An Ontology for Constructing Scheduling Systems

Stephen F. Smith and **Marcel A. Becker**
The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue Pittsburgh, PA 15213
sfs,mb81@cs.cmu.edu
(412) 268-8811

Abstract

In this paper, we consider the use of ontologies as a basis for structuring and simplifying the process of constructing domain-specific problem-solving tools. We focus specifically on the task of scheduling. Though there is commonality in scheduling system requirements and design at several levels across application domains, different scheduling environments invariably present different challenges (e.g., different dominating constraints, different objectives, different domain structure, different sources of uncertainty, etc.), and hence we can expect high-performance application systems to require customized solutions. Unfortunately, the time and cost associated with such domain-specific system development at present is typically quite large.

Our work toward overcoming this application construction bottleneck has led to the development of OZONE, a toolkit for configuring constraint-based scheduling systems. A central component of OZONE is its scheduling ontology, which defines a reusable and extensible base of concepts for describing and representing scheduling problems, domains and constraints. The OZONE ontology provides a framework for analyzing the information requirements of a given target domain, and a structural foundation for constructing an appropriate domain model. Through direct association of software component capabilities with concepts in the ontology, the ontology promotes rapid configuration of executable systems and allows concentration of modeling effort on those idiosyncratic aspects of the target domain. The OZONE ontology and toolkit represent a synthesis of extensive prior work in developing constraint-based scheduling models for a range of applications in manufacturing, space and transportation logistics.

We first motivate the use of ontologies as model building tools, establishing linkages to recent concepts in software engineering and proposing an extended view of ontologies that includes capability descriptions. We then describe our perspective on the structure of planning and scheduling domain models and summarize major components of current OZONE scheduling ontology.

Ontologies and Model Building

In recent years, the field of software engineering has placed increasing emphasis on software reusability as a key to reducing the time and cost of application system construction and maintenance. Techniques for development and (re)use of software components have received wide attention and use (Biggerstaff & Perlis 1989; Krueger 1992), and tools that support system development from reusable building blocks are maturing (Cotter 1996; Smith 1990; Batory & O'Malley 1992). Despite this activity, however, the systematic development of applications from components remains an open issue. One obstacle stems from the lack of communication and coordination between component developers (who must *design for reuse*) and component users (who *design with reuse*) (Becker & Díaz-Herrera 1994); overly complex components are difficult to reuse while overly simple components do not provide sufficient building blocks. Two current areas of software engineering research aimed specifically at promoting reusability are (1) domain analysis (Hess *et al.* 1990; Arango & Prieto-Díaz 1991) and (2) software architectures (Garlan & Shaw 1994; Clements 1996). Methodologies for domain analysis center around formulation of a *domain model*, which is intended to precisely delineate the scope of an application domain, the objects in this domain, desired system functionalities and features, and the dimensions along which these functionalities vary. Research in software architecture has focused on categorizing reusable architectural styles (Garlan, Allen, & Ockerbloom 1994), on architectural description languages (Shaw & Garlan 1994), and on architectural patterns (Gamma *et al.* 1994) that support composition of components. What is missing are mechanisms to support the transition from (abstract) domain models to specific architectural designs and system implementations.

Within the artificial intelligence community, issues of knowledge acquisition and knowledge sharing have raised similar software reuse challenges, and have pushed research in related directions. Knowledge engineering, for example, has evolved from a process concerned principally with knowledge extraction, where each application is considered uniquely, to a model construction process, where applications are categorized according to type of task and applicable methods (Wielinga *et al.* 1992; Steels 1990). Similarly, trends toward definition of generic tasks and task-specific problem solving architectures (e.g., (Chandrasehakaran 1986)) are motivated by many of the same reasons that underlie software architecture research. One important area of recent research in knowledge-based systems has been the development and use of *ontologies* (Gruber 1993; Uschold 1996; Swartout *et al.* 1996). This work has concentrated primarily on issues of reusable or sharable knowledge bases, focusing on formalizing particular bodies of knowledge, on languages for encoding ontologies, and on methodologies for ontology construction. The broader relevance of ontologies to design and specification of task-specific problem solvers has also been recognized (Wielinga *et al.* 1992; Wielinga & Schreiber 1993), but has received less attention.

The work reported in this paper takes a similar, extended view of the role of ontologies. We advocate the use of ontologies as a means of bridging the gap between domain analysis and application system construction. Our basic approach is to consider an ontology as a framework for specifying models in a particular problem domain, i.e., a meta-model that provides a vocabulary for formulating application models in this problem domain, as well as a set of constraints on what can be expressed. The scope of the ontology is restricted to a particular problem domain, which permits much stronger assumptions to be made with regard to system architecture and sub-structure. On this basis, concepts in the ontology can be explicitly linked to software component capability descriptions, enabling the ontology to serve both as an mechanism for indexing and retrieving relevant software components and as a specification of overall configuration requirements. More generally, the association of component capabilities with concept definitions in the ontology promotes direct configuration of executable systems from specification of an abstract domain model.

This approach to application system construction underlies the design of OZONE, an object-oriented toolkit for configuring constraint-based scheduling systems (Smith, Lassila, & Becker 1996). In the sections below, we describe the ontology that OZONE provides for formulating scheduling domain models.

The Structure of Domain Models in OZONE

As discussed above, the OZONE scheduling ontology can be characterized as a meta-model of the domain of scheduling. It provides a language for describing those aspects of the scheduling domain that are relevant to construction of an application system, and a set of constraints on how concepts in the language fit together to form consistent domain models. Consistency, in this context, relates to the information and knowledge required to insure executability of the model. Generally speaking, the ontology serves to map user-interpretable descriptions of an application domain to application system functionality.

This linkage is established within the OZONE ontology through the inclusion of *capabilities* as an integral part of concept definition. Capabilities provide an operational semantics to the concepts defined in the the ontology, in a form that reflects a specific bias with respect to application system design. In particular, the ontology presumes an underlying constraint-based solution framework and scheduling system architecture (Smith 1994; Smith, Lassila, & Becker 1996); this commitment follows directly from the strong match of constraint-based techniques to the decision-support requirements of practical scheduling environments. Capabilities, then, encapsulate reusable components for configuring and customizing constraint-based solution methods. For example, the concept of a "resource" contributes capabilities for querying and managing its available capacity over time, and different resource types (e.g., reusable, consumable) provide specific "implementations". Given a solution method that incorporates these capabilities, the ontology provides a direct basis for its customization to match the resources in any target domain.

In the remainder of this section, we summarize the basic components of the OZONE scheduling ontology. By convention, we use capitalization to distinguish specific concepts that are included. We start with an overview of the principal concepts involved and their inter-relationships, and then consider each individually in more detail.

Basic components of scheduling models and their relationships

Like several contemporary process modeling and ontology development efforts (Uschold *et al.* 1996; Gruninger & Fox 1994; Le Pape 1994; Lee, Yost, & Group 1994; Tate 1996; Smith 1989) the OZONE scheduling ontology adopts an activity-centered modeling viewpoint. Scheduling is defined as a process of feasibly synchronizing the use of RESOURCES by ACTIVITIES to satisfy DEMANDS over time, and application problems are described in terms of this abstract domain model. Figure 1 illustrates the base concepts involved and their structural relationships. A DEMAND is an input request for one or more PRODUCTS, which designate the GOODS or SERVICES required. Satisfaction of DEMANDS centers around the execution of ACTIVITIES. An ACTIVITY is a process that uses RESOURCES to produce goods or provide services. The use of RESOURCES and the execution of

ACTIVITIES is restricted by a set of CONSTRAINTS.

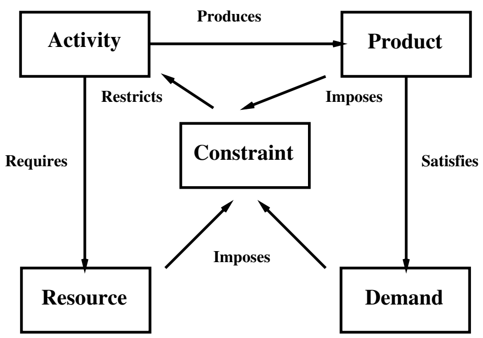

Figure 1: Abstract Domain Model

These five base concepts of the ontology - **DEMAND, ACTIVITY, RESOURCE, PRODUCT,** and **CONSTRAINT** - together with the inter-relationships depicted in Figure 1, define an abstract model of a scheduling domain, and a framework for analyzing and describing particular application environments. Associated with each concept definition are terminologies for describing basic properties and capabilities. Properties define attributes or parameters of relevance to specifying an executable scheduling model. The abstract model and its properties are extensible through concept specializations to define more specific models for various subdomains. Figure 2 indicates model specializations for two such subdomains: manufacturing production scheduling and transportation scheduling. Capabilities designated in the abstract model, alternatively, establish protocols for operationalizing concept definitions in terms of the component functionality required to compose overall solution methods. Specializations of concepts in the abstract model, then, provide a library of implementations that reflect important ontological distinctions. In this respect, the abstract model underlying the OZONE ontology can be viewed as a template for specifying executable domain models.

Associated Capabilities

The capabilities defined in the abstract domain model relate generally to aspects of solution and constraint management, and, as indicated earlier, are rooted in an underlying constraint-based problem solving model. In OZONE, plans and schedules are represented as networks of ACTIVITIES, with an ACTIVITY containing various decision variables (e.g., start time, end time, assigned resources). To construct a schedule that satisfies a given input DEMAND, it is necessary to first instantiate a set of ACTIVITIES that will produce (provide) the designated PRODUCT. This instantiation process is accomplished by *Instantiate-Product-Plan*, a joint capability of DEMAND and PRODUCT

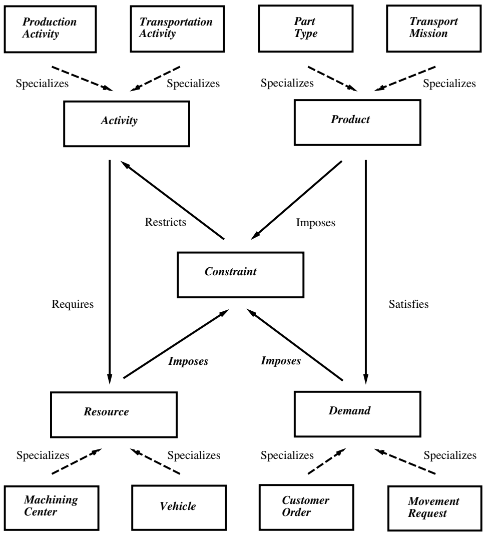

Figure 2: Layered models of scheduling subdomains

that integrates prototype plan information defined by the PRODUCT with the specific parameters of the triggering DEMAND. One consequence of *Instantiate-Product-Plan*, for example, is the imposition of constraints on the start and end times of instantiated activities following from the READY and DUE DATES specified in the DEMAND.

To schedule an ACTIVITY, it is necessary to choose specific RESOURCES, which involves determining intervals where resources have capacity available to support execution of the ACTIVITY, and subsequently allocating capacity of chosen RESOURCES to ensure that they will not be used by other ACTIVITIES. The semantics of allocating (and de-allocating) resource capacity varies according to the type of RESOURCE involved, and hence a RESOURCE provides primitive *Allocate-Capacity* and *Deallocate-Capacity* capabilities. To this end, a RESOURCE maintains a representation of its available capacity over time. A third RESOURCE capability, *Find-Available-Time*, uses this representation to provide a mechanism for identifying intervals of time where the RESOURCE currently has available capacity.

A *Find-Schedulable-Time* capability is associated with an ACTIVITY, which intersects the availability intervals found by a given set of required RESOURCES. This capability, and a companion *Find-Alternative-Resources* capability, provide generic primitives for elaborating a space of alternative decisions. Another pair of ACTIVITY capabilities, *Reserve-Resources* and *Free-Resources*, provide complementary primitives for committing to and retracting specific scheduling decisions. Both *Reserve-Resources* and *Free-Resources* rely, in turn, on the *Propagate-Constraints* capabilities associated with temporal and value CONSTRAINTS to incrementally update the possible values for various decision variables.

The OZONE Scheduling Ontology

In the following subsections, we consider the five basic components of the OZONE scheduling ontology in more detail. In doing so, we assume the existence of basic temporal concepts such as TIME-INTERVALS and TIME-POINTS (c.f. (Allen 1984)).

DEMANDS

Concept Definition. A **DEMAND** is a request for goods and services, or more generically PRODUCTS, that the system being modeled can provide. DEMANDS specify the input goals that drive the system, along with any CONSTRAINTS that must be taken into account when achieving them. The set of outstanding DEMANDS at any point determine the current scheduling problem to be solved.

Properties. A DEMAND has several defining properties:

- **PRODUCT** - The PRODUCT is the object of the DEMAND. It specifies the type of good or service that is requested.
- **RELEASE-DATE** - The earliest time an ACTIVITY for achieving the DEMAND can start.
- **DUE-DATE** - The latest time an ACTIVITY for achieving the DEMAND should end.
- **TEMPORAL-RELATIONS** - These are synchronization constraints with respect to achievement of other system DEMANDS
- **PRIORITY** - The relative importance of the DEMAND, providing a basis for establishing a partial ordering over the entire set of demands.
- **ACTIVITIES** - The set of activities that fulfill the DEMAND. As indicated earlier, these are determined by *Instantiate-Product-Plan*, a joint capability of DEMAND and PRODUCT concept definitions.

For most types of DEMANDS, there will be additional parameters which further specify the requested PRODUCT. DEMAND parameters will vary for different types of goods or services, but typical parameters include:

- **QUANTITY** - A parameter relating to the size of the DEMAND (e.g., the number of goods requested, the amount of material to be processed)
- **MATERIAL** - A parameter relating to the type of material that must be processed
- **ORIGIN, DESTINATION** - If the DEMAND is a request for material to be moved, then ORIGIN and DESTINATION locations are necessary parameters

PRODUCTS

Concept Definition. A **PRODUCT** is a good or service provided by some system of interest. A PART-TYPE is a typical PRODUCT of a manufacturing system; a transportation system alternatively provides TRANSPORT-SERVICES. A PRODUCT is realized through execution of some set of activities. A DEMAND for a PRODUCT is considered satisfied when all of these activities have completed.

Properties. From the standpoint of managing system activities in response to external DEMANDS, properties of interest in defining a PRODUCT relate to the mapping from DEMANDS to ACTIVITIES. Specifically, a PRODUCT definition includes the following:

- **ACTIVITIES** - the set of processing steps required to produce or provide the PRODUCT (i.e., a plan for realizing this PRODUCT)
- **RESOURCES** - the set of resources that can be utilized to execute various ACTIVITIES of the PRODUCT plan.

A PRODUCT specification, together with the constraints and parameters of a requesting DEMAND, enables instantiation of a set of ACTIVITIES for fulfilling the DEMAND (*Instantiate-Product-Plan*). From a scheduling perspective, these ACTIVITIES contain the decision variables (start times, end times, assigned resources) of the problem to be solved; and the instantiation process restricts the domains of these decision variables according to the constraints specified in the DEMAND.

RESOURCES

Concept Definition. Central to the definition of our scheduling ontology is the concept of a RESOURCE. A **RESOURCE** is an entity that supports or enables the execution of ACTIVITIES. RESOURCES are generally in finite supply and their availability constrains when and how ACTIVITIES execute. Making efficient use of RESOURCES in support of multiple, competing ACTIVITIES is the crux of the scheduling problem, and, from the standpoint of constructing scheduling models, the distinguishing characteristics of RESOURCES relate to constraints on their availability.

The availability of a RESOURCE can be defined generally in terms of some dynamically changing aspect of state. Most typically, a RESOURCE is modeled as providing some amount of CAPACITY, a numeric quantity which varies over time as a function of allocating the RESOURCE to various ACTIVITIES and its associated allocation semantics. This is the approach taken in (Fadel, M.S. Fox, & Gruninger 1994; Uschold *et al.* 1996). However, there are also RESOURCES whose availability is more a function of qualitative state: ACTIVITIES require the RESOURCE to be in a particular state or subset of possible states (e.g., to be *idle* as opposed to *busy*) rather than requiring that the RESOURCE have a sufficient amount of CAPACITY. Hence we distinguish two broad classes of resources from the standpoint of availability:

- **CAPACITATED-RESOURCES** - RESOURCES whose availability is characterized in terms of the amount of CAPACITY that is available. In this case, concept specializations provide capabilities for maintaining a representation of available capacity over time (*Increase-Capacity, Decrease-Capacity*), for allocating and deallocating capacity to activities (*Allocate-Capacity, DeAllocate-Capacity*), and for finding periods where capacity is available (*Find-Available-Time*).
- **DISCRETE-STATE-RESOURCES** - RESOURCES whose availability is a function of some discrete set of possible state values. Here definitions provide analogous capabilities for querying, updating and protecting state values over time.

In the case of CAPACITATED-RESOURCES, constraints on availability (i.e. usage of capacity) depend on several different properties of the resource. One determining characteristic is whether RESOURCE CAPACITY is used or consumed by an ACTIVITY when it is allocated:

- A **REUSABLE-RESOURCE,** is a RESOURCE whose capacity becomes available for reuse after an ACTIVITY to which it has been allocated finishes. We say that the ACTIVITY uses the RESOURCE(Uschold *et al.* 1996)
- A **CONSUMABLE-RESOURCE,** is one whose CAPACITY, once allocated to an ACTIVITY does not become available again. We say that the ACTIVITY consumes the RESOURCE.

Though we could further distinguish a third class,*RENEWABLE-RESOURCES*, to refer to RESOURCES that have their CAPACITY increased by ACTIVITIES (Fadel, M.S. Fox, & Gruninger 1994), we instead consider production of RESOURCE CAPACITY to be a separable issue. In our model, ACTIVITIES utilize RESOURCES to produce PRODUCTS. In a resource producing ACTIVITY, the RESOURCE CAPACITY generated is the PRODUCT (or output) of the ACTIVITY; it is not assuming the role of a RESOURCE in this context. Moreover, RENEWABILITY is a property that is equally relevant to REUSABLE-RESOURCES as well as CONSUMABLES. Any RESOURCE can be designated as RENEWABLE by additionally defining it to be a PRODUCT.

A second aspect of RESOURCES that impacts usage (or consumption) of CAPACITY by ACTIVITIES is physical structure. In this respect, RESOURCES can be classified as:

- **ATOMIC-RESOURCE** - This is a RESOURCE that is not divisible and can only be configured to support one process at a time. We can distinguish two subtypes:
  - A **UNIT-CAPACITY-RESOURCE** can only be used by one ACTIVITY during any given TIME-INTERVAL. In this case we could equivalently model the RESOURCE as a discrete state variable with two values : *busy* and *idle*.
  - A **BATCH-CAPACITY-RESOURCE** can support multiple ACTIVITIES if there is sufficient capacity, and if they they require the same resource configuration and are temporally synchronized to occur over the same TIME-INTERVAL. BATCHING-COMPATIBILITY constraints specify the commonality in resource configuration that is required of multiple ACTIVITIES for simultaneous use of a BATCH-CAPACITY-RESOURCE. These constraints are defined with respect to different types of ACTIVITIES that the RESOURCE can support. For example, for two TRANSPORT-ACTIVITIES to be supported by the same vehicle at the same time, both have to require transport between the same locations.
- An **AGGREGATE-RESOURCE** represents a pool of resources, which may be composed of smaller AGGREGATE-RESOURCES or ATOMIC-RESOURCES. The CAPACITY of an AGGREGATE-RESOURCE reflects the collective CAPACITY of its constituent SUB-RESOURCES. This CAPACITY can be independently allocated to multiple activities over any given TIME-INTERVAL, subject only to any constraints induced from the structure of the aggregated SUB-RESOURCES. Based on the nature of SUB-RESOURCE structure, we can define several types of AGGREGATE-RESOURCE:
  - **HOMOGENEOUS-RESOURCE-POOL** - An AGGREGATE-RESOURCE composed of n SUB-RESOURCES of the same type. HOMOGENEOUS-RESOURCE-POOLS can be further differentiated as:
    - **SIMPLE-CAPACITY-POOL** - A HOMOGENEOUS-RESOURCE-POOL which is composed of n UNIT-CAPACITY-RESOURCES and can thus simultaneously support n independent activities. This corresponds to the definition of CAPACITATED-RESOURCE given in (Fadel, M.S. Fox, & Gruninger 1994).

  * **STRUCTURED-CAPACITY-POOL** - A HOMOGENEOUS-RESOURCE-POOL composed of n BATCH-CAPACITY-RESOURCES or n AGGREGATE-RESOURCES of capacity c, having total CAPACITY $n * c$. This type of resource can simultaneously support n independent activities only if the capacity required by any one activity $\leq c$. Any extra capacity over a given TIME-INTERVAL can potentially be used to support additional activities, but only if COMPATIBILITY constraints are satisfied.
- **HETEROGENEOUS-RESOURCE-POOL** - An AGGREGATE-RESOURCE that is composed of RESOURCES of different types and CAPACITIES.

Regardless of the level of detail at which RESOURCE allocation decisions are to be considered in a given domain (e.g., at the level of ATOMIC-RESOURCES or higher), AGGREGATE-RESOURCES capture the hierarchical structure of domain resources in most environments. One consequence is that the unavailability of an AGGREGATE-RESOURCE over a given TIME-INTERVAL always implies the unavailability of its constituent SUB-RESOURCES over the same TIME-INTERVAL.

Properties. The properties of a RESOURCE of primary interest here are those which affect its availability and utilization. Lets first consider availability. In the case of a CAPACITATED-RESOURCE, availability is a function of its **CAPACITY**. CAPACITY is a QUANTITY (or set of QUANTITIES) of some unit measure (e.g., volume, weight, number of activities) that is available for allocation to ACTIVITIES over time. The allocation of a CAPACITATED-RESOURCE to an ACTIVITY implies use or consumption of some amount of CAPACITY, and the number of ACTIVITIES that can be simultaneously supported is limited by the total CAPACITY of the RESOURCE. We can distinguish between different types of CAPACITY models, which impose different CAPACITY CONSTRAINTS:

- A **UNIFORM-CAPACITY** model represents CAPACITY as a scalar QUANTITY. The CAPACITY-CONSTRAINT of a RESOURCE with UNIFORM-CAPACITY requires that, at any point in time, the sum of the CAPACITY used/consumed by all supported ACTIVITIES $\leq$ the CAPACITY of the RESOURCE.
- A **HETEROGENEOUS-CAPACITY** model represents CAPACITY as a vector of two or more UNIFORM-CAPACITIES, reflecting partitioned sub-CAPACITIES. For example, a ship might have separate cargo holds. The CAPACITY-CONSTRAINT of a RESOURCE with HETEROGENEOUS-CAPACITY is the conjunction of the CAPACITY-CONSTRAINTS associated with constituent UNIFORM-CAPACITIES.
- A **MULTI-DIMENSIONAL-CAPACITY** model defines CAPACITY in terms of two or more QUANTITIES, with each contributing a separate CAPACITY-CONSTRAINT that must be satisfied. For example, the capacity of an aircraft might be defined in terms of both maximum weight and volume. In the case of MULTI-DIMENSIONAL-CAPACITY, the CAPACITY-CONSTRAINT requires that for each different unit measure, the sum of the CAPACITY utilized by all supported ACTIVITIES $\leq$ the CAPACITY of the RESOURCE.

In the case of a DISCRETE-STATE-RESOURCE, "availability" corresponds to being in a state that matches the condition of the ACTIVITY that requires the resource. A DISCRETE-STATE-RESOURCE has a set of possible STATE-VALUES. If the RESOURCE is controllable, individual STATE-VALUES can be additionally defined as PRODUCTS; this allows linkage to ACTIVITIES for bringing about their specific values. Allocation of a DISCRETE-STATE-RESOURCE to an ACTIVITY implies commitment to (or protection of) a specific STATE-VALUE over some TIME-INTERVAL, and multiple ACTIVITIES can be simultaneously supported, as long as compatible STATE-VALUES are required.

In many cases, usage of a RESOURCE also depends on other physical properties. Generally, the physical properties of interest will be a function of the domain, but some fairly generic examples include its **SPEED**, which contrains how long ACTIVITIES take to perform, and its **RANGE**, which affects whether it can be used for a particular ACTIVITY or not. Another general physical property of a REUSABLE-RESOURCE is its **SETUP-DURATION**, which specifies how long it takes to configure the RESOURCE for use by a particular ACTIVITY. We can distinguish different types of SETUP-DURATION models:

- A **CONSTANT-SETUP-TIME** model implies that the RESOURCE requires a fixed amount of time to be configured for use by an ACTIVITY, regardless of its prior state.
- A **STATE-DEPENDENT-SETUP-TIME** model implies that the amount of time required to configure the RESOURCE for use by an ACTIVITY is variable and depends on the specific prior configuration of the RESOURCE. A special form of STATE-DEPENDENT-SETUP-TIME is **SEQUENCE-DEPENDENT-SETUP-TIME**, where setup time is assumed to be a function of the last ACTIVITY that was processed using the RESOURCE.

Other **USAGE-RESTRICTIONS** can also limit the availability of RESOURCES:

- **UNAVAILABILITY-INTERVALS** - A TIME-INTERVAL where a RESOURCE cannot be allocated is one simple type of USAGE-RESTRICTION. UNAVAILABILITY-INTERVALS can reflect **RESOURCE-BREAKDOWNS**, periods where **DOWN-SHIFTS** or **RESOURCE-MAINTENANCE** have been planned, or other unmodeled circumstances.
- **CUMULATIVE-USAGE-CONSTRAINTS** - There may also be restrictions on the total amount of RESOURCE use permitted over a given TIME-INTERVAL.

ACTIVITIES

Concept Definition. An **ACTIVITY** represents a process that can be executed over a certain time interval. An ACTIVITY requires RESOURCES to execute and its execution both depends on and affects the current state of these RESOURCES. An ACTIVITY can also have other EFFECTS (e.g., PRODUCTS are produced, other enabling RESOURCE states are established), and it is these EFFECTS that lead ultimately to satisfied DEMANDS. An ACTIVITY may be decomposable into a set of more-detailed SUB-ACTIVITIES, enabling processes to be described at multiple levels of abstraction.

Properties. From the standpoint of the problem solver, an ACTIVITY designates a set of decision variables. The action of *scheduling* an ACTIVITY involves determining values for these variables. The basic decision variables associated with an ACTIVITY are:

- **START-TIME, END-TIME**, which delineate the interval during which the ACTIVITY will occur, and
- **ASSIGNED-RESOURCES**, which indicates the set of RESOURCES allocated to the ACTIVITY

An ACTIVITY has a number of properties that constrain the values that can be assigned to these decision variables:

- **DURATION** - the time required for the ACTIVITY to execute.
- **RESOURCE-REQUIREMENTS** - the set of RESOURCE usage/consumption constraints that must be satisfied for the ACTIVITY to execute.
- **RELATIONS** - the set of TEMPORAL-RELATIONS between this ACTIVITY and others.
- **DEMAND** - the DEMAND that the ACTIVITY was instantiated to satisfy. The DEMAND imposes EARLIEST-START-TIME and LATEST-FINISH-TIME constraints, and associates PRIORITY information.
- **PARAMETERS** - depending on the type of ACTIVITY, there may be one or more PARAMETERS relating to the ACTIVITY's associated DEMAND. For example, if the associated DEMAND is for a QUANTITY of some PRODUCT, then the ACTIVITY might also have a QUANTITY, in this case indicating the portion of the total QUANTITY that it produces.
- **STATUS** - an ACTIVITY may be in one of several states: *UNSCHEDULED, SCHEDULED, IN-PROCESS,* or *COMPLETED.*

Following a constraint-based problem solving orientation, an ACTIVITY provides capabilities for incrementally allocating resources and making variable assignments (*Reserve-Resources*), for retracting previous assignments (*Free-Resources*), and for propagating the consequences of these decisions to related ACTIVITIES (*Propagate-Constraints*). An ACTIVITY thus maintains EARLIEST and LATEST bounds on its START-TIME and END-TIME, as well as a set of currently feasible RESOURCE-ALTERNATIVES. An ACTIVITY also defines primitives for exploring alternative sets of resource assignments (*Find-Alternative-Resources*) and alternative intervals where resources are simultaneously available (*Find-Schedulable-Time*).

CONSTRAINTS

Concept Definition. Generally speaking, a **CONSTRAINT** restricts the set of values that can be assigned to a variable. In the scheduling domain, CONSTRAINTS restrict the assignment of START and END-TIMES and the allocation of RESOURCES to ACTIVITIES. From this perspective, we can identify several basic types:

- **VALUE-COMPATIBILITY-CONSTRAINTS** restrict the values of non-temporal decision variables, and specify conditions under which a value assignment to a given variable is compatible with those of other variables or properties in the model. In the case of basic scheduling models, these CONSTRAINTS relate specifically to RESOURCE assignment decisions and are referred to as **RESOURCE-COMPATIBILITY-CONSTRAINTS**. They designate the conditions under which a given RESOURCE (or type of RESOURCE) can be feasibly used to perform a given ACTIVITY. They may represent physical capabilities and limitations of RESOURCES, or external (e.g., user-imposed) restrictions.

  We can distinguish two varieties of VALUE (or RESOURCE) COMPATIBILITY-CONSTRAINTS:

  - A **STATIC-COMPATIBILITY** specifies a resource usage condition that depends on some other static property of the ACTIVITY that requires the RESOURCE (e.g., parameters of its associated DEMAND, PRODUCT characteristics, other properties of the ACTIVITY itself). For example, an aircraft has a maximum range and may also be capable of carrying only certain types of cargo. Depending on the type of cargo to be moved and the distance of the cargo's destination (both parameters of the input DEMAND), this aircraft may or may not be a compatible (feasible) RESOURCE assignment. From a problem solving perspective, STATIC-COMPATIBILITY-CONSTRAINTS can be applied when an ACTIVITY is first instantiated to prune the space of alternatives in advance of scheduling.

  - A **DYNAMIC-COMPATIBILITY** specifies a compatibility condition or dependency between two RESOURCE assignments (or more generally between two decision variables). It may involve separate RESOURCE assignments for a single ACTIVITY (e.g., the chosen air crew must be qualified to fly the chosen aircraft) or may constrain the RESOURCE assignments of two distinct activities (e.g., the chosen aircraft for both legs of the flight must be the same). One specific type of DYNAMIC-COMPATIBILITY mentioned earlier is a **BATCHING-COMPATIBILITY**, which dictates the circumstances under which two ACTIVITIES can simultaneously use capacity of a BATCH-CAPACITY-RESOURCE. DYNAMIC-COMPATIBILITY-CONSTRAINTS must be re-applied each time a decision is made.

- **TEMPORAL-CONSTRAINTS** restrict the values of temporal decision variables, i.e., ACTIVITY START-TIMES and END-TIMES. There are two basic types:

  - An **ABSOLUTE-TIME-CONSTRAINT** places an absolute lower or upper bound on the value of a TIME-POINT. Examples of ABSOLUTE-TIME-CONSTRAINTS previously mentioned include The RELEASE-DATE-CONSTRAINT and the DUE-DATE-CONSTRAINT imposed by a DEMAND.
  - A **RELATIVE-TIME-CONSTRAINT**, alternatively, restricts the separation between two TIME-POINTS. According to whether or not the constrained TIME-POINTS belong to the same interval or not, we define two subtypes:

    - **INTERVAL-RELATIONS** - An INTERVAL-RELATION synchronizes the occurrence of two TIME-INTERVALS (e.g., two ACTIVITIES). It specifies an ordering with respect to the respective START-TIMES and/or END-TIMES of the two related intervals, and the relation may be quantified by a metric LOWER-BOUND and UPPER-BOUND on the temporal separation between ordered TIME-POINTS. An unquantified INTERVAL-RELATION is interpreted as having LOWER-BOUND, UPPER-BOUND values of $0, \infty$. The set of INTERVAL-RELATIONS includes:

      - **BEFORE** - For two intervals I_1 and I_2, I_1 BEFORE $I_2[lb, ub]$ implies that $ST(I_2) \geq ET(I_1) + lb \bigwedge ST(I_2) \leq ET(I_1) + ub$.
      - **SAME-START** - For two intervals I_1 and I_2, I_1 SAME-START $I_2[lb, ub]$ implies that $ST(I_2) \geq ST(I_1) + lb \bigwedge ST(I_2) \leq ST(I_1) + ub$.
      - **SAME-END** - For two intervals I_1 and I_2, I_1 SAME-END $I_2[lb, ub]$ implies that $ET(I_2) \geq ET(I_1) + lb \bigwedge ET(I_2) \leq ET(I_1) + ub$.
      - **CONTAINS** - For two intervals I_1 and I_2, I_1 CONTAINS $I_2[lb_1, ub_1, lb_2, ub_2]$ implies that $ST(I_2) \geq ST(I_1) + lb_1 \bigwedge ST(I_2) \leq ST(I_1) + ub_1 \bigwedge ET(I_1) \geq ET(I_2) + lb_2 \bigwedge ET(I_1) \leq ET(I_2) + ub_2$.

    - **DURATION-CONSTRAINTS** - A DURATION-CONSTRAINT imposes a LOWER-BOUND or UPPER-BOUND (or both) on the separation between the START and END points of a given TIME-INTERVAL. For interval I_1 and $[lb, ub]$, $lb \leq ET(I_1) - ST(I_1) \leq ub$. An ACTIVITY-DURATION and a RESOURCE'S SETUP-DURATION are two previously mentioned types of DURATION-CONSTRAINTS.

- **RESOURCE-AVAILABILITY-CONSTRAINTS** define third class of physical CONSTRAINT which impacts the assignments of both RESOURCES and START/END-TIMES to ACTIVITIES. The various types of CAPACITY-CONSTRAINTS and USAGE-RESTRICTIONS discussed earlier fall into this category.

- **INSTANTIATION-CONSTRAINTS** represent a final class of CONSTRAINT which restricts the creation of decision variables. In the case of basic scheduling models, decision variables are properties of ACTIVITIES and we refer to this category of constraints more specifically as **ACTIVITY-INSTANTIATION-CONSTRAINTS**. ACTIVITY-INSTANTIATION-CONSTRAINTS include restrictions on how DEMANDS can be mapped to sets of ACTIVITIES. For example, in distributing the cargo that must be moved to satisfy a transport DEMAND across several movement ACTIVITIES (e.g., due to vehicle CAPACITY limitations), there may be physical constraints on how the cargo can be disaggregated.

Properties. A CONSTRAINT may be considered to be **HARD** or **SOFT**. The problem solver is never allowed to violate HARD-CONSTRAINTS. SOFT-CONSTRAINTS, alternatively, are considered to be **RELAXABLE** if need be. For example, DUE-DATE-CONSTRAINTS are treated as RELAXABLE-CONSTRAINTS in many scheduling contexts. The designation of RELAXABLE-CONSTRAINTS is typically accompanied by a specification of **OBJECTIVES** or **PREFERENCES**. When due dates can be relaxed, for example, minimizing tardiness is a common OBJECTIVE. OBJECTIVES and PREFERENCES prioritize the space of possible RELAXATIONS of a CONSTRAINT and provide a basis for measuring solution quality.

Concluding Remarks

The OZONE scheduling ontology is the result of considerable prior experience in building planning and scheduling systems, in application domains ranging from manufacturing production scheduling (Smith 1994) to space mission planning (Muscettola *et al.* 1992) to transportation logistics (Smith & Lassila 1994). The class library design and implementation have followed from retrospective analysis of these scheduling domains and systems (e.g., (Becker & Díaz-Herrera 1994)), together with application of object-oriented analysis and design principles (Smith & Lassila 1994). As evidence of the efficacy of the model-building approach, OZONE was recently applied to develop a quite substantial prototype for reactive, aero-medical evacuation replanning in just two person-months time (Lassila, Becker, & Smith 1996). Though bearing similarity in many respects to various strategic deployment scheduling problems previously addressed with OZONE, the medical evacuation domain also required some fundamentally different capabilities (e.g., integration of itinerary routing and resource allocation decision-making). The use of the ontology and associated constraint management capabilities enabled application development to be quickly localized to those aspects of the domain that required component specialization or extension.

The scheduling ontology presented above is mainly concerned with modeling the entities and constraints of a particular domain. It does not address issues relating to the development of problem solving methods and heuristics that best match domain requirements and characteristics. Though not discussed in this paper, the OZONE toolkit does also provide a companion, agenda-based framework for configuring and integrating problem-solving methods to meet domain-specific requirements. One goal of our current research is to extend our ontological approach to domain modeling to cover this aspect of application construction as well.

Acknowledgements

Ora Lassila has been a principal contributor to the design and development of the OZONE scheduling toolkit, and the work described in this paper has benefited significantly from his association.

This work was sponsored in part by the Department of Defense Advanced Research Projects Agency and Rome Laboratory, Air Force Material Command, USAF, under grant numbers F30602-90-C-0119 and F30602-95-1-0018 and the CMU Robotics Institute. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. 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 Advanced Research Projects Agency and Rome Laboratory or the U.S. Government.

References

Allen, J. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23(2):123–154.

Arango, G., and Prieto-Díaz, R. 1991. Domain analysis concepts and research directions. In Prieto-Díaz, R., and Arango, G., eds., *Domain Analysis and Software Modeling.* Los Alamitos, CA: IEEE Computer Society Press.

Batory, D., and O'Malley, S. 1992. The design and implementation of hierarchical software systems with reusable components. *ACM Transactions on Software Engineering and Methodology.*

Becker, M., and Díaz-Herrera, J. 1994. Creating domain specific libraries: a methodology, design guidelines and an implementation. In *Proceedings of 1994 3rd International Conference on Software Reuse.*

Biggerstaff, T., and Perlis, A. 1989. *Software Reusability.* ACM Press.

Chandrasehakaran, B. 1986. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert* 1(3):23–30.

Clements, P. 1996. Coming attractions in software architecture. Technical Report Technical Report CMU/SEI-96-TR-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Cotter, S. 1996. *Inside Talligent Technology.* Reading, MA.: Addison-Wesley. chapter The Talligent programming model: framework concepts.

Fadel, F.; M.S. Fox, M.; and Gruninger, M. 1994. A generic enterprise resource ontology. In *Proc. of 3rd. IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises.*

Gamma, E.; Helm, R.; Johnsosn, R.; and Vlissides, J. 1994. *Design Patterns: Elements of Reusable Object-Oriented Design.* Addison-Wesley.

Garlan, D., and Shaw, M. 1994. An introduction to software architecture. Technical Report Technical Report CMU-CS-94-166, Carnegie Mellon University, Pittsburgh, PA.

Garlan, D.; Allen, R.; and Ockerbloom, J. 1994. Exploiting style in architectural design environment. In *Proc. of ACM SIGSOFT'94 Symposium on Foundations of Software Engineering.*

Gruber, T. 1993. Towards principles for the design of ontologies used for knowledge sharing. Technical

Report Technical Report KSL-93-04, Stanford University, Palo Alto, CA.

Gruninger, M., and Fox, M. 1994. An activity ontology for enterprise modeling. Technical Report Technical Report, Ind. Eng. Department, University of Toronto.

Hess, J.; Novak, W.; Carrol, P.; Cohen, S.; Holibaugh, R.; Kang, K.; and Peterson, A. 1990. A domain analysis bibliography. Technical Report SEI-90-SR-3, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Krueger, C. 1992. Software reuse. *Computing Surveys* 24(2):131–83.

Lassila, O.; Becker, M.; and Smith, S. 1996. An exploratory prototype for aero-medical evacuation planning. Technical Report Technical Report CMU-RI-TR-96-02, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Le Pape, C. 1994. Implementation of resource constraints in ilog schedule: A library for the development of constraint-based scheduling systems. *Intelligent Systems Eng.* Summer.

Lee, J.; Yost, G.; and Group, P. W. 1994. The pif process interchange format and framework. Technical Report Working Paper No. 180, MIT Center for Coordination Science.

Muscettola, N.; Smith, S.; A., C.; and D'Aloisi, D. 1992. Coordinating space telescope operations within an integrated planning and scheduling framework. *IEEE Control Systems* 12(2).

Shaw, M., and Garlan, D. 1994. Characteristics og high-level languages for software architecture. Technical Report CMU/SEI-94-TR-23, School of Computer Science and Software Engineering Institute , Carnegie Mellon University, Pittsburgh, PA.

Smith, S., and Lassila, O. 1994. Configurable systems for reactive production management. In *Knowledge-Based Reactive Scheduling*. Amsterdam (The Netherlands): IFIP Transactions B-15.

Smith, S.; Lassila, O.; and Becker, M. 1996. Configurable, mixed-initiative systems for planning and scheduling. In Tate, A., ed., *Advanced Planning Technology*. Menlo Park: AAAI Press.

Smith, S. 1989. The opis framework for modeling manufacturing systems. Technical Report Technical Report CMU-RI-TR-89-30, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Smith, D. 1990. Kids: A semiautomatic program development system. *IEEE Transaction of Software Engineering* 16(9):1024–43.

Smith, S. 1994. Opis: A methodology and architecture for reactive scheduling. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*. Morgan Kaufmann Publishers.

Steels, L. 1990. Components of expertise. *AI Magazine* 11(2):29–49.

Swartout, B.; Patil, R.; Knight, K.; and Russ, T. 1996. Toward distributed use of large-scale ontologies. submitted to 1996 Banff Knowledge Acquisition Workshop.

Tate, A. 1996. Towards a plan ontology. *AI*IA Notizie (Journal of the Italian Association for AI)* 9(1).

Uschold, M.; King, M.; Moralee, S.; and Zorgios, Y. 1996. The enterprise ontology v. 1.1. available from http://www.aiai.ed.ac.uk/ enterprise/enterprise/ontology.html, University of Edinburgh, UK.

Uschold, M. 1996. Building ontologies: Towards a unified methodology. Technical Report Technical Report AIAI-TR-197, University of Edinburgh.

Wielinga, B., and Schreiber, A. 1993. Reusable and sharable knowledge bases: A european perspective. In *Proc. Int. Conf. on Building and Sharing Very Large-Scaled Knowledge Bases '93*. Tokyo: Japan Information Processing Dev. Center.

Wielinga, B.; Velde, W.; Schreiber, G.; and Akkernamans, H. 1992. The kads knowledge modelling approach. In *Proc. 2nd Japanese K. A. for Knowledge-Based Systems Workshop.* Hitachi Advanced Research Lab.