A Reusable Resource Manager to Support Activity Coordination

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

University of Massachusetts, Amherst MA 01003, USA

Abstract. System behaviors can be expected to vary widely depending upon the availability, or shortage, of resources that they require. Thus, the precise specification of resources required by, and resources available to, a system is an important basis for being able to reason about, and optimize, system behavior. Previous resource models for such disciplines as management and workflow have lacked the rigor to support powerful reasoning and optimization. Some resource models for operating systems have been quite rigorous, but have generally been overly narrow in scope. This paper argues that it is important to be able to optimize and reason about the broad and complex resource requirements of modern distributed multi agent systems. This entails precisely modeling a wide range of entity types, including humans, tools, computation platforms, and data. The paper presents a metamodeling approach that can be used to create precise models of a wide range of resource types, and provides examples of the use of this metamodel. The 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 supporting the execution of processes defined as hierarchies of steps, each of which incorporates a specification of resource requirements.

Keywords

Resources, resource specification, process execution, process centered environments

1 Introduction

Software process technology, workflow management, multi-agent scheduling, and computer supported cooperative work all address issues concerning the coordination of the activities of multiple agents to accomplish a complex task. One key element in all of these areas is the data that is being manipulated, shared, and communicated among the agents. In software process technology and workflow management, data communication is typically represented through data flow edges. In multi-agent scheduling, knowing which agents are producing data and which agents are consuming data is key for developing a good schedule. In CSCW, various locking mechanisms are imposed upon the shared data to allow

the participants to interact in a well-defined manner. We belive that by modeling resources other than data these technologies can benefit with improved process and workflow descriptions, improved scheduling, and improved coordination. In this paper, we present a resource management system that is separate from these technologies and is reusable by these technologies.

In our model, a resource is anything that is required to complete an activity and for which there may be contention. Examples of resources include the people who perform the activities, physical objects required for the activity, licensed software, documentation and other shared data that cannot be used concurrently by multiple users. An agent is a particular type of resource, namely an active resource that can carry out an activity. An agent may be human, software, or even a machine such as a robot.

Modeling resources precisely can benefit activity coordination in several ways. It can enable additional analyses, such as considering the impact of adding resources to or removing resources from an activity. The resources might impact the cost of the activity, its duration or its quality. Modeling resources precisely can improve the accuracy of schedules produced by a scheduler, can allow a scheduler to select schedules that avoid resource contention and that allocate the most appropriate resources for each task to achieve the overall goals of the complex task instead of greedily allocating the best resources to activities as the demands arise.

Operating systems research has long ago demonstrated the pivotal importance of 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 particular design tool, then the various members of a design team are able to work in parallel only if there are multiple copies of that tool available. Similarly, if the design activity has been decomposed into separate parallel subtasks, the design process may go faster, but only if more than one qualified designer is available to work on the project.

Correspondingly, 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 parallelized. Thus, for example, if only one design tool is available for use by two designers, delay may be avoided or reduced by scheduling one designer to perform some other task (for example, requirements revision) while the other designer has use of the one copy of the design tool.

Of course operating systems research has also long ago demonstrated that it is all too easy to create such problems as deadlocks and livelocks if one is not careful in the scheduling of the assignment of resources. These problems 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 is awaiting the results

of a prototype activity in order to complete requirements specification, while a prototyper is awaiting the completion of the requirements specification in order to complete the prototype.

In our work, we are interested in creating process specifications that are sufficiently precise and rigorous that they effectively support reasoning of the sorts just indicated. We would like to be able to identify how resources limit parallelization so that we can schedule them appropriately. We would like to be able to identify where resource usage could lead to deadlocks and starvation /it (Note that this also said race conditions, but I am not sure what the relationship is between resources and race conditions. – Barb), so that we can assure that they do not occur. We would like to be able to infer when additional resources could potentially speed up execution of the overall activity, and when an activity has an excess of resources, some of which might potentially be reassigned. We would like to be able to manage contention for resources that are in high demand, and those to which access should be managed by disciplines such as transaction mechanisms. *(This last bit about transactions has me confused. Transactions only are meaningful for data resources. So isn't dataflow information sufficient for this type of analysis? What is special about resources here? – Barb)*

Since we believe that the need for resource management is widespread we have designed a reusable resource manager that can support a wide variety of resource models. The resource manager supports defining resources and their relationships to each other, querying the resource manager for resources matching a particular need, locking resources for use, and reserving resources for future use. To maintain its reusability, the resource manager is not tightly integrated into any activity coordination mechanism. Therefore, declaraing the resource needs of an activity is outside the resource manager as is prescribing the specific protocols about when resources should be reserved, acquired, and released. We have successfully integrated the resource manager into Little-JIL [**?**], a process definition language, and MAS [**?**], a multi-agent scheduling system. In this paper, we describe the how to define and use a resource model.

2 Defining the resource model

A resource model serves two purposes. First, it is a mechanism to describe and organize the resources available within an organization. It is essential that a resource modeling notation be expressive enough to capture the characteristics of resources and relationships among resources that accurately reflect the actual resource entities being modeled. Second, a resource model is used dynamically to control concurrent access to resources, to schedule resources, to analyze the impact of the resource environment on an organization's ability to carry out an activity. Thus, the dynamic modeling capability of a resource management mechanism is also vitally important. In this section, we present the descriptive features of the resource modeling mechanism, while in the next we present the dynamic characteristics of our mechanism.

2.1 Resource Entities

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. A resource model is described as a collection of resource classes, resource instances, and relationships between them. A **resource instance** represents a unique entity from the physical environment, such as a specific person, printer, or document. A **resource class** represents a set of resources (other classes and/or instances) that have some common properties. Resource classes are further subdivided into unschedulable resource classes and schedulable resource classes. A **schedulable resource class** is one in which its instances are similar enough that they may be interchangeable for some activities. For example, `Printer` would probably be a schedulable resource class. An **unschedulable resource class** is more abstract and is intended more as an organizational convenience when defining the resource model. For example, `Person` might be a sensible resource class, but is too general to be scheduled for a specific task.

Each resource is described using a set of typed attribute-value pairs. There are three pre-defined attributes required of all resources: a name, a textual description, and a set of criteria assertions. Criteria assertions are membership tests used to ensure that the resources are placed in the model consistent with semantic rules defined by the modeler. Criteria assertions are described further in Section **??**.

Schedulable resource classes and resource instances also have a capacity attribute associated with them. Capacity is used to indicate the maximum rate at which a resource can be used. The units by which capacity is measured are specific to the type of resource. For example, a printer's capacity would be measured in pages per minute, while a human's capacity would be measured in hours per week. Capacity is used to support resource sharing and is discussed further in Section **??**.

In addition, the resource modeler can define user-defined attributes that are unique to the specific type of resource. 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 output, but that attribute would not be required of resources that are not printers. These descriptive attributes have static values, that is, the values do not change as a result of use of the resources.

2.2 Relationships between Resources

The resource modeling mechanism supports the definition of three types of relationships between resources: IS-A relationships, Requires relationships, and Whole-Part relationships. IS-A relationships provide a hierarchical abstraction mechanism. Requires relationships express functional dependencies between the use of resources. Whole-part relationships allow resources to be manipulated at either coarse or fine granularities and to allow refinement of resource usage over time.

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 the predefined resource class named `Resource`. A resource instance may belong to multiple classes, thus the entire resource 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 (that is, not in a common ancestor), the instance contains both attributes, qualified by the class name. For example, suppose Amelia is an instance of both the `Programmer` class and the `Translator` class. Suppose each of these classes has a `languages` attribute. Amelia might have Java and C++ as the value for the `Programmer.languages` attribute and English and Japanese as values for the `Translator.languages` class. An example of an IS-A hierarchy is presented in Fig. **??**.

Fig. 1. IS-A hierarchy of an example environment

Requires Functional dependencies between resources are captured with a **Requires** relation. A Requires relation denotes that one resource requires another resource in order to perform any work. For example, a piece of software requires

a computer with a certain configuration or a delivery person requires a vehicle. The fundamental property captured by the Requires relation is that any conceivable use of one resource requires another resource. By capturing these relations in the resource model, the dependency is permanently captured and does not need to be repeated whenever the first resource is used. Arities may be associated with the Requires relation to indicate how many instances of a resource are required.

It is also possible to associate a capacity attribute with a Requires relation. This is useful in situations in which a resource does not require exclusive use of a second resource. For example, while a software package might require use of a specific computer, it generally does not require exclusive use of the computer. When the computer is used for that software package, the capacity available for other activities is reduced by the amount specified in the Requires relation.

The Requires relation can exist between resource classes, resource instances, or a combination of these. A Requires relation between resource instances establishes a tight, static binding between the resources. For example, a particular software package might only be loaded on a specific computer. To use that software package therefore requires use of that specific computer. A Requires relation between resource classes establishes the dependency, but defers the binding of which specific required resource is assigned. For example, a delivery person requires a car, but it does not matter which delivery person uses which car in general. A specific task to be performed by a delivery person might place additional requirements on the car. For example, the product to be delivered might need to remain cool, so the delivery car would then require air conditioning. Those additional requirements can be described in the context in which the resources are being used.

An example of the Requires relation is presented in Fig. **??**. The example shows that a resource instance under schedulable resource class `Coder` requires one resource instance `Visual C++` which, in turn, requires only one resource instance under schedulable resource class Computer. *Seems we should show partial capacity usage between Visual C++ and Computer. – Barb*

Whole–Part The semantics of the Whole-Part relation is that one resource is logically or physically part of a second resource. The difference between Requires and Whole-Part is subtle. Both the Whole and the Parts are modeled as resources to indicate that the resource can be used as a whole, or parts of it can be used individually. Arities and capacities can also be associated with the Whole-Part relation. An example of a Whole-Part relation is that a team is a Whole whose Parts are the team members. The team can be given an assignment, implying a joint assignment to the team members. The assignment can be refined later, giving parts of the assignment to the individual members. This capability does not exist for resources connected with the Requires relation.

The Whole-Part relation can be between resource classes or between resource instances. A Whole-Part relation between resource classes specifies a typing relation that must hold for all Whole instances. For example, a Team resource class

Fig. 2. Requires relation example

might have a Whole-Part relation with a Member resource class with an arity of 3–5. Such a relation implies that any Team instance must have a Whole-Part relation with 3–5 Member instances. Thus, when a Whole instance is added to the resource model, it is imperative that the corresponding Whole-Part relation be created to identify the specific team's members. This differs from the Requires relation in which the binding between resource instances can be deferred until the resources are used, if desired.

An example of Whole–Part relation overlayed on the IS-A hierarchy is presented in Fig. **??**. The figure shows that resource instance `Team1` is an aggregate of resource instances `Frank` and `Dave`.

2.3 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 resource classes and resource instances that are children of the resource class. The additional assertions introduced at a child resource class may not contradict those of a parent resource class. While this is not checkable in general, such contradictions would make it impossible to populate the resource class with instances. Therefore, the resource classes farther from the root along a certain path of IS-A relations have more membership criteria to satisfy and thus represent more specific classes than those closer to the root.

As an example, one might define a resource class FastPrinter as a child of the more general Printer class. The FastPrinter class might have associated with it an assertion that its capacity is greater than 12 pages per minute. Each resource

Fig. 3. IS-A and Whole-Part relations

class or instance below it must satisfy this criteria. Failure to do so indicates that the model does not satisfy the semantics that were intended.

The resource manager checks the criteria assertions after any changes to the criteria assertions, to the attribute values of a resource, or the addition of any resources connected (transitively) with IS-A links.

2.4 Modifying the Resource Model

We expect a resource model to be a relatively static entity. Many resources represent physical entities, such as people and equipment. These types of resources change relatively infrequently. As such, we expect these resources to be added to a model by a human resource modeler. We provide a GUI tool to facilitate the creation, reorganization, and visualization of these resource models.

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 create, reorganize, and examine the resource models.

3 Using the resource model

Once a resource model has been defined, it can be used to control resource sharing, to plan future resource usage, or to reason about whether activities could be performed in less time by increasing the number of resources available. There are four primary operations on a resource model:

- **Identification** of resources that can satisfy specific requirements.
- **Reservation** of a resource for a specific start time and duration at some point in the future.
- **Acquisition** of a resource locking the resource for use in a specific activity.
- **Release** of a resource so that it can be used by other activities.

In this section, we describe these operations. It is important to keep in mind that the activities for which the resources are used are defined and controlled from outside 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.

3.1 Identifying Needed Resources

The first step in using a resource model is to determine what resources exist and how well they match the needs of the activity to be performed. To do this, a resource user specifies the name of a resource class and a query over the attribute values contained by resources of that class. The resource manager then returns a collection of resource instances that match that resource class and query. To find matching resources, the resource manager finds all instances connected transitively via IS-A links from the named resource class. The query is applied to each instance to see if it satisfies the necessary conditions and, if so, that instance is added to the set to be returned.

The identify method of the `ResourceModel` class is defined as:

```
Enumeration identify (String resourceName, Interrogator queryObject, Vector queryParameters)
```

The resource name identifies a resource class or instance to look for. Once the matching resource is found, the method named `query` inside the `queryObject` object is invoked, passing in the query parameters. This technique of passing in a query object allows for arbitrarily complex Java code in the identification of the resource. All resources that pass the query are collected together and returned in an enumeration object. The `identify` method throws an exception if there is no resource with the given name or if there is no resource that satisfies the query.

3.2 Acquiring Resources for Use

The resource manager controls the use of resources by keeping track of bindings to activities that are using them. The resource manager has only an abstract

notion of the activities that are using them. An activity requests use of a resource by specifying either a specific resource instance by name, or a resource class and query to identify a resource to acquire. If the resource manager can successfully identify a matching resource, the resource is locked for use by the given activity. If more than one resource matches, the resource manager selects one. The acquisition request may also indicate a capacity of the resource that is going to be used if exclusive use of the resource is not required.

If the acquisition request is for a resource that requires another resource, the acquisition only succeeds if all transitively-required resources can also be acquired. Similarly, if the resource is a Whole, the corresponding Parts must also be available for the acquisition to succeed.

Note that in general, the resource manager is not actually able to prevent unauthorized use of the resources as their use is done externally to the resource manager. For example, a person can 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.

The method to acquire resources is defined in the `ResourceModel` class as:

```
ResourceInstance acquire (Activity theActivity, String resourceName,
Interrogator queryObject, Vector queryParameters, Capacity neededCapacity)
```

The `Activity` object keeps track of the resources that it is using. This allows for monitoring of resource usage, releasing resources from an activity, or adding resources to an activity dynamically. The `Activity` object should also refer back to an object representing the activity in the application so that acquire resources can be reasoned about from the perspective of the application.

The `acquire` method fails, throwing an exception, if there is no resource with the given name, there is no resource matching the query, or if all resources matching the query are already reserved or acquired for other activities.

Note that the actual implementation does not have a capacity attribute. The resource model contains a counter representing the number of copies of the resource. Each acquisition decrements the counter. – Barb

3.3 Reserving Resources for Future Use

Acquisition results in immediate locking of a resource for use. Reservation supports the ability to plan future use of a resource. As with acquisition requests, reservation requests must specify the resource instance to reserve or a resource class and query as well as the capacity required. In addition, a reservation request must identify the time in the future that the reservation is for and the duration of time that it is for. If there is exactly one matching resource instance available at that time, that resource instance is reserved. If there is a schedulable resource class for which all its resource instances satisfy the query, the reservation is made at the level of the schedulable resource class. In this way, the selection of a specific resource is 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 and by not binding a specific resource

instance at reservation time, it is more likely that the more specific request can be satisfied.

The resource manager creates schedulable resource classes dynamically to allow late binding of reservation requests even when the matching resource instances do not wholly comprise an existing schedulable resource class. For example, suppose our resource model had a class named `Printer` and all printer instances in the building were direct children of the `Printer` class. A printer reservation request might use a query that stipulated a specific value for the `location` attribute. If there was a single printer at that location, that instance would be reserved. If there were multiple printers at that location and at least one printer at a different location, a schedulable resource class would be created dynamically to represent the co-located printers and a reservation would be placed on the class.

Reservations automatically reserve required resources and part resources in a manner similar to acquisitions.

Before using a reserved resource, an acquisition request should be made identifying the reservation that is to be converted into an acquisition.

There is no reserve method defined in the ResourceModel API. The signature for reserve would presumably be the same as for acquire with the addition of a start time and duration. Also it would return a schedulable resource class, not a resource instance. – Barb

3.4 Releasing Resources

If a resource is acquired or reserved, it can be made available again for others to use by releasing the resource. A reserved resource is automatically released if the duration of its reservation expires and it has never been converted to an acquisition.

There are two methods to release resources. First, the `ResourceModel` class defines:

```
void release (Activity theActivity)
```

This releases all resources that were previously acquired or reserved for that activity. Second, the `ResourceInstance` class defines:

```
void release (Activity theActivity)
```

This releases a single resource from the activity. It is also possible to pass in a capacity to release part of the capacity of a resource, but not all of it.

3.5 Refining Usage of Part Resources

In general, a resource cannot be reserved or acquired if there is insufficient capacity available given its current reservations and acquisitions. An exception to this is in the use of Part resources. The main purpose of the Whole-Part relation is to allow a Whole resource to be reserved/acquired resulting in all its parts being

similarly reserved/acquired, but then to allow the initial reservation/acquisition to be refined to a collection of reservations/acquisitions on the Parts. For example, imagine a project management tool that assigns the design of a software system to a team of designers. This ensures that each member of the team is reserved for that task. As the design progresses, the large design assignment will be refined into more specific tasks, each of those given to a smaller team or an individual.

To support refinement, Parts may be reserved/acquired within the context of an existing reservation/acquisition of the Whole. In these cases the reservation/acquisition are made with respect to the capacity already reserved by the Whole reservation/acquisition and not with respect to the generally available capacity of the Parts. *Rodion, is there anything in the API to support this.*

Note that unlike aggregation found in object-oriented design notations, our Whole-Part relation does not denote that the part is entirely owned by the Whole. In this way, an individual might devote 50% of their time to one team and 50% of their time to another team, for example.

4 Example

I just copied the old section 2 (Motivation) here. It doesn't quite work as is. We have included several resource model examples during the section on defining the resource model. We have basically no examples in the section on using the resource model. I think what we need is an example describing use of the model. It would be even better if we had an example of analysis since we talk about that a lot in the intro. Analysis is really application-oriented, however, so it might not be possible to address that here. Such an example is hinted at in the paragraphs below but is pretty vague about how one would actually use the resource manager to do what we say. – Barb

In order to motivate our work we present an example of how it might be useful in addressing a typical software development problem. To that end, we suggest how the management of resources specified by our modeling formalism can improve the effectiveness of a small software development team. We hypothesize that the team has a variety of human resources, among which are three developers, one of whom is both a qualified coder and designer, one of whom is a qualified coder and tester, and one of whom is qualified to do design work, coding, and testing. Non-human resources include one license for the Rational Rose design support system, and one license for the Visual C++ tool.

Let us further hypothesize that the team is currently in the late 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. Thus, some of the team's activities entail recoding, some requires redesign, and all of it requires retesting. It is not hard to also imagine that the progress of the team is, from time to time, slowed by the lack of availability of resources. In particular there are probably times when it would be desirable to have more qualified designers or coders. More likely, however,

there are probably times at which progress could be expedited if the team had available an additional Rose license and an additional Visual C++ license.

We see a number of ways in which the resource management capability that we have developed could be of substantial benefit to this organization. A reasonable scenario might be that the organization has money to purchase only one additional license, and it would be important for it to have some way to determine the best choice, a Rose license or a Visual C++ 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 for analysis of flow time reduction achievable through one purchase or the other (or both). It might be used to study increases in human resource utilization, or it might be used to suggest ways in which the process itself might be altered to reduce flow time without having to purchase new software licenses at all.

The capability we present here assumes that a process is represented as a set of steps that are connected to each other by control flow and/or dataflow dependencies, and that it supports the careful specification of potential parallel activities. The capability requires that each step has attached to it a specification of the resources that are required in order for the step to be executed. Attaching such resource specifications to the various steps enables a process execution supervisor to assure that steps have what they need before being executed. More important, however, such specifications support determination of how much parallelization can be supported by the available resources, and which infusions of new resources are most likely to be most useful, especially when exceptional cases have arisen. Thus, in our example it may be reasonable to purchase an additional design support tool, either because of the lack of availability of a second designer, or because the set of tasks that are dictated by the process does not require or allow parallel design activities. Precise process and resource specification should help determine such things.

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

5 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.

5.1 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 [**?**], and MVP-L [**?**], 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 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 [**?**], 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 [**?**], Statemate [**?**], and ProcessWeaver [**?**]. In all of these cases, however, the sorts of resources that are modeled are rather limited in scope.

5.2 Related work in operating systems

The problem of scheduling resources has been extensively studied in the field of operating systems (for example, [**?**], 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 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)
- 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 [**?**] or [**?**]). The similarities in purposes of resource modeling between software engineering and operating systems fields appear only in research on admissions control ([**?**], [**?**], [**?**], [**?**]) which is more in the domain of networks research. For instance, 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 [**?**] and [**?**] 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.

5.3 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 [**?**]. **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 [**?**]) and a reactive plan execution system). The ACT formalism [**?**] 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).

5.4 Related work in management

An example of a resource modeling approach in a management system is presented in the Toronto Virtual Enterprise (**TOVE**) project [**?**]. 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.

5.5 Related work in other distributed software systems

The **Jini** distributed software system [**?**], 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.

6 Evaluation and Future Work

begin from overview

The mechanism described here is quite general and would be used in different ways by different types of systems. A simple process execution environment might rely just on acquisition and release, greedily acquiring resources as necessary. A planning system would reserve resources as the plan is being developed and acquire and release them according to the plan created. Static analysis might examine the resource requests being made by a system to determine whether the necessary resources exist and where delays or even deadlock might arise when resources are contended for. The resource model is also intended to be manipulable by non-programmers. We therefore require both the static definition of the resource model and its dynamic reservation and acquisition status to be visualizable and easily manipulable through a GUI tool.

end from overview

In an earlier paper [**?**] we describe some of the experience we have had in applying this resource management system. Most of this experience has come from integrating this system with the Little-JIL [**?**] process programming system, and using that integrated system to program processes for robot coordination, data mining, and negotiation, as well as software engineering processes such as collaborative design and regression testing. This past experience has 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. These experiences have served to reinforce our commitment to further experimentation and evaluation, which we believe will lead to further improvements to this system.

Our experiences have also encouraged us to continue to enhance the features of this system. We are planning to focus more effort on the design and implementation of better languages for the definition of the resource model and the specification of resource requirements. Currently we rely primarily on the use of Java for these specifications, and we seek languages that will be more accessible to non-programmer users.

We would also like to see the creation of a simulation system that would use the process and resource specification systems we have developed to draw inferences about projected resource needs and utilization. A simulation system

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 in the process itself.

We would also like to see the resource management system enhanced with a capability for modeling the skilling-up of humans as a result of their execution of certain process steps. Our current system assumes that agents have fixed skill levels. But clearly humans will learn as they go through a lengthy process. As a consequence their skill levels should go up. This is clearly a desirable modeling feature and should be incorporated in a future version of our system.

As a more distant goal we see the addition of a best match resource querying mechanism. Ideally, it should be able to find a closest match to a resource query based on the user specified criteria. Currently, the query is satisfied by returning all the matching resources. For such a mechanism to operate we also need to devise metrics that would allow to measure "distances" between resources (to be exact, their attribute values) and resource queries.

Finally, we need to gain more experience in the use of the resource manager in support of other systems. To this end, we intend to continue the development of processes that use resources, to evaluate how well the resource management system addresses the needs of the process execution environment and specific processes, and to determine how well the resource management system can facilitate planning and analysis activities.

7 Acknowledgments

This research was partially supported by the Air Force Research Laboratory/IFTD and the Defense Advanced Research Projects Agency under Contract F30602-97-2-0032. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of the Defense Advanced Research Projects Agency, the Air Force Research Laboratory/IFTD, or the U.S. Government.