *Composite Schedulable Resource Class* would be a *DesignTeam* class, say, that had several teams of designers as instances. Therefore, any resource allocation method applicable to design teams in general may be applied to such a class.

Resource Attributes

We represent each resource as a set of typed attribute-value pairs. There is a small set of predefined attributes. The predefined attributes required of all resources are: a *Name*, a *Textual Description* of the resource and a set of *Criteria Assertions*. (Criteria assertions are described later in this section.)

Schedulable Resource Classes have *Unit* attributes. The *unit* of a resource is a label that describes how the resource is measured. For example, an electrical circuit might be measured in amps, or a person's time might be measured in number of hours per day. All elements further down the hierarchy of a class assume identical unit attributes. *Resource Instances* have *Capacity* and *Availability* attributes. The *Capacity* of an instance indicates the maximum acquirable quantity of the instance. *Availability* attributes track use of resources indicating the quantity of a resource that is available for use. Also needed are *AllocatedTo* and *AmountAllocated* attributes, described later in this section.

In addition, the resource modeler can associate user-defined attributes that are unique to the specific resource being modeled. The attribute values of a resource serve to *identify* the resource and distinguish it from other similar resources. For example, a *Printer* may have an attribute indicating whether it is a color printer or not.

Resource Associations

The resource modeling mechanism supports the definition of three types of associations among the types elaborated above: *is-a* associations, *whole-part* associations and *requires* associations.

is-a hierarchy

The relationship between a resource class and its members is expressed with an *is-a* link. The resource classes and their *is-a* links form a singly rooted tree. The *root* of the tree is a predefined *Resource Class* called *Resource*. A *Resource Instance* may belong to many resource classes; hence the entire model forms a singly rooted DAG. Each child of an *is-a* link inherits all attributes of its parents. If the same attribute name appears in multiple parents and is originally defined in separate classes (i.e., not in a common ancestor), the instance contains all attributes, qualified by class name. So for example, if Becky is an instance that belongs to both the Programmer and the Translator classes, and both have *Language* attributes, then she could have C++ and Java as attribute values for the *Programmer.Language* attribute and English and French for her *Translator.Language* attribute. An example of an *is-a* hierarchy is shown in *Fig. 2a*.

Whole-Part Associations

The semantics of a *Whole-Part* association is that a resource is physically or logically part of a composite resource. Both the composite and the part are *schedulable* resources (instances) so that resources may be used as a whole, or parts of it may be used individually. An example of a whole-part association is that a team is a composite whose parts are the team members (see more examples in *Fig. 2b*). Any assignment to a team is therefore a joint assignment to the team members.

Whole-Part associations have a ***Contribution*** attribute that defines how much of the part's capacity is devoted to the composite. The capacity of a *Composite Resource Instance* is the sum of the contributions of each constituting *Resource Instance*. Therefore, the capacity of a *Composite Resource Instance* changes dynamically as *Resource Instances* join or leave the group. An instance can be a part of multiple composites as long as it does not over-commit its own capacity. For example, a programmer may work for more than one team.

To elucidate, let CRI represent a team, and each RI_i represent constituting team members. Let X_i be the capacity of each RI_i, X be the capacity of the CRI and x_i be contributions of each RI_i to the CRI, where $0 <= x_i <= X_i$. *For i=1 to n*, if x_i were not indicated, $x_i = X_i$, by default. Therefore, the range of values that the capacity of CRI can assume may be expressed as $0 <= X <= \Sigma X_i$ and in general, Capacity(CRI) = Σx_i. Note that since a given RI may be a part of more than one CRI say $CRI_1 \ldots CRI_m$, then all the allocations may be satisfied at a time, only if, *for j=1 to m,* Σ *Contribution(*RI, CRIj) <= Capacity(RI).

Requires Associations

Functional dependencies between resources are captured with a **Requires** association. This indicates that in order for one resource to be useful, it needs another. For example, a piece of software requires a computer that is configured to run it. The semantics of the *Requires* relationship differs depending on the type of the destination node of the edge. In general, if a *Requires* edge falls on an *instance*, it implies that a specific instance is required by another

resource. However, if a *Requires* edge falls on a *Schedulable Resource Class,* it implies that any one of the instances of that class is required.

There are five attributes we may associate with *Requires* edges. The first is the *AmountRequired* attribute. This attribute is specified when exclusive use of an *instance* is not required, i.e., only a fraction of the resource's capacity is required. This allows for *fractional allocation* of resources and enables resource sharing. This is necessary to support parallel tasks that need to share resources. The second is the *Cardinality* attribute. This attribute may be specified on *Requires* edges that fall on *Schedulable Resource Classes* only. Other attributes are described later in this section.

If a *Requires* edge originated from a resource class, all instances of the class would automatically inherit this requirement. The third attribute, the *Name,* is used to allow overriding instead of inheritance. Instances can override an inherited edge by specifying their own edge with the same name. If however, a new *Requires* edge, with a name different from the one that exists at its parent's level, is added, the requirement for this instance, would be considered additive, to what is required at the class level.

Now we elaborate the semantics of the attributes just mentioned. Let the *AmountRequired* attribute be *Y units* on an edge falling on a *Resource Instance*, where *Y* <= Capacity(*Resource Instance*). If this attribute is not specified, the resource manager, by default, allocates the instance, in its entirety. In the case of a composite resource instance, if Y < Capacity(CRI), then (Y/Capacity(CRI) * 100%) of each constituent Resource Instance's contribution to its composite would get allocated. That is, the contribution of each part to the composite would be reduced proportionally.

For more concrete examples of this usage of the *Requires* edge, refer to the resource model example in *Fig 2*. Let the *Unit attribute* of the *Printer* class be *pages/day*. Let P1 and P2 have *Capacities* of *200* and *500 units* respectively. If A is a *Manager* who requires a dedicated printer, P1, there would exist a *Requires* edge from A to P1. If F is a *documenter* who needs no more than *50 units* of P2, this may be represented as a *Requires* edge from F to P2, with *AmountRequired* set to *50*.

For *Requires* edges falling on *Schedulable Resource Classes*, if only the *Cardinality* attribute is specified to be *m* on the edge, this implies the requirement of *m* instances of a *Schedulable Resource Class*. If this attribute is not specified, the resource manager, by default, allocates one instance in its entirety. If both the *AmountRequired* and *Cardinality* attributes are specified on this edge to be *Y units* and *m*, this implies the requirement of *Y units* each of *m instances* of the class.

We now present more concrete cases where *Requires* edges may fall on *Schedulable Resource Classes*. *In Fig. 2,* if *DocumentationTeam1* requires *150 pages/day* of printing capacity of one *Printer* instance and the team is not bothered which particular printer instance, P1 or P2, is acquired for the purpose, then a *Requires* edge from *DocTeam1* to *Printer,* with *AmountRequired* set to *150 units* serves the purpose. The second example is that of *CodingTeam2* requiring *2* licenses of a *CodingTool*. So let the *Unit attribute* of *Coding Tool* be the number of licenses and given that VC++ is the only coding tool available, this would be satisfied by a *Requires* edge from *CTeam2* to *CodingTool* with *Cardinality attribute* set to *2*. If its availability is *2* or more licenses, *2* copies get allocated.

Two other attributes may be defined on *Requires* edges: *Fixed* and *Proportional*. These attributes take boolean values. By default, *Fixed* is not set and *Proportional* is set. Fixed allocation indicates that the capacity required is constant, whereas proportional allocation indicates that the amount required is proportional to that required of the requirer, dedicated to the activity. The following examples describe this further.

Let R1, R2, R3 be three resource elements (classes or instances). Let R1 require *10 units* of R2 with the *PROPORTIONAL attribute* set, and R2 require *5 units* of R3, with *FIXED attribute* set. If Capacity(R1) = *20 units*, and only *10 units* of R1 are required, then only *5 units* of R2 is allocated; and irrespective of what amount of R2 is allocated, a *fixed* amount of *5 units* of R3 is always allocated.

Resource Allocation State

Once a resource is allocated, it is necessary to maintain to which specific task or entity it has been allocated to and by what amounts. The maintenance of such information allows easy manipulation of allocated amounts when the need arises. Examples of such needs may include 1) a greater amount of a resource may be needed to speed up an activity 2) a higher priority pre-emptive request would require dynamic change in allocation state of currently allocated resources. This information may be captured as attributes of *Resource Instances*. Hence, on allocation of a resource*,* two attributes must be set: (i) *AllocatedTo* – this holds a reference to the specific task to which this resource is allocated (ii) *AmountAllocated* – the amount of this resource allocated for the task.

Criteria Assertions

A *Resource Class* may define a set of criteria assertions. A criteria assertion is a *boolean expression* over the attribute values of the resource. These assertions serve as membership criteria for all the children (resource classes or instances) of this class. The additional assertions added at a child *Resource Class* may not contradict those of a parent class. While this is not checkable in general, such

contradictions would make it impossible to populate the resource class with instances, as no instance would satisfy all criteria assertions and therefore could not be added to the model. Therefore, the classes farther from the root, along a certain path of *is-a* links, have more membership criteria to satisfy and thus represent more specific classes than those closer to the root. As an example, one might have a *FastPrinter* class as a child of the more general *Printer* class. The *FastPrinter* class may have with it an assertion that its capacity be greater than *15 pages per minute*. Each resource class *or* instance that is a child of this class should satisfy this criterion. Failure to do so would indicate that this model does not satisfy the semantics that were intended. The *Resource Manager* checks criteria assertions after any of the following: changes to the criteria assertions, changes to attribute values of a resource, or the addition of any resources connected (transitively) with *is-a* links.

Resource Model Editing

To allow for easy creation, editing, reorganization, and visualization of a resource model, we provide a GUI tool, the *Resource Model Editor*. We expect a great deal of dynamic changes to occur to resource entities due to acquiring and releasing of resources before and after tasks. Additionally, we expect some resources to be created, or perhaps destroyed, dynamically during the execution of some activity. For example, a software engineering process produces artifacts, such as design documents, implemented classes, test cases etc. Each of these could be modeled as a resource. To facilitate this, we provide an API that allows tools to dynamically edit the model.

4 RESOURCE MANAGEMENT

The information in a resource model may be used to control resource sharing, plan future usage, or reason about speeding up activity. In this section, we describe how our *Resource Manager* helps support the above. It should be recalled that activities using resources are defined outside of the resource manager. The responsibility of the *Resource Manager* is to provide information about the types and availability of resources and to track their usage. Moreover, the *Resource Manager* could maintain a history of requests for resources that failed, and thus evaluate the need for more resources of a particular type, which as mentioned in *Section 2*, is important in speeding up of an overall activity.

Identifying Resources

To identify all resources that meet a particular need, the client of the resource specifies a resource class name and a query over the attributes of the class. The *Resource Manager,* finds all instances connected transitively via *is-a* links from the named resource class and applies the query to each instance to see if it satisfies the query, and, if so, adds that instance to the set to be returned.

Acquiring Resources

Acquiring a resource allocates the requested resource to the requester immediately. The acquisition of resources by a task is similar to requiring of resources by other resources as described earlier in *Section 3*. For example, a design task *requires* a design team; hence it tries to *acquire* one. Acquisition of a resource for an activity is effected by the specification of a resource instance or class by name and a related query. If the *Resource Manager* can identify a resource that matches the specification correctly, it is locked for use by the given activity. If more than one resource matches, the resource manager *selects* one. *See Section 7* for more on *Resource Selection*.

Since acquisition is similar to the *Requires* association earlier described, two of the attributes of the *Requires* association (*AmountRequired, Cardinality*) may be specified in an acquisition specification. *Fixed and Proportional* attributes are not applicable here though, as acquisition originates from an activity and not a resource. Also, if the acquisition request is for a resource that requires another resource, the acquisition succeeds only if all transitively required resources get acquired successfully.

However, the *Resource Manager* cannot prevent unauthorized use of resources, when their use is made outside of the resource manager. For example, a person may be given an assignment without the resource manager being informed. While this cannot be prevented, it does compromise the *Resource Manager*'s ability to assist in planning and scheduling.

Reserving Resources

Acquisition results in immediate locking of resources. Reservation on the other hand supports the ability to plan for future use of the resource. A reservation request is a specification of the resource class or instance to reserve, a query over the resource's attributes, the *time* the resource is needed and for what duration. The reservation is made at the level as specified in the resource specification. Therefore if the level requested is a *Schedulable Resource Class*, then the reservation is made at the that *class'* level; this way, selection of a particular instance of the *Schedulable Resource Class* may be deferred until acquisition time. This has the potential to increase the overall utilization of the resources since a later request might require a specific resource; hence by not reserving a specific instance at reservation time, it is more likely that a more specific request is satisfied. If however, the request is for an instance that is available and satisfies the query, the instance is reserved till it is used. Reservation may be exclusive or not, similar to acquisition. If the request is for a resource that requires other resources, these other resources are also reserved using rules already stated. This is necessary since reservation of a resource for use in the future is successful, not only when it is acquirable, but also when the resources it requires are acquired successfully.

Releasing of Resources

If a resource is acquired or reserved, it can be made available again for others to use by *releasing* it. A resource is released on the completion of an activity for which it was acquired or reserved. A resource is also released if the duration for which it is reserved expires and it is not acquired for use at this time.

5 OUR EXPERIENCE WITH THIS APPROACH

We have gained experience with our resource manager by integrating it into Little-JIL, a visual process programming language. In this section, we describe how the resource manager meets resource management needs in this application domain.

In Little-JIL [10], a process is represented as a hierarchical decomposition of steps. Attached to each step is a list of resources that are required to carry out that step. Typical resources in Little-JIL include agents (which may be human, software, robots, for example), physical resources (printers, computers, specialized hardware, for example), licensed software (compilers, design tools, word processors, for example), and access permissions for data (documents or portions thereof, for example). In Little-JIL, each step has an agent. From Little-JIL's perspective, the agent is distinguished from the other resources by virtue of the fact that it is the entity with which the Little-JIL interpreter communicates to assign tasks and get results. *From the resource manager's perspective, however, an agent is simply a resource*. Little-JIL assumes that a resource model describing all available resources is defined outside of the process. The resource specifications attached to steps enable the identification, acquisition, and release of resource instances that meet the needs of the step. This binding between specific resource instances managed by our resource manager and specific step instantiations in a Little-JIL process being executed is done dynamically as described in *Section 4*. Assuming the necessary resources exist in the model, the agent is acquired and Little-JIL assigns to it the responsibility for executing the step to which it is bound. The agent might not start executing the step immediately (minutes or even weeks, depending on the nature of the process). To avoid blocking the resources for long periods of time, the Little-JIL interpreter does not acquire the rest of the resources required for the step until the agent indicates that it is starting to execute the step. At this point, if the resources are not available, the resource manager fails to acquire the resources and Little-JIL throws an exception that is handled by a handler designed to deal with exceptions of this sort. Resources may be passed to substeps when those substeps are ready for execution. When the step (and all its substeps) have completed, Little-JIL releases the resources that were acquired for execution of that step.

The process (*see Fig 3*) that used the resource model example shown in *Fig 2*, was a simple multi-user design process built on top of the *Booch Object Oriented Design methodology*. In this process, a design team is given a design task. They perform some initial design activities as a team and then subdivide the assignment into individual design assignments. The design activity follows the steps of 1) identifying abstractions, 2) identifying the semantics of the abstractions, 3) identifying the relationships between abstractions, and 4) implementing the abstractions. Each of these steps is carried out by a human (or team of humans), but some *substeps* are used to check *postconditions* on completion of some of these steps, and these substeps may have software agents. For example, the *IsDocumented* substep invoked after *IdentifyMajorAbstractions*, to check if some documentation exists for each named abstraction, has a software agent. The software agent required here is not licensed software. There is no contention for them, no need to schedule them, and thus they do not need to be part of the resource model. A specific resource instance can also be specified as the agent for a step; however, it is more usual for a specification to name a resource class. In cases where specific resource instances are specified, these resource instances may be introduced into the model to support the specific process. In such cases, these software agents are not the types of general reusable, contended-for resources we normally expect to find in a resource model. They exist simply because the Little-JIL interpreter requires that all agents be treated as resources. Hence, we found that it was sometimes necessary to customize a resource model to support a specific Little-JIL process. Our experience was that such customizations turned out to be simple to do, and did not seem to have any noticeable effect on the operation of the resource manager.

Whole-Part relationships allowed us to model the design team as a 'whole', which is responsible for an entire subproject. As a team they meet to decompose the subproject into major abstractions that can be further designed in (relative) isolation by individual designers.

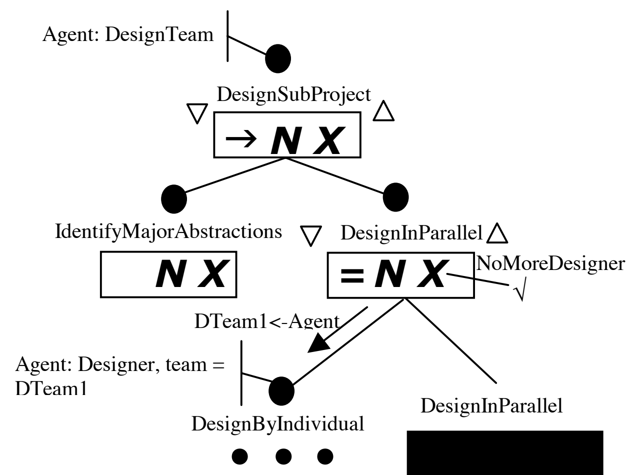

Fig 3. A LittleJIL Process with Resource Declarations

This is represented in the process by specifying a *DesignTeam* to be the agent for the *DesignSubProject* step. At this point the resource manager is asked to acquire all team members, of the assigned team, say *DTeam1* (*see Fig 2*) for the design activity. Once this has been done, the entire team is inherited as the agent for the *IdentifyMajorAbstractions* step. Once this resource has been identified, the *DesignInParallel* step is recursively invoked. The first substep of *DesignInParallel*, the *DesignByIndividual* step, is invoked, and this step requires as an agent a *Designer* who is a member of the *DTeam1*. Here, either *B or C* are allocated. The purpose of this step is to give an individual assignment to an individual member of the team and so their assignments are simply refined to the more specialized tasks at hand.

The motivation behind separating a resource model from the process is to allow a single process to be reused effectively across a range of resource availability scenarios (reusable processes). The process specifies the essential resource requirements using specifications, while the specific instances are bound dynamically based upon what is available in the environment. In addition to supporting substitutable resources, this also allows us to specify a process in which activities can be performed in parallel if sufficient resources exist but need to be done sequentially if there are insufficient resources. This has led to a common Little-JIL idiom of *resource-bounded parallelism* as exemplified by the *DesignInParallel* step where multiple instantiations of a step to be performed in parallel are allowed, with each step getting new resources. When all the available resources have been allocated, a newly instantiated step's request for resources will be denied, an exception will be thrown, and no more parallel instantiations will be created. We have also found *resource-bounded recursion* to be a useful idiom, although it does not occur in this example.

The agents can communicate directly with the resource manager when required. For example, an agent might be in the best position to select which specific resources to acquire, or might want to release resources in some substep rather than waiting for the entire step to complete. This is particularly true for the management of human resources, where one would almost certainly want a human manager to make such decisions as which people should perform which specific assignments. The entire functionality of the resource manager is thus available to agents to refine the use of resources within a process. The *ClassReserver* agent is an example of using the resource manager's functionality to create a process specific acquisition procedure. The *ClassReserver* agent is a GUI tool that enables the human designer to select which classes the designer wants to work on, acquiring those classes for the designer and thereby preventing other designers from working on the same classes simultaneously. Other processes might provide other mechanisms for doing this binding, such as having a human manager specify all class-designer assignments.

6 RELATED WORK

Other resource modeling and specification work has been done in such resource sensitive application areas as software process, operating systems, artificial intelligence planning and management. The approaches in these areas have some similarities to our own work, as they concern themselves with such similar problems as the coordination of activities that can span long time periods.

Related work in Software Process

A number of software process modeling and programming languages and systems have addressed the need to model and manage resources. Among the most ambitious and comprehensive have been APEL [9], and MVP-L [14], both of which have attempted to incorporate general resource models and to use resource managers to facilitate process execution. APEL's approach is similar to ours in that APEL deals with resource management as a separate issue that is orthogonal to other process issues. APEL provides a way to specify an organizational model that includes human resources and their aggregates (teams). It also introduces the notion of a *position* that is very similar to our notion of a resource class, in that it tags human resources according to their skill sets. APEL's roles define the capacity in which a resource is used by a specific activity. APEL's resource modeling approach, however, seems to be less general and comprehensive than our model of resources. In addition it does not seem to incorporate any provision for support of scheduling.

There are a number of other process languages that provide for the explicit modeling of different sorts of resources. MERLIN [5], for example, provides rules for associating tools and roles (or specific users) with a work context (which may be likened to a Little-JIL step). Some others that offer similar limited capabilities are ALF [4], STATEMATE [8], and ProcessWeaver [1].

In all of these cases, however, the sorts of resources that are modeled are rather limited in scope.

Related work in Operating Systems

The problem of scheduling resources has been extensively studied in the field of operating systems. The most common resources in this problem domain include peripheral devices and parts of code or data that require exclusive access. The differences between the needs of resource management in operating systems and software engineering (or artificial intelligence) arise from the the following facts about operating system resources: (i) Resources are used for much shorter periods of time (hence more elaborate notions of availability are not usually needed). (ii) Resources are generally far less varied (e.g., humans are not considered resources in this domain). (iii) Resource usage is much less predictable. Independent programs

compete for the same resources and are executed at unpredictable times.

Operating systems research on resource management per se, usually focuses on scheduling techniques of a specific resource (such as a CPU or a hard disk as in [6] and [16]). The similarities in purposes of resource modeling between software engineering and operating systems fields, appear only in research on admissions control ([3], [2], [15], [12]) in networking and both of these fields require a resource environment abstraction that can model several kinds of resources. They also require a way to satisfy a general resource request (as opposed to the request for a specific instance). This means that a search mechanism is needed in both cases. The resource modeling approaches in the field of networks research are somewhat similar to our approach in that they also often introduce a hierarchy of resources and provide some functionality (i.e., operations for search, reservation and acquisition of resources). It is interesting to note that the authors of [3] and [2] also saw the need to make the resource model independent from the model representing tasks (applications in their terminology). Resource models in this domain, however, seem to lack flexibility and generality.

Related work in AI Planning Systems

Probably, the closest resource modeling approach to ours is suggested in the DITOPS/OZONE system. OZONE is a toolkit for configuring constraint-based scheduling systems [17]. DITOPS is an advanced tool for generation, analysis and revision of crisis-action schedules that was developed using the OZONE ontology. The closeness is evidenced by the fact that OZONE also incorporates a definition of a resource, contains an extensive predefined set of resource attributes, uses resource hierarchies, offers similar operations on resources, and also resource aggregate querying. We believe that our resource modeling approach places a greater emphasis on human resources in the predefined attributes and allows for an implementation that is easier to adapt to different environments.

The Cypress integrated planning environment is another example of a resource-aware AI planning system. It integrates several separately developed systems (including a proactive planning system (SIPE-2 [18]) and a reactive plan execution system). The ACT formalism [11] used for proactive control specification in the Cypress system, has a construct for resource requirements specification. It allows the specification of only a particular resource instance. The resource model does not allow for resource hierarchies and the set of predefined resource attributes is rigid and biased towards the problem domain (transportation tasks).

Related work in Management

An example of a resource modeling approach in a management system is presented in the Toronto Virtual Enterprise (TOVE) project [7]. This approach suggests a set of predefined resource properties, a taxonomy based on the properties and a set of predicates that relate the state with the resource required by the activity. The predicates have a rough correspondence to some methods of our resource manager. It is very likely that our resource manager would satisfy the functionality requirements for a resource management system necessitated by the activity ontology suggested in the TOVE project.

Related work in other distributed software systems

The Jini distributed software system, developed by Sun Microsystems, seems to employ a resource modeling approach that seems somewhat similar to ours. The Jini system is a distributed system based on the idea of federating groups of users and the resources required by those users. The overall goal of the system is to turn a network into a flexible, easily administered tool on which human and computational clients can find resources. One of the end goals of the Jini system is to provide users easy access to resources. Jini boasts the capability for modeling humans as resources, allows for resource hierarchies, and provides ways to query a resource repository using a resource template that is very similar to resource queries in our suggested approach.

7 EVALUATION AND FUTURE WORK

In an earlier paper [13], we describe some experience we have had in applying an earlier version of this resource management system. Most of this experience has come from integrating this system with the Little-JIL process modeling system [10]. We have used this system to model processes for robot coordination, negotiations, information warfare scenarios, data mining and software engineering processes such as collaborative design and regression testing. This experience has confirmed that the features described in this paper are of substantial value.

Future work includes addition of a tracking mechanism into the *Resource Manager* that would help draw inferences about projected resource needs and utilization. A mechanism of this sort would enable users to gauge the gains and losses that would be likely to result from making changes in resource availability, or from making changes to the activity itself.

Also, our resource querying mechanism needs to be able to find “best-match” resources, based on user-specified queries. Despite having this, it is expected that on several occasions, multiple instances in the model may satisfy the query. At this point, we need some kind of mechanism to either pick one of many instances or help decide which the best instance is. This has prompted us to think of developing and embedding in the resource manager, various *resource selection processes*. These processes may vary from simple “pick-any-one” processes to more complex negotiation processes, depending on what exactly is required by the activity.

Finally we need to gain more experience in the use of the resource manager to support planning and coordination systems. To this effect, we are continually developing processes to evaluate how well the resource management system meets the needs of activity coordination.

ACKNOWLEDGEMENTS
This research was partially supported by the Air Force Research Laboratory (AFRL)/IFTD and the Defense Advanced Research Project Agency (DARPA) under contract F30602-97-2-0032. The views and conclusions contained herein are those of the authors and do not represent the official policies or endorsements either expressed or implied, of DARPA, AFRL/IFTD, or the US Government.

REFERENCES
1. C. Fernstrom. Process Weaver: Adding Process Support to UNIX. *The Second Intl. Conf. on the Software Process*, pages 12-26, 1993.
2. S. Chatterjee and J. Strosnider. A Generalized Admissions Control Strategy for Heterogeneous, Distributed Multimedia Systems. *Proc. of ACM Multimedia,* San Francisco, CA, 1995.
3. S. Chatterjee, J. Sydir, B. Sabata, and T Larence. Modeling Applications for Adaptive QoS-based Resource Management. *Proc. of the 2nd IEEE High Assurance System Engg. Workshop*, Bethesda, Maryland, 1997.
4. G. Canals, N. Boudjlida, J.C. Derniame, C. Godart, and J. Lonchamp. ALF: A framework for building process-centered software engineering environments. *Software Process Modeling and Technology*, pages 153-185. Research Studies Press, Ltd., Taunton, Somerset, England, 1994.
5. G. Junkermann, B. Peuschel, W. Schafer and S. Wolf. MERLIN: Supporting cooperation in software development through a knowledge-based environment. *Software Process and Technology*, pages 103-129, Research Studies Press, Ltd., Taunton, Somerset, England, 1994.
6. P. Goyal, X. Guo, and H. Vin. A Hierarchical CPU Scheduler for Multimedia Operating Systems. *Proc. of the 2nd Symp. on Operating Systems Design and Implementation*, pages 107-122, Seattle, Washington, 1996.
7. M. Gruninger and M.S. Fox. An Activity Ontology for Enterprise Modeling. *Submitted to AAAI-94, Dept of Industrial Engg., Univ. of Toronto*, 1994.
8. D. Harel, H. Lachover, A. Namaad, A. Pnuelli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Trans. on Software Engg.*, 16(4):403-414, 1990.
9. J. Estublier, S. Dami, and A. Amiour. APEL: A graphical yet executable formalism for process modeling. *Automated Software Engg.*, 1997.
10. B.S. Lerner, L.J. Osterweil, S.M. Sutton Jr., and A.Wise. Programming process co-ordination in Little-JIL. *Proc. of the 6th European Workshop on Software Process Technology*, Number 1487 in Lecture Notes in Computer Science, pages 127-131, Weybridge, UK, 1998.
11. K.L.Meyers and D.E. Wilkins. The ACT Formalism. Working Document:Ver2.2, *SRI, Artifical Intelligence Center*. 1997. http://www.ai.sri.com/act/act-spec.ps
12. K. Nahrstedt and R. Steinmetz. Resource Management in Networked Multimedia Systems. *IEEE Computer,* pages 52-64, 1995.
13. R.M. Podorzhny, B.S. Lerner, L.J.Osterweil. Modeling Resources for Activity Coordination and Scheduling. *Proc. of the 3rd Internal Conf. on Coordination Models and Languages*. Springer-Verlag, 1999.
14. H. Rombach and M. Verlage. How to assess a software process modeling formalism from a project member's point of view. *2nd Intl. Conf. on the Software Process*, pages 147-159, 1993.
15. S. Floyd and V. Jacobson. Link Sharing and Resource Management Models for Packet Networks. *IEEE Trans. on Networking*. 3(4):365-386, 1995.
16. P. Shenoy and H. Vin. CELLO: A Disk Scheduling Framework for Next-Generation Operating Systems. *Proc. of ACM SIGMETRICS*, 1998.
17. S.F. Smith and M.A.Becker. An Ontology fo constructing Scheduling Systems. *Workig Notes from AAAI Spring Symosium on Ontological Engg.*, Stanford, CA, 1997.
18. D.E.Wilkins. Using the SIPE-2 Planning System: A Manual for Ver4.17. *SRI Artificial Intelligence Center,* Menlo Park, CA, 1997.