Modeling Resources for Activity Coordination and Scheduling

Rodion M. Podorozhny , Barbara Staudt Lerner and Leon J. Osterweil

University of Massachusetts, Amherst MA 01003, USA

Abstract. This paper describes experience in applying a resource management system to problems in two areas of agent and activity coordination. In the paper we argue that precise specification of resources is important in activity and agent coordination. The tasks and actions that are to be coordinated invariably require resources, and the scarcity or abundance of resources can make a considerable difference in how to best coordinate the tasks and actions. That being the case, we propose the use of a resource model. We observe that past work on resource modeling does not meet our needs, as the models tend to be either too informal (as in management resource modeling) to support definitive analysis, or too narrow in scope (as in the case of operating system resource modeling) to support specification of the diverse tasks we have in mind.
In this paper we introduce a general approach and some key concepts in a resource modeling and management system that we have developed. Although rigorous and complete specification of the model and system is beyond the scope of this paper, the descriptions provided suffice to support explanation of two experiences we have had in applying our resource system. In one case we have added resource specifications to a process program. In another case we used resource specifications to augment a multiagent scheduling system. In both cases, the result was far greater clarity and precision in the process and agent coordination specifications, and validation of the effectiveness of our resource modeling and management approaches. A range of future work in this area is indicated at the conclusion of the paper.

Keywords

Resource modeling, resource management, process programming, planning, agent coordination

1 Introduction

Resources are essential to the performance of any task. Indeed, it is not unreasonable to adopt as a working definition that a resource is any entity or agent that is needed in order to perform a task, but for which there is only a limited supply available. That being the case, it is reasonable that specifications of needed resources be present in specifications of the way in which tasks are to be done, and in attempts to automate support for such tasks.

The need for the specification of resources has been realized by other communities. Resource specifications are important in operating systems, where contention for resources can lead to such dangerous situations as deadlock and starvation. Thus, operating systems incorporate models of key resources, keep track of resource utilization and analyze utilization to either recognize or avoid such pathologies as deadlocks. Resource specifications are important in management as well, where they are critical to the effective scheduling of tasks to individuals and granting of access to, and utilization of, scarce resources to support the execution of tasks.

More recently workflow, software process, and activity coordination research has reconfirmed the importance of resource specification. One key goal of systems that support the modeling and execution of processes is to expedite the performance of complex tasks. Early systems in this area have focused particularly strongly on the structure of these tasks. Analysis of task structures has been shown to be useful in indicating possible parallelization of activities, for example. But it has become increasingly clear that such analyses are handicapped if they are not supplemented by consideration of resources.

Thus, for example, a simple structural analysis of a process specification may seem to indicate that two tasks may be parallelizable. But if both require access to a single resource that cannot be shared, then parallelization of the tasks is impossible. As another example, a project is likely to benefit if the most experienced or competent performers are assigned to the most critical tasks. Only analysis and evaluation of available human resources can help here.

These realizations have sparked an interest in resource modeling, specification, and analysis by the community of researchers working on process, workflow and activity coordination technology. Study of the existing literature in the area of resource specification has indicated that the existing work is largely insufficient to meet the needs of these newer areas. Management resource modeling seems too informal to support the needs of workflow and process. Models in this area seem to be largely intended to support humans who are expected to make judgments about resource allocation that may be mostly intuitive. Operating system resource modeling is, in contrast, quite rigorous and precise. The rigor and precision of models in this area is necessary as they are used to support automated analyses and reasoning. The resources about which such reasoning is carried out are, however, rather limited in scope and nature. Generally operating systems reason about hardware resources such as storage devices, and blocks of central memory.

Resource models for use in workflow, process, and activity coordination are intended to support automated reasoning and analysis about the full spectrum of resources that may be needed in any activity to be modeled. As the span of applicability of these systems is very broad, they seem to require at least the breadth and diversity of scope that characterizes management resource models. But as the goals of such systems include the ability to reason definitively about such issues as deadlock, and optimal allocation, the rigor and precision that characterizes operating system resource modeling also seems necessary here.

In this paper we describe early experiences with such a system. Our resource specification and modeling system is designed to support the representation of an unusually broad spectrum of resource types with precision and rigor that seems sufficient to support powerful reasoning and inference.

Space limitations prevent the presentation of the details of our approach (these are contained in a companion paper that is in preparation). But we do summarize key features of this system and indicate why we believe our early experiences with it support our belief in the value of our approaches.

2 Overview

Our resource management system is intended to be a component that should be useful and usable in any one of a variety of larger systems that may, for example, support processes execution, carry out reasoning about real-time systems, or perform multiagent planning. Since we believe that the need for powerful and precise resource management is widespread we have designed this component to support the modeling of a wide range of types of resources, from physical entities such as robots, to electronic entities such as programs or data artifacts, and to human entities. The resource management component similarly does not prescribe specific protocols about how resources should be used, but rather leaves the definition of those protocols to the external system with which it is to be integrated. In this section, we present an overview of our resource management component, introducing some key terminology. A detailed and rigorous presentation of our resource management system cannot be presented here due to space limitations. This is the subject of a separate, companion paper that is to appear shortly.

A **resource model** is a model of those entities of an environment that may be required, but for which an unlimited supply cannot be assumed. The resource model is organized as a collection of resource instances, resource classes, and relations among them.

A **resource instance** is a representation of a unique entity from the physical environment, such as a specific person, printer, or document. Each resource instance is described using a set of typed attribute-value pairs. There are predefined attributes required of all resources, such as a name and description, as well as user-defined attributes that are specific to the various types of resources. The attribute values of a resource serve to identify the resource and distinguish it from other similar resources. For example, a printer might have an attribute indicating whether it produces color or black-and-white output, but that attribute would not be required of resources that are not printers. A human might have an attribute that represents his or her level of skill in coding in a particular language, while hardware devices would not have this attribute.

A **resource class** represents a set of resources (other classes and/or instances) that have some common attributes. The resource classes in a resource model form a singly-rooted DAG. The root of the DAG is the predefined resource class named `Resource`. Each child class inherits all the attributes of its parent,

but presumably adds in some additional attributes. Each child class name adds a qualifier to its parent class name.

There are two kinds of resources classes: schedulable and unschedulable. A **schedulable resource class** is ordinarily a collection of resources, all of which can actually be allocated to tasks. These instances are generally sufficiently similar that it can be expected that they are ordinarily substitutable for each other. For example, `Laser-Printer` would probably be a schedulable resource class in most circumstances, as laser printers generally offer the same sorts of printing capabilities. Thus, a **schedulable resource class** can be viewed as a conceptual generalization that groups a number of resources with very similar capabilities and performance characteristics. It would make sense to introduce a **schedulable resource class** only if it is expected that there can be two or more very similar resource instances to be generalized.

An **unschedulable resource class** is more abstract and is intended more as an organizational convenience when defining the resource model. For example, the distinguished root `Resource` is not schedulable, and the class `Hardware` is generally not likely to be schedulable either. While it is ordinarily expected that at least one of the children of an **unschedulable resource class** will not be a resource instance, efficiency considerations may sometimes dictate that all children of an **unschedulable resource class** be resource instances. An example of this is given in the next section.

A resource model also contains three relations that connect the resources classes and instances in the model: the **IS-A** relation, the **Requires** relation, and the **Whole-Part** relation.

The **IS-A relation** was hinted at above. It is the relation that defines pairs of classes (or class-instance pairs) that share sets of attributes. In particular, the attributes of the parent in the **IS-A relation** are all inherited by the child in the relation.

The **Requires** relation connects one resource instance or resource class to another to indicate that some particular resource class, or a member of some particular resource class, is always required in order for the first to be useful. For instance, a particular piece of software might require a computer with a particular operating system or with some minimum memory requirements. Use of this relation dictates that these dependency requirements are to be universally true, independent of any particular application in which the related resources are to be used.

A **Whole–Part** relation connects resources that may at times need to be considered part of an aggregate resource in addition to being considered as individual resources. For example, individual developers are separate resources, but may also at times need to be considered to be part of a development team. An example of this, arising from our experience, is described in a later section of this paper.

We have taken resource models represented as just described and built around them a resource management system. The purpose of this system is to support applications that need support for allocating resources and keeping track of

current allocation status. This system is built upon the use of four primary operations on the resource model:

- **Identification**, which identifies specific resource instances that can satisfy specific stated requirements. Requirements can be expressed either as a specific resource name (which could be a class or instance name), or by queries over the attribute values of the resource class that is required. In either case, the requirement specification can also include an amount of time for which the resource is requested.
- **Reservation**, which reserves a specific resource instance (generally one that has previously been identified).
- **Acquisition**, which locks a resource instance for use in a specific activity. The resource instance is generally one that has been previously reserved or one identified with a resource specification.
- **Release**, which frees a reserved or acquired resource instance so that it can be used by other activities.

Reservation and acquisition do not necessarily require exclusive use of a resource instance. Instead, they can specify the quantity or fractional usage of a resource instance that they will require. For example, a person might be required to work on a specific assignment for 10 hours/week for 4 weeks. The remaining capacity can be allocated to other activities that request it.

The remainder of this paper describes our experiences integrating our resource management component into two systems: a process programming execution system and an artificial intelligence multiagent planning system.

3 Experiences with our approach

We have gained experience with our resource manager by integrating it into two different systems. The first is Little-JIL, a visual process programming language. The second is MASS, a multi-agent planning system. In this section, we describe how the resource manager meets resource management needs in these two application domains.

3.1 Integration with Little-JIL

In Little-JIL ([9], [17]), 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 execution 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 execution agent. From Little-JIL's perspective, the execution 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 execution 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 follows.

The resource declarations attached to each step identify the name of a resource class or instance and, optionally, a query defined in terms of the attribute values of the desired instance.[1] When a step is first considered for execution, all resources required by the step are *identified*. If the resource manager cannot find a matching resource in the resource model, an exception is thrown indicating this and control flows to an exception handler defined for the Little-JIL program. Assuming the necessary resources exist in the model, the execution agent resource is acquired and Little-JIL assigns to it responsibility for executing the step to which it is bound. The execution agent may have several steps assigned to it and therefore might not actually start the step for some time (minutes or even weeks, depending on the nature of the process). In order to not tie up the resources for an unnecessarily long period of time, the Little-JIL interpreter does not actually *acquire* the rest of the resources required for the step until the agent indicates that it is starting the step. At that point, it is possible that the necessary resources are being used elsewhere. In this case, the resource manager indicates its inability to acquire the resources and Little-JIL throws an exception that again 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.

A Static Resource Model Figure 1 shows a resource model used within a simple Little-JIL process that specifies the coordination of a team of people who are carrying out an object oriented design activity. This model defines six unschedulable resource classes used to organize the model: `Person`, `Group` (of people), `Software`, `Software Agent`, `Software Tool`, and `Computer`. Person is subdivided into two schedulable classes: `Manager` and `Designer`. There are three instances beneath `Manager`: `Amy`, `Bill`, and `Carol`. There are four designer instances: `Carol`, `Frank`, `Dave`, and `Emily`. Carol apparently has both managerial and design expertise and thus can serve in either role. The figure shows only the names of the resources, but not their other attribute values. The attributes associated with `Person` include such things as Salary and Years of Experience. The `Designer` class might add additional attributes such as Domains of Expertise.

[1] Currently these queries are written in Java, but ultimately we expect to provide a query language that will allow in-place resource specifications directly in the process definition.

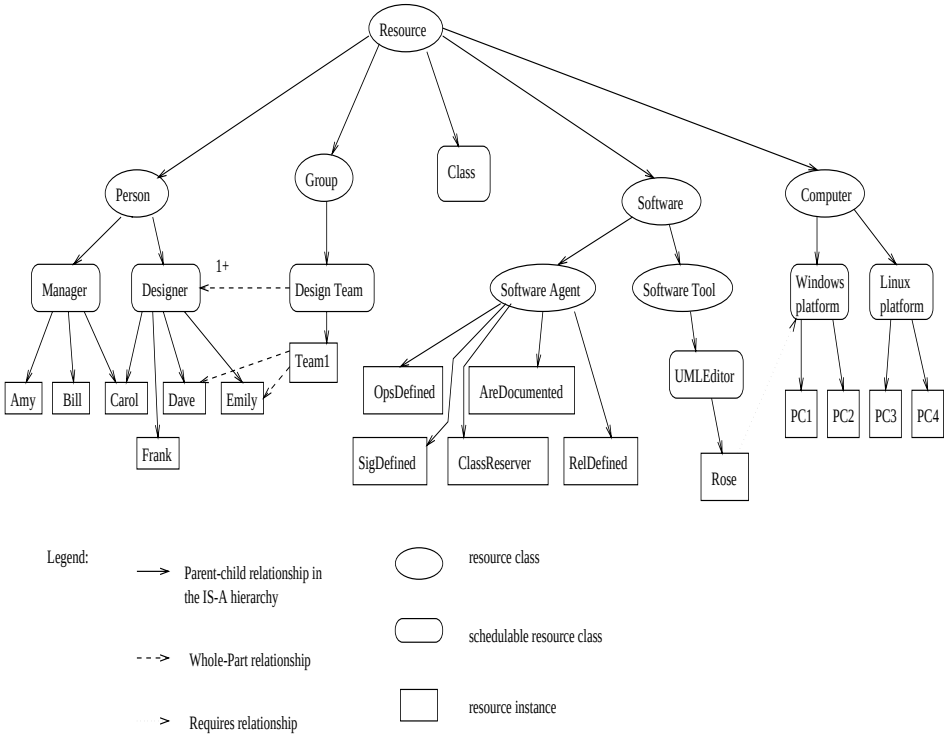

Fig. 1. Resource Model used in a Little-JIL Design Process

Most of the edges in the resource model represent IS-A relations. There is a Requires relation between `Rose`, an instance of `UMLEditor`, and the schedulable resource class that represents PCs running Windows. This indicates that any step that uses `Rose` also requires a PC running Windows. Because the Requires edge exists, a step that uses `Rose` does not need to declare its need of a Windows PC in any other way.

This example also illustrates an important interplay between the different types of relations. Were it not for the Requires relation just mentioned, it might be reasonable to treat `Computer` as a schedulable resource class having four instances. But the need to indicate the dependence of the `Rose` software resource on a particular kind of platform creates a need for the schedulable resource class `Windows-Platform` as a subclass of `Computer`, which at that point is most reasonably considered to be an unschedulable class.

The alternative of having the four computer platform instances all be children of the `Computer` unschedulable resource class requires that the operating system simply be an attribute attached to the instances. There are several reasons not to do that, however. First, if we had done that, it would not be possible to specify in the resource model that `Rose` required a computer running Windows without binding it to specific computers, which is overly restrictive. If we choose not to

represent the requires relation in the model, the process programmer would be forced to add the Windows computer as a resource at each step that specified `Rose`. Furthermore, if we had another `UMLEditor` that ran on Linux, the resource specification would become more complicated in order to ensure that compatible software licenses and computing platforms were available when the step began.

The resource model also includes a Whole-Part relation between the `Design Team` class and the `Designer` class, indicating each design team contains one or more designers. The `Team1` instance is therefore required to have one or more designers as parts. In this case, it has two members, `Dave` and `Emily`. It is possible that the initial resource model might not include any specific teams. The teams could be created during process execution, creating both the Team instances as well as the Whole-Part edges identifying the team members. In the figure, we have shown the team composition, which may have been created statically before execution of the process, or may be thought of as a snapshot of the resource model after the relation has been created dynamically.

Figure 1 also shows two types of software: software agents and software tools. A software agent is a piece of software that can be assigned the responsibility of performing a step of the process, whereas a software tool is a licensed tool that a user requires in order to complete a step. Thus, the software agents are resources required directly by the Little-JIL interpreter, while software tools are resources required by the human agent carrying out a process step.

The remaining resource class is named `Class`. This class represents a data artifact, namely the collection of the designs of the various individual classes that are created as the products of the activity of designing the particular application system that is being specified by this process. These application classes are represented as resources instances, and represented in our resource model, in order to support coordination of the activities of the designers of the application. Specifically, as the design of the application proceeds, individual classes of the design being created will have to be acquired as resources for various substeps of the individual process steps, in order to make them available to the individual designers who are the execution agents of those steps. In this way, the resource manager can assist in case it is desired that multiple designers do not all work on the design of the same class. Different processes could, of course, allow designers to work on the same class design collaboratively, or even require that different designers must work on the same class collaboratively. Our resource modeling and management capability can be used to support all of these situations. It should be noted that the resource model shows no resource instances as children because this example assumes that the various instances are to be created dynamically and the design process has not yet progressed to the point of creating any of them.

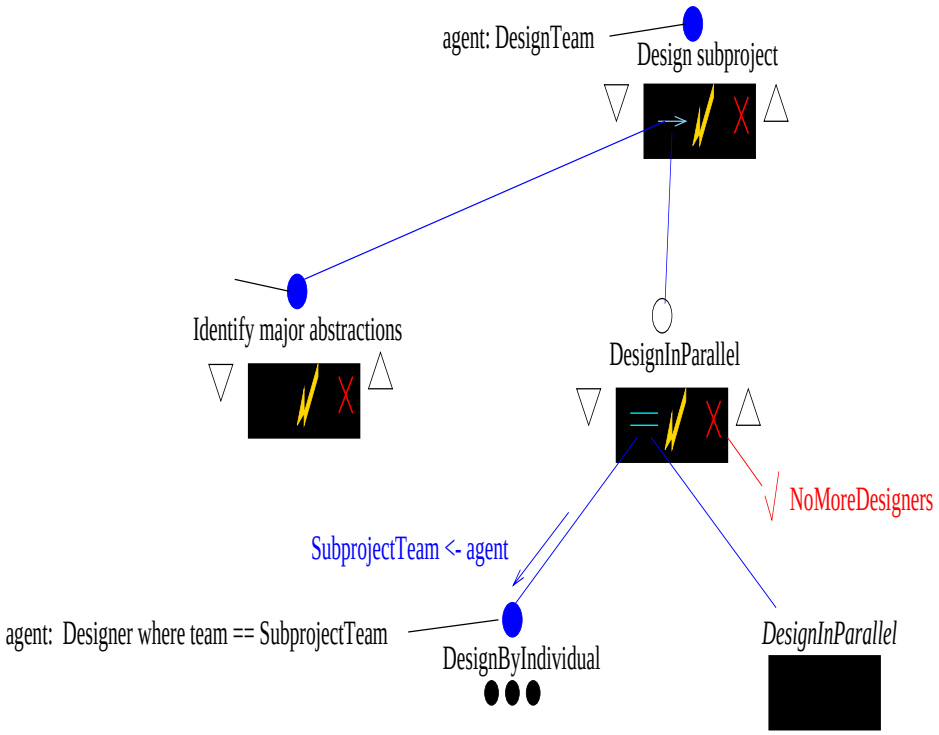

Fig. 2. A Little-JIL Process with Resource Declarations

Dynamic Interaction between a Little-JIL Process and the Resource Manager The process that used this resource model was a simple multi-user design process built on top of the Booch Object Oriented Design methodology ([1], [13]) as shown in Figure 2. 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 four step Booch methodology 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 post-conditions on completion of some of these steps, and these substeps may have software systems as their execution agents. For example, the AreDocumented substep (Figure 1) is invoked after identifying the abstractions to check that some documentation exists for each named abstraction. This substep is to have a software system as its execution agent.

Some comments on this example seem to be in order. First, we note that software systems often have a distinctive nature in the resource model as these systems are often available in sufficient abundance that scheduling of them is not required. Such is the case in this example, and that explains why requests

for them are not shown in this Little-JIL process program. The software systems required here are 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.

It is also worth noting that in this example we see a case where a specific resource instance is specified as the required execution agent for a step. 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 (or even created for) 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. Because of this 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.

The Whole-Part relation between a team and its members is critical in expressing the notion of team and individual responsibilities commonly found within organizations. The design team as a whole 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. This is represented in the process by specifying a `Design Team` to be the agent for the `Design Subproject` step. At this point the resource manager is asked to acquire all team members for the design activity. Once this has been done, the entire team is inherited as the agent for the `Identify Major Abstractions` step. Once this resource has been identified, the `Design In Parallel` step is recursively invoked. As the first substep of `Design In Parallel`, the `Design By Individual` step is invoked, and this step requires as an execution agent a `Designer` who is a member of the `Design Team`. The purpose of this step is to give an individual assignment to an individual member of the team.[2] Since the design team members have already been acquired by a parent step, they are not re-acquired here. Instead, their assignments are simply refined to the more specialized tasks at hand. This is done by making the members of the design team be the entire resource model that is available for reservation and allocation to this step. The Whole-Part relation is essential to guiding the resource manager in doing this.

The design process demonstrates another feature that we have found to be a particularly effective use of a resource model to support the definition of reusable processes. 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. 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 per-

[2] Note that we have elided many details in order to focus on the essential resource management issues of interest here.

formed 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 `Design In Parallel` step. Resource-bounded parallelism allows multiple instantiations of a step to be performed in parallel, 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.

In addition to the interaction between the resource manager and the Little-JIL interpreter, the agents themselves can communicate directly with the resource manager when the default identification/acquisition/release semantics provided by the interpreter are insufficient. For example, an agent, rather than the process program, 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.

3.2 Integration with a Multi-Agent Planning System

The second resource sensitive system that was used to evaluate the resource manager is MASS [14], a multi-agent planning system. The resource manager was used in an application of MASS to the task of housekeeping. Although the MASS computational model is significantly different from that of Little-JIL, the resource manager was still found to be useful.

In MASS, as in Little-JIL, a process is considered to be a collection of steps, each of which is executed by an execution agent. But in MASS, unlike Little-JIL, the set of execution agents is established at the beginning of the process, the set of steps is also established, but rather than having a step interpreter bind execution agents to steps, the agents identify the steps themselves and bind them as their tasks. As a result, it is the individual tasks (steps) that are contended for, rather than the execution agents (which do the contending). Thus, the execution agents do not need to be represented using schedulable resource classes in our resource model. They are regarded as being fixed from the outset and therefore it suffices to consider them as individual resource instances. They are not even depicted in our model.

The collection of resources that the agents need is also fixed from the outset. While their availability cannot be assured at any time, their existence is never in doubt. Thus, the use of the `Identify` method of our resource manager was not needed. Thus, in MASS agents acquire resources when they need them, release them when they are done, and react appropriately if they are not available when the agent wants them.

We now briefly outline an example of the use of the resource manager for a specific application of MASS. In this example, the agents are a dishwasher, a washing machine, a vacuum cleaner, an air conditioner, and a heater. The resource model for this example is shown in Figure 3.

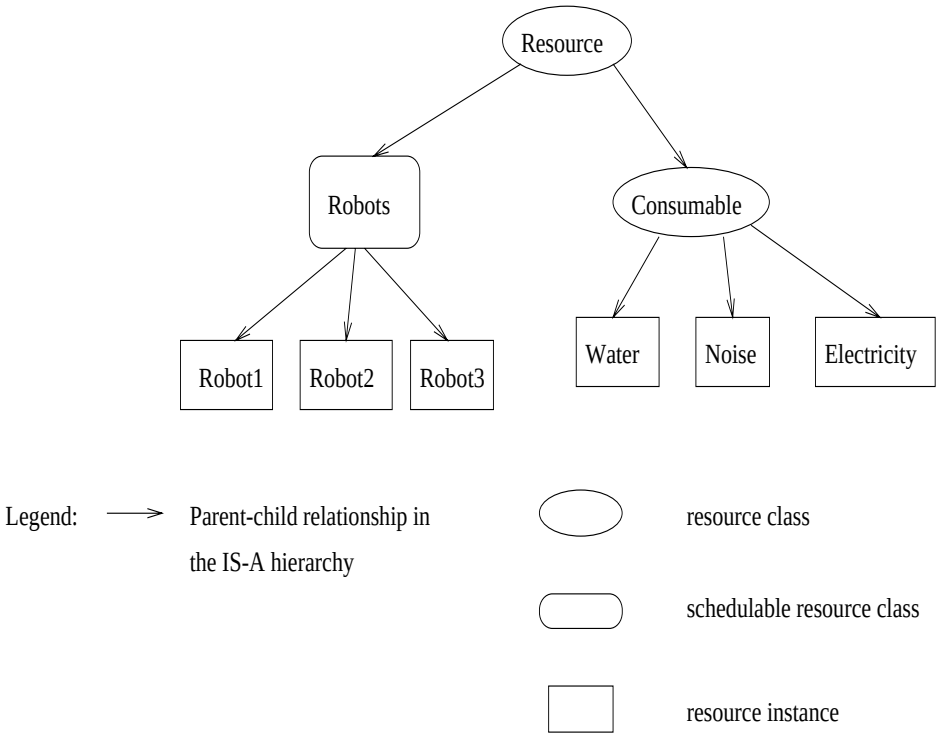

Fig. 3. Resource Model for the Housekeeping Task

While the MASS system does not require the use of many capabilities offered by the resource management system, it did require augmenting the resource management system by adding a rather different type of resource, namely consumable resources. Consumable resources as those that have a limited capacity, but may be shared by different agents up to that capacity level. The resource may either be exhausted through use, or may be regenerated either instantaneously or over time. In our example there are three consumable resources: water, allowed noise,

and electricity. All are instantaneously regenerated when released by an agent. In the example, depending on the capacity of the water flow, it might or might not be possible to operate the dishwasher and washing machine at the same time. Similarly, this application limits the amount of noise that can be produced at one time. It is possible that an agent will not be able to execute at a particular time because it would raise the noise volume above the desired threshold.

We were able to model consumable resources by adding to them a capacity attribute. For example, the capacity of water might be expressed in gallons per minute. When an agent acquires a resource, it indicates how much capacity it needs. If there is sufficient capacity remaining, it is given to that agent. Otherwise, an exception is thrown so the agent can decide whether to wait or to work with a lower capacity. When the agent is done with the resource, it releases the capacity it was using.

This particular resource model is an approximation of the resource environment. A more accurate representation would have also modeled the access points, such as faucets or electrical outlets, as resources that were contended for. This model also assumes a constant capacity for the resources. Some consumable resources are more accurately represented as resources whose capacity decreases until an agent replenishes the resource. An example of this would be paper for a printer or gasoline for a car.

The following is an informal description of a typical task that MASS is supposed to solve. Let us assume it is 16:00 now. In a two hour period the following goals have to be accomplished to prepare for the arrival of guests:

- The dishes have to be cleaned
- The tablecloth has to be washed
- The floors need to be cleaned
- Room temperature needs to be at 68 degrees when the guests arrive

Constraints:

- The noise level should be low between 17:00 and 17:30 because the baby is napping.

The MASS system orchestrates activities performed by the agents taking into account resource and environmental constraints. Every agent produces its own local schedule. An agent informs other agents of its schedule and conducts negotiation if there are any conflicts over resources. Resource conflicts can occur if the agents, as a group, intend to use more capacity of a particular resource during a particular time period than is available in the environment. In the case of contention for a resource, agents negotiate the use of the resource and decide which one of them is going to get access to the resource during this time. Once the decision is made the local schedules of the agents involved are correspondingly updated to exclude the contention.

This example, combined with the Little-JIL example, demonstrates the flexibility provided by the resource manager. The types of resources modeled and the ways in which the resources are used are quite different in the two cases. Even

so, the resource manager was able to address all these concerns and operate effectively in these very different environments.

4 Related work and other Approaches

High adaptability to new environments, the use of a predefined, yet extensible, resource class abstraction (i.e. through the use of our IS-A hierarchy) and ease of integration are some key characteristics that differentiate our work from previous approaches to resource modeling in resource sensitive systems.

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 these areas concern themselves with such similar problems as the coordination of activities that can span long time periods.

4.1 Related work in software process

There have been a number of software process modeling and programming languages and systems that have addressed the need to model and manage resources. Among the most ambitious and comprehensive have been APEL [8], and MVP-L [11], both of which have attempted to incorporate general resource models and to use resource managers to facilitate process execution. We believe that these systems do not support resource modeling with sufficient rigor, precision, and generality. There are a number of other languages that provide for the explicit modeling of different sorts of resources that seem to fit nicely into our larger resource modeling capability. 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 [7], and ProcessWeaver [2]. In all of these cases, however, the sorts of resources that are modeled are rather limited in scope.

4.2 Related work in operating systems

The problem of scheduling resources has been extensively studied in the field of operating systems ([3], Chapter 6.4). 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 fact that operating system resources:

- are generally all resource instances, and hence there is little need for resource hierarchies.
- are used for much shorter periods of time (hence, more elaborate notions of availability are not usually needed).

– are generally far less varied (e.g. Humans are not considered to be resources in operating systems), while process and AI systems must consider far broader classes of resource types.
– are generally far more static. Typically it is necessary to reboot an operating system in order to add resources.

As a result operating systems resource management systems are of only limited applicability to our needs.

4.3 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 [12]. **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 (the SIPE-2 planning execution [16], PRS-CL, etc.) The ACT formalism [10] 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).

4.4 Related work in management

An example of a resource modeling approach in a management system is presented in the Toronto Virtual Enterprise (**TOVE**) project [6]. This approach suggests a set of predefined resource properties, a taxonomy based on the properties[3] and a set of predicates that relate the state with the resource required by the activity [4]. 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.

[3] Properties include Divisibility (this property can take two values - Consumable or Reusable), Quantity, Component (part-of relationship), Source

[4] Predicates include: *use(state,activity)*, *consume(state,activity)*, *release(state,activity)*, *produce(state,activity)*, *quantity(state,resource,amount)*

4.5 Related work in other distributed software systems

The **Jini** distributed software system [15], which is currently being 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 resources can be found by human and computational clients. 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, provides ways to query a resource repository using a resource template that is very similar to resource queries in our suggested approach. Because information about **Jini** is limited it is difficult to say what kind of a resource model is used. It is also difficult to see how easily **Jini's** resource model can be adapted to new environments.

5 Evaluation and Future Work

These two applications of the resource system confirmed that the features of the system we have developed are of substantial value, and that the approaches we are taking seem appropriate. Our experiences have resulted in the creation and modification of our initial notions and decisions.

For example, the `Whole-Part` relation was incorporated into the resource modeling capability after the need became apparent in trying to address the problem of programming the design of software by teams.

In addition, the MASS example showed the need to treat acquisition and release of consumable and replenishable resources differently from reusable ones. The IS-A hierarchy of the resource model was modified to include the Consumable class and an attribute describing this property for every resource instance was introduced. In addition to that, the semantics of acquisition and release of a consumable resource was modified to decrease the capacity of a consumable resource after it was used by a consumer (e.g. dishwasher for water resource), increase it after it was used by a producer (e.g. pump for water resource) and allow several agents to use the same consumable resource for the same period of time if there is enough capacity.

Our experiences have encouraged us to continue to develop our resource model and resource management system further. Next we plan to introduce features that would ease definition of the resource model and resource requirements. The process of definition of the resource model can be facilitated by a resource specification language and a GUI. The resource specification language would also enforce rigor in definition of resources. The GUI to support definition of a resource model would provide a user-friendly way to change or modify the resource model. A resource requirement specification language would allow a non-programmer to specify the requirements (currently the queries are specified in Java). We believe that this language should be orthogonal to the activity

specification language so that it would be able to enhance an arbitrary resource sensitive activity specification.

Finally, we are currently completing work on more complete, precise, and rigorous specifications of the resource modeling formalism, and the resource manager.

6 Acknowledgments

We wish to thank Stan Sutton, Eric McCall, Sandy Wise and the members of the Software Process team of the University of Massachusetts Laboratory for Advanced Software Engineering Research (LASER) for their many helpful comments and suggestions, and for their assistance with the Little-JIL evaluation. We would also like to thank Regis Vincent for his assistance with the MASS evaluation. Finally, we acknowledge the support of the Defense Advanced Research Projects Agency, and Rome Laboratories for their support of this research under grant F30602-94-C-0137 and F30602-97-2-0032.

References

1. G. Booch. *Object-Oriented Design with Applications.* The Benjamin/Commings Publishing Company, Inc., 1994.
2. C.Fernström. PROCESS WEAVER: Adding process support to UNIX. In *The Second International Conference on the Software Process*, pages 12–26, 1993.
3. H. M. Deitel. *An Introduction to Operating Systems.* Addison-Wesley, Reading, Massachusetts, 1984.
4. G.Canals, N.Boudjlida, J.-C.Derniame, C.Godart, and J.Lonchamp. ALF: A framework for building process-centered software engineering environments. In A. Finkelstein, J. Kramer, and B. Nuseibeh, editors, *Software Process Modelling and Technology*, pages 153–185. Research Studies Press, Ltd., Taunton, Somerset, England, 1994.
5. G.Junkermann, B.Peuschel, W.Schäfer, and S.Wolf. MERLIN: Supporting cooperation in software development through a knowledge-based environment. In A. Finkelstein, J. Kramer, and B. Nuseibeh, editors, *Software Process Modelling and Technology*, pages 103–129. Research Studies Press, Ltd., Taunton, Somerset, England, 1994.
6. M. Gruninger and M. S. Fox. An Activity Ontology for Enterprise Modelling. Submitted to AAAI-94, Dept. of Industrial Engineering, University of Toronto, 1994.
7. D. Harel, H. Lachover, A. Naamad, A. Pnueli, 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 Engineering*, 16(4):403–414, April 1990.
8. J.Estublier, S.Dami, and A.Amiour. APEL: A graphical yet executable formalism for process modelling. In *Automated Software Engineering*, March 1997.
9. B. S. Lerner, L. J. Osterweil, J. Stanley M. Sutton, and A. Wise. Programming process coordination in Little-JIL. In V. Gruhn, editor, *Proceedings of the 6th European Workshop on Software Process Technology (EWSPT '98)*, number 1487

in Lecture Notes in Computer Science, pages 127–131, Weybridge, UK, September 1998. Springer-Verlag.
10. K. L. Myers and D. E. Wilkins. The Act Formalism. Working document: Version 2.2, SRI International, Artificial Intelligence Center, September 25 1997. http://www.ai.sri.com/ act/act-spec.ps.
11. H. Rombach and M.Verlage. How to assess a software process modeling formalism from a project member's point of view. In *The Second International Conference on the Software Process*, pages 147–159, 1993.
12. S. F. Smith and M. A. Becker. An Ontology for Constructing Scheduling Systems. In *Working Notes from 1997 AAAI Spring Symposium on Ontological Engineering*, Stanford, CA, March 1997.
13. X. Song and L. J. Osterweil. Engineering Software Design Processes to Guide Process Execution. *IEEE Transactions on Software Engineering*, 24(9):759–775, 1998.
14. R. Vincent, B. Horling, T. Wagner, and V. Lesser. Survivability Simulator for Multi-Agent Adaptive Coordination. *Proceedings of International Conference on Web-Based Modeling and Simulation*, 30(1):114–119, 1998.
15. J. Waldo. *Jini Architecture Overview*. Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303, 1998. http://www.javasoft.com/products/jini/index.html.
16. D. E. Wilkins. *Using the SIPE-2 Planning System: A Manual for Version 4-17*. SRI International Artificial Intelligence Center, Menlo Park, CA, October 1997.
17. A. Wise. Little-JIL 1.0 Language Report. Technical report 98-24, Department of Computer Science, University of Massachusetts at Amherst, 1998.