The OPIS Framework for Modeling Manufacturing Systems

Stephen F. Smith

CMU-RI-TR-89-30$_2$

Center for Integrated Manufacturing Decision Systems
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

December, 1989

This research has been sponsored in part by IBM Corporation under contract 567016 and the Robotics Institute.

Abstract

In this paper we present a general framework for modeling manufacturing systems. Our interests are in knowledge-based scheduling and simulation of large-scale manufacturing environments, and these interests have significantly influenced the approach to modeling we have taken. The modeling framework is developed using object programming and frame-based representation techniques. It provides an extensible set of modeling primitives that emphasizes (1) specification of the full range of constraints that influence production management and control decisions, (2) the development and use of hierarchical models of manufacturing processes and required resources, and (3) a clear separation of control knowledge relating to resource allocation and manufacturing process management. The framework enables the construction of models that reflect the full complexity of actual manufacturing environments and are interpretable from both scheduling and simulation perspectives.

Table of Contents

List of Figures

1. Introduction

In this paper we present a framework for modeling manufacturing systems. Our interests are in knowledge-based simulation and scheduling of large-scale manufacturing environments, and these interests have significantly influenced the approach to modeling we have taken. Specifically, the modeling framework emphasizes:

- *the developrnent of models that reflect the reality of the manufacturing environment* - The framework defines representational primitives that enable detailed modeling of the full range of constraints that influence scheduling and control decisions.
- *the development and use of hierarchical models of manufacturing processes and required resources.* The simulation and coordination of large-scale manufacturing systems requires the flexibility to selectively and dynamically vary the level of detail at which resource allocation decisions are considered.
- *a clear separation of concerns of resource allocation and concerns related to management of the products being manufactured.* Resource allocation policies are associated with resources; product management policies are associated with control activities.
- *the definition of extensible primitives.* Reliance on object programming and frame-based representation techniques enables the framework to be straightforwardly customized to model the important idiosynchracies of a given manufacturing environment.

This paper is written with two objectives in mind. First, it is intended to present a general knowledge-based modeling perspective and examine basic alternatives vis a vis the definition of various aspects of a knowledge-based modeling framework. Second, it is intended to serve as an implementation-level reference of the specific modeling framework we have defined. Thus, representational issues and the modeling framework are discussed in terms of its implementation language, CRL [4]. The reader is referred to [5] for a description of a simulation kernel that operates on models defined by this framework. The framework generalizes and extends the modeling primitives underlying the OPIS factory scheduling system [8].

1.1. Overview of the Approach

Generally speaking, we advocate a declarative approach to modeling manufacturing systems. A model is specified in terms of five basic types of entities, **operations**, **resources**, **products**, **demands** and **production units**, and the modeling framework defines knowledge structuring primitives relative to each. These primitives provide an extensible framework for representing relevant aspects of the manufacturing system to be modeled, a relational organization that reflects appropriate interdependencies among the manufacturing system entities that are modeled, and a model semantics relative to scheduling and control decision-making.

In more detail, the basic model building components consist of the following:

- **operations** - Operations are descriptions of specific activities that are performed within the manufacturing system. Generally speaking, an operation is a specification of the set of constraints that define a particular activity (e.g. resource requirements, duration constraints, precedence relations relative to other activities, process related control policies, etc.). Operations are organized hierarchically to describe manufacturing processes at different levels of detail. Descriptions of manufacturing processes (or production plans) are instantiated to represent the activities actually taking place (or planned to take place) on the factory floor.
- **resources** - Resources are descriptions of the various resources that are required to perform manufacturing activities. Resource descriptions encode resource allocation constraints and policies at different levels of abstraction, providing the basis for hierarchical definitions of manufacturing processes.

- **products** - Products are descriptions of the materials that are manufactured by the manufacturing system. Products specify the constraints on their manufacture (e.g. the production plan that must be executed, the quantity in which they are produced, material requirements, etc.).
- **demands** - Demands are descriptions of the obligations for product delivery that the manufacturing system has undertaken. Demands specify requests for specific quantities of products within specific time constraints, as well as client-dependent priority information.
- **production units** - Production units are descriptions of the actual entities that are manipulated by the manufacturing system. Each represents a a given set (or quantity) of products to be manufactured. Production units are created in response to demands for specific types of products, and share models (through relational inheritance) with specific demand and product descriptions.

1.2. Organization of the Paper

The remainder of the paper is organized as follows. In Section 2, we make specific assumptions regarding the representation of temporal constraints and define a set of basic temporal primitives for use in subsequent sections. In Section 3, we consider issues related to the representation of manufacturing activities. This is followed in Section 4 by a similar treatment of the issues related to modeling the resources that must be allocated to support manufacturing activities. Finally, in Section 5, we address issues related to modeling products that are produced by the manufacturing system, demands for these products, and the production units that are actually manipulated by the manufacturing system.

2. Basic Temporal Primitives

Fundamental to the modeling, simulation, and scheduling of manufacturing systems is a framework for representing and reasoning about temporally constrained activities. Production demands have requested start times and deadlines, work shifts in different areas of the factory span specified time periods, there are precedence relations between various manufacturing operations, operations are carried out over specific time intervals, etc. An adequate set of temporal primitives must support both relative and absolute specification of temporal constraints. The intent of this section is simply to introduce the set of basic temporal objects that will be needed in subsequent sections, and sketch the approach that is taken to reasoning about temporally constrained activities.

2.1. Representing Absolute Temporal Constraints

Given our need to represent and reason extensively about absolute time constraints, we adopt a point-based framework for modeling time. Of course, we are typically interested in the persistence of various facts over time (e.g. when a given manufacturing operation takes places or is planned to take place), and accordingly define the **time-interval** (see Figure 2-1) as the basic time object. Characteristics of the time points delineating a given time interval are defined by associating each interval with a specific **time-line**. This association is made through the **dated-by** relation (see Figure 2-2), which is defined as an inheritance relation.[1] Thus, we assume that the START-TIME and END-TIME attributes of a given time interval designate specific points on a given time line (as opposed to time points being represented as objects whose values are constrained to a specific set of points on the time line). This assumption has implications with respect to the management of absolute temporal constraints which we will briefly consider below.

---

[1] Conceptually, we could associate all time point manipulation functions with the associated time line. Pragmatically, this will be expensive, and we will instead rely on the time-interval accesser to know what type of interval it is manipulating. We illustrate the concept by specifying two inheritable properties in the definition of **time-line**, a method outputting interval start and end points (*print-point*) and a POINT-GRANULARITY attribute.

{{**time-interval**
IS-A: **time-object**
START-TIME:
END-TIME:
DURATION:
DATED-BY: }}

{{**time-line**
IS-A: **time-object**
DATES:
POINT-GRANULARITY:
print-point: }}

Figure 2-1: The **time-interval** and **time-line** definitions

{{**dated-by**
IS-A: **relation**
DOMAIN: (TYPE is-a **time-interval**)
RANGE: (TYPE is-a **time-line**)
INVERSE: **dates**
INCLUSION: (instance **inclusion-spec**
TYPE: slot
SLOT-RESTRICTION: (NOT dates is-a instance) } }}

{{**dates**
IS-A: **relation**
DOMAIN: (TYPE is-a **time-line**)
RANGE: (SET (TYPE is-a **time-interval**))
INVERSE: **dates** }}

Figure 2-2: The **dated-by/dates** relations

Intervals and time lines are respectively specialized into three associated subtypes (see Figures 2-3, 2-4, 2-5). **Calendar-intervals** provide the primary means for specifying absolute time constraints on activities as well as recording actual activity execution intervals. **Days-of-week-intervals** and **hours-of-day-intervals** will be used later in developing work shift specifications.

{{**calendar-interval**
IS-A: **time-interval**
DATED-BY: **calendar** }}

{{**calendar**
IS-A: **time-line**
DATES: **calendar-interval**
print-point: pt>date }}

Figure 2-3: **calendar-interval** and **calendar** definitions

{{**days-of-week-interval**
IS-A: **time-interval**
DATED-BY: **days-of-week** }}

{{**days-of-week**
IS-A: **time-line**
DATES: **days-of-week-interval**
print-point: pt>day }}

Figure 2-4: **days-of-week-interval** and **days-of-week** definitions

{{**hours-of-day-interval**
IS-A: **time-interval**
DATED-BY: **hours-of-day** }}

{{**hours-of-day**
IS-A: **time-line**
DATES: **hours-of-day-interval**
print-point: pt>hour }}

Figure 2-5: **hours-of-day-interval** and **hours-of-day** definitions

2.2. Representing Relative Temporal Constraints

With regards to expression of relative temporal constraints, we presume a basic set of symbolic relations (e.g. the **successor/predecessor** relations defined in Section 3.2) whose semantics dictate specific consequences with respect to the absolute time constraints associated with the related entities. These relations will be introduced and discussed in subsequent sections in the context of the specific types of entities they are defined to relate.

Let us, however, return momentarily to the issue of "constant" vs "variable" representations of time points and the management of absolute time constraints (an important issue in the context of scheduling). Our assumption of a constant time-point representation implies an underlying constraint propagation mechanism that embeds the temporal semantics of various symbolic relations and uses this knowledge to properly maintain the earliest start and latest end times of the time intervals associated with various entities in the system.[2] The disadvantage here is the special purpose nature of the propagation mechanism (and the complexity of its implementation). The introduction of a variable representation of time points (in the manner of [2]), wherein time points are explicitly represented and related to one another via *distance* constraints, provides a framework for specification of a simple general propagation mechanism. Under this alternative approach, the introduction of a specific symbolic relation (e.g. the introduction of a **successor** relation during instantiation of an actual manufacturing operation to be performed - see Section 3.2) results in the introduction of additional distance constraints between relevant time points (the specific constraints being a function of the semantics of the symbolic relation) which are interpreted by the general propagation machinery. The choice between these two approaches to representing and maintaining absolute time constraints has little impact on the material presented in

[2]This is the approach taken in the current OPIS scheduler [6].

remainder of this document. We only note that a future version of this document and the implementation of the modeling framework it describes will assume the latter approach identified above.[3].

3. Modeling Manufacturing Activities

Of concern in this section is representation of the activities that must be modeled within a manufacturing system. This, of course, includes specification of various manufacturing operations and their organization into production plans. It also includes activities not directly related to production of products, such as machine maintenance and repair. All manufacturing activities are modeled within the framework as **operations**. In the subsections below, we first consider various aspects of their representation independently, and then consider their complete definition.

3.1. Hierarchical Descriptions of Manufacturing Activities

Since we are interested in hierarchical descriptions of manufacturing operations, we first define a set of relations for describing composition/decomposition of manufacturing operations. Here and below we distinguish between hierarchically organized prototype descriptions and the hierarchically organized instances of these prototype descriptions that are created to represent the specific operations that are taking place within the manufacturing system. Hierarchically organized prototype descriptions provide representations of generic manufacturing processes at different levels of precision (e.g. production plans) and provide a basis for *instantiating* the actual operations that must be performed.

For purposes of constructing hierarchical descriptions of manufacturing operations, we distinguish two basic forms of process abstractions:

- conjunctive abstractions - operations defined as conjunctive abstractions decompose into a sequence of operations at the next lower level in the hierarchy.
- disjunctive abstractions - operations defined as disjunctive abstractions decompose into a set of alternative operations at the next lower level in the hierarchy.

Within the representation, manufacturing activities at all levels of description are modeled as **operations**. An operation has a TYPE, whose value distinguishes the operation as either a conjunctive/disjunctive abstraction or an *atomic* activity in the model (see Section 3.6 below). Hierarchical descriptions of arbitrary depth can be constructed using these two basic abstraction building blocks. Since the objective is to provide a basis for reasoning about resource allocation decisions at multiple levels of details (for purposes of either simulation or scheduling), process abstractions are motivated by useful abstractions of the resources that they require. A corresponding framework for modeling resources at different levels of abstraction is presented below in Section 4.

In Figures 3-1 and 3-2, we define basic composition/decomposition relations used to define process abstractions. The **possible-sub-operation-of/possible-sub-operations** relation pair provides primitives for organizing prototype operation descriptions, which are modeled in the representation as specific types (or classes) of operations. The actual operations performed in the factory are represented as instances of prototype operations and are related to other instances of prototype operations via the **sub-operation-of/sub-operations** relational primitives.

---

[3]A knowledgecraft implementation of this generalized approach to maintenance of temporal constraints is described in [1]

{{**possible-sub-operation-of**
IS-A: **relation**
DOMAIN: (TYPE is-a **operation**)
RANGE: (TYPE is-a **operation**)
INVERSE: **possible-sub-operations** }}

{{**possible-sub-operations**
IS-A: **relation**
DOMAIN: (TYPE is-a **operation**)
RANGE: (SET (TYPE is-a **operation**))
INVERSE: **possible-sub-operation-of** }}

Figure 3-1: Relations for hierarchical descriptions of prototype operations

{{**sub-operation-of**
IS-A: **relation**
DOMAIN: (TYPE instance **operation**)
RANGE: (TYPE instance **operation**)
INVERSE: **sub-operations** }}

{{**sub-operations**
IS-A: **relation**
DOMAIN: (TYPE instance **operation**)
RANGE: (SET (TYPE instance **operation**))
INVERSE: **sub-operation-of** }}

Figure 3-2: Relations for hierarchical descriptions of instantiated operations

Values in the range of a specific **possible-sub-operations** relation (or a **sub-operations** relation in the case of instantiated operations) identify the specific set of sub-operations that the relation's domain operation abstracts. The relation, on the other hand, contains no information about the relationships among the identified sub-operations themselves (i.e. there is no assumed ordering relative to the values present in the range of the relation). These constraints are specified at the sub-operation level of abstraction in the hierarchical model (and we consider their representation below in Section 3.2). In the case of conjunctive process abstractions, it is convenient to explicitly designate the entry and exit points of the operation sequence being abstracted. The relations defined in Figures 3-3 and 3-4 are defined to this end, again for both prototype and instantiated operations respectively.[4].

[4]The information supplied by these relations could of course be inferred when needed from consideration of the more detailed sub-operation descriptions identified by a given **possible-sub-operations** (or **sub-operations**) relation. Since the information is static in nature, however, it seems appropriate to "compile" this information into the representation

{{**initial-operation-of**
IS-A: **relation**
DOMAIN: (TYPE is-a **operation**)
RANGE: (TYPE is-a **operation**)
INVERSE: **subplan-entry-operation** }}

{{**subplan-entry-operation**
IS-A: **relation**
DOMAIN: (TYPE is-a **operation**)
RANGE: (SET (TYPE is-a **operation**))
INVERSE: **initial-operation-of** }}

{{**final-operation-of**
IS-A: **relation**
DOMAIN: (TYPE is-a **operation**)
RANGE: (TYPE is-a **operation**)
INVERSE: **subplan-exit-operation** }}

{{**subplan-exit-operation**
IS-A: **relation**
DOMAIN: (TYPE is-a **operation**)
RANGE: (SET (TYPE is-a **operation**))
INVERSE: **final-operation-of** }}

Figure 3-3: Additional Relations for hierarchical prototype descriptions

Hierarchical descriptions of manufacturing processes provide the opportunity to regulate the level of detail at which various resource allocation decisions are reasoned about. In the context of large-scale simulation, for example, it may be desirable to selectively consider more or less detail in different areas in the factory. Similarly, from a scheduling perspective, more or less detail may be desirable, depending perhaps on the unpredictability inherent in the processes being modeled or the time horizon of the scheduling decisions being made. Determination of an appropriate set of process abstraction levels is obviously a function of the particular manufacturing system being modeled. In particular, characteristics and organization of the resources that must be allocated within the manufacturing system will dictate the decision-making levels of interest. We will discuss issues relating to specification of appropriate abstraction levels and regulation of the level of precision of decision-making in Section 4, when we consider the representation of resources.

{{**first-sub-operation-of**
IS-A: **relation**
DOMAIN: (TYPE instance **operation**)
RANGE: (TYPE instance **operation**)
INVERSE: **first-sub-operation** }}

{{**first-sub-operation**
IS-A: **relation**
DOMAIN: (TYPE instance **operation**)
RANGE: (SET (TYPE instance **operation**))
INVERSE: **first-sub-operation-of** }}

{{**last-sub-operation-of**
IS-A: **relation**
DOMAIN: (TYPE instance **operation**)
RANGE: (TYPE instance **operation**)
INVERSE: **last-sub-operation** }}

{{**last-sub-operation**
IS-A: **relation**
DOMAIN: (TYPE instance **operation**)
RANGE: (SET (TYPE instance **operation**))
INVERSE: **last-sub-operation-of** }}

Figure 3-4: Additional Relations for hierarchical descriptions of instantiated operations

3.2. Representing Precedence Relations

Manufacturing operations are also related according to precedence constraints, which dictate the order in which operations defined at a given level of abstraction in the model must be performed. In defining a set of precedence relations for expressing these constraints, we again distinguish between prototype and instantiated operation descriptions (see Figures 3-5 and 3-6).

{{**possible-successors**
IS-A: **relation**
DOMAIN: (TYPE is-a **operation**)
RANGE: (SET (TYPE is-a **operation**))
INVERSE: **possible-predecessors** }}

{{**possible-predecessors**
IS-A: **relation**
DOMAIN: (TYPE is-a **operation**)
RANGE: (SET (TYPE is-a **operation**))
INVERSE: **possible-successors** }}

Figure 3-5: Precedence relations between prototype descriptions

{{**successor**
IS-A: **relation**
DOMAIN: (TYPE instance **operation**)
RANGE: (TYPE instance **operation**)
INVERSE: **predecessor** }}

{{**predecessor**
IS-A: **relation**
DOMAIN: (TYPE instance **operation**)
RANGE: (TYPE instance **operation**)
INVERSE: **successor** }}

Figure 3-6: Precedence relations between instantiated operations

These two relation pairs have considerably different semantic interpretations. In the case of precedence relations among instantiated operations (i.e. the **successor/predecessor** relations) sequences of operations are defined that are known with certainty. In the context of simulation, these relations define the sequence of operations actually carried out to produce specific production units, repair failed machines, etc. In the context of scheduling, they define expected manufacturing processes for purposes of reasoning about allocation of resources.

On the other hand, precedence relations defined within prototypical process descriptions (i.e. expressed via **possible-successors/possible-predecessors** relations) are intended to provide a basis for describing generic manufacturing processes, defining instead sets of possible operation sequences. This raises several additional representational requirements:

- Manufacturing processes are typically conditional in nature, implying the need to represent *state-dependent* precedence relations. We can identify two broad types of state-dependent precedence relations:
  - dynamic - These relations define operation sequences that are contingent on the dynamic aspects of the current production state. For example, the outcome of a specific test/inspection operation will dictate whether the actual successor operation will be the next step in the product's production or a product rework operation.
  - static - These relations define operation sequences that are contingent on determinable aspects of the current production state. For example, many manufacturing processes involve a fixed amount of iteration over specific sub-processes.
- Specification of repair/rework components of production processes introduces further complications. A given production repair or rework subprocess is often sharable (i.e. a POSSIBLE-SUCCESSOR at several distinct points in the overall production process), and can sometimes be self-referential in nature (i.e. the need for repair/rework is recognized in the midst of repair/rework). This implies the need for a representation of precedence relations that distinguishes operation sequences that should be interpreted as "subroutine definitions".
- Finally, conditional precedence relations can exist between different manufacturing processes (e.g. production processes operating on different production units). For example, successful completion of a given manufacturing subprocess on a given production unit (as determined during a final inspection step) may indicate stability of the subprocess and enable initiation of the subprocess with respect to other production units. This implies a need to represent outcome dependent causal relationships between manufacturing processes operating on different production units.

These representational requirements are addressed augmenting each value in the range of an operation's **possible-successors** relation with a meta-description of the relation implied by the value. These meta-descriptions are referred to as **Connection-specs**, and provide a framework for encoding both the causal constraints and shared subprocess assumptions that underlie each specific precedence relation. This is made precise in Figure 3-7.

{{**connection-spec**
IS-A: **conceptual-object**
LINK-TYPE:
range: (OR *jump call return exit*) }}

{{**outcome-dependent-spec**
IS-A: **connection-spec**
CONDITION:
PROBABILITY:
MESSAGE-DESTINATIONS:
range: (SET (TYPE instance **wait-operation**)) }}

{{**computable-condition-spec**
IS-A: **connection-spec**
LINK-TYPE: *jump*
PREDICATE: }}

Figure 3-7: The **connection-spec** definition

The LINK-TYPE attribute included in the basic **connection-spec** definition provides a specification of the precedence relation vis a vis the use of shared process descriptions. Four possible values are allowed, with the following semantics:

- *jump* - Jump implies a coupling of two operations within the same prototype process description. In instantiating prototype operations to represent a particular production unit's production process, movement across *jump* relations occurs in a memoryless fashion (i.e. knowledge of the operation in the domain of *jump* relation plays no role in the interpretation of subsequently encountered precedence relations).

- *exit* - Exit designates the domain operation as the last operation of a prototype process description at a given level of detail in the hierarchical model. In this case, the value of the range of the relation is not an operation but simply the designated marker symbol *end*. Interpretation of an *exit* relation causes either movement to one of the POSSIBLE-SUCCESSORS of the abstract operation representing the just completed process at the next higher level of description, or completion of the overall production process (if at the highest level of description). The CONDITION associated with an *exit* link (see discussion of **outcome-dependent-spec** below) is assumed to match the CONDITION of one of the POSSIBLE-SUCCESSORS at the next higher level of abstraction.

- *call* - Call implies a coupling of the relation's domain operation with the first operation of a shared process description. In moving across a *call* relation, the particular instance of the domain operation of the relation that just terminated is recorded for use in subsequent interpretation of the corresponding *return* relation.

- *return* - Return designates the domain operation as the last operation of a shared process description. As in the case of *exit* links, the value of the range of the relation is the designated marker symbol *end*. Interpretation of a *return* relation causes movement to one of the POSSIBLE-SUCCESSORS of the operation previously marked as the most recent call point.

The CONDITION of a *return* link is assumed to match the CONDITION of one of the POSSIBLE-SUCCESSORS of the operation previously recorded as the call point.

The **connection-spec** definition is specialized in two ways to characterize the two types of state-dependencies identified above. **Outcome-dependent-specs** describe precedence relations whose relevance depends on the specific outcome of the relation's domain operation. In this case, the specified CONDITION designates the operation termination condition under which the relation is appropriate, and the PROBABILITY indicates the likelihood of this outcome. Specified conditions are assumed to be symbols (e.g. *success*, *fail1*, etc.) Probability values are assumed to range from 0 to 1 with the additional constraint that the sum of the probability values associated with each of the POSSIBLE-SUCCESSORS of a given operation equals 1. Thus, the specs associated with a given set of POSSIBLE-SUCCESSORS define the range of possible outcomes of the domain operation, and provide a basis for simulating actual operation outcomes. Figure 3-8 provides an example of the use of **outcome-dependent-specs**.

```
{{op1
    ...
    POSSIBLE-SUCCESSORS:
        op2
            {INSTANCE outcome-dependent-spec
                LINK-TYPE: jump
                CONDITION: success
                PROBABILITY: .95 }
        op1-rework
            {INSTANCE: outcome-dependent-spec
                LINK-TYPE: jump
                CONDITION: fail
                PROBABILITY: .05 }
    ... }}
```

Figure 3-8: Example of the use of **outcome-dependent-specs**

The **outcome-dependent-spec** also defines a set of MESSAGE-DESTINATIONS, which specify causally related activities involving different production units. n the event that the outcome of the relation's domain operation indicates that the relation is to be enforced, triggering messages are sent to all operations designated in MESSAGE-DESTINATIONS. These operations are, by definition, **wait-operations** (see Section 3.6.2), which represent conditional suspensions of production activity.

A second type of connection spec, termed a **computable-condition-spec**, is defined for description of precedence relations that are contingent on computable aspects of the current production state (e.g. how far along in the production process a given production unit is). In this case, the relevance of a specific precedence relation is determined through evaluation of a designated PREDICATE. **Computable-condition-specs** provide a basis for expressing bounded iteration relative to a given manufacturing process and always presume a LINK-TYPE of *jump*. An example of their use in this regard is given in Figure 3-9.

```
{{rl-liftoff
    ...
    POSSIBLE-SUCCESSORS:
        final-inspect
            {INSTANCE computable-condition-spec
                LINK-TYPE: jump
                PREDICATE: last-chip-layer-completep }
        new-chip-layer-prep
            {INSTANCE computable-condition-spec
                LINK-TYPE: jump
                PREDICATE: chip-layers-remainingp }
    ... }}
```

Figure 3-9: Example of the use of **computable-condition-specs**

3.3. Associating resource requirements

Specification of manufacturing activities also requires representation of resource requirements. We partition the set of resource requirements to be associated with specific manufacturing activities into two types:

- primary resource requirement - We assume that for any particular operation, a specific resource can be designated as primary from the standpoint of allocation. The type of resource designated as primary can vary for different operations (e.g. an AGV would be the primary resource in a transporting operation, an expose machine would be the primary resource requirement of a photoexpose operation, etc.), but we assume that if the operation requires use of a stationary resource, which at the lowest level designates either a machine or a particular work station in the factory (see Section 4), then the stationary resource will be the primary resource requirement. Primary resources are distinguished in that they provide a locus for associating control policies.
- secondary resource requirements - Secondary resource requirements designate those supporting resources that are required to perform the operation to which they are associated. Secondary resources are assumed to always be mobile resources such as operators and tools (see Section 4).

These two sets of requirements are defined for specific operations using the **primary-resource** and **secondary-resources** relations respectively.

{{**primary-resource**
IS-A: **relation**
DOMAIN: (TYPE instance **operation**)
RANGE: (TYPE instance **resource**)
INVERSE: **primary-resource-for** }}

{{**primary-resource-for**
IS-A: **relation**
DOMAIN: (TYPE instance **resource**)
RANGE: (SET (TYPE instance **operation**))
INVERSE: **primary-resource** }}

{{**secondary-resources**
IS-A: **relation**
DOMAIN: (TYPE instance **operation**)
RANGE: (SET (TYPE instance **mobile-resource**))
INVERSE: **secondary-resource-for** }}

{{**secondary-resource-for**
IS-A: **relation**
DOMAIN: (TYPE instance **mobile-resource**)
RANGE: (SET (TYPE instance **operation**))
INVERSE: **secondary-resources** }}

Figure 3-10: Relations for describing resource requirements

The range specifications for both the **primary-resource** and **secondary-resources** relations designate instances of particular resources. Taken together, these two relations identify all resources required by a given operation, and each resource identified is assumed to be required for the entire duration of the operation. The relations do not, however, fully specify resource requirements, as the association of a resource requirement with an operation does not necessarily imply that 100% of the resource is required. This is particularly true at higher levels of abstraction, where aggregate resources are identified as resource requirements (see Section 4). Thus, the **primary-resource** and **secondary-resources** relations are more accurately viewed as constraints that state that some percentage of each designated resource's capacity is required to perform the operation. The interpretation of a given resource requirement is made precise through attachment (in the same manner as connection specs) of a **resource-requirement-spec** to the resource designated in the range of the relation.

{{**resource-requirement-spec**
IS-A: **conceptual-object**
REQUIRED-RESOURCE-FUN:
REQUIRED-QUANTITY-FUN: }}

Figure 3-11: The **resource-requirement-spec** definition

The attributes introduced in the **resource-requirement-spec** (Figure 3-11), which are both assumed to be lisp functions that take the related operation as their sole argument, define two ways in which a resource requirement might be further specified relative to the percentage of the resource actually required:

- amount of capacity required - As we will see in Section 4, a numerical CAPACITY value is specified for each resource to define its overall potential for parallel activity. Given this, a numerical specification of the amount of capacity that is required by an operation is typically sufficient to fully define a resource requirement. However, the amount of capacity required depends on the nature of the operation and cannot always be specified as a constant. The amount of capacity required for a baking operation in an oven, for example, depends on the size of the production unit. Accordingly, capacity requirements are represented procedurally within the **resource-requirement-spec** through specification of an appropriate function (REQUIRED-QUANTITY-FUN). We define two subtypes of **resource-requirement-spec** in Figure 3-12 which provide definitions for the two most common situations. In the case of **constant-quantity-spec**, a QUANTITY attribute is introduced and this value is simply returned by the specified REQUIRED-QUANTITY-FUN. In the case of **batch-quantity-spec**, the specified REQUIRED-QUANTITY-FUN is assumed to return the size of the production unit associated with the related operation.

- specific subset of the resource required - In the case of secondary resources, it is sometimes impractical to explicitly identify the resource required in the range of the relation. Consider, for example, the use of masks in semiconductor manufacturing operations. A different mask is required for each type of wafer that is manufactured (typically 1000s). At the same time, wafer manufacturing processes are basically differentiable only at the product family level (typically 10s). Thus, association of precise mask requirements with operations would require representation of 1000s of highly redundant manufacturing processes (i.e. with the processes for wafers belonging to the same product family varying only in the type of mask required at specific steps in the process). In light of such situations, we also associate a function (REQUIRED-RESOURCE-FUN) with the **resource-requirement-spec** to provide an alternative means for specifying resource requirements that vary only with respect to the contents of the production unit being manipulated by the operation. Figure 3-13 illustrates its use in the case of the mask example. Here the **secondary-resources** relation specifies the need for capacity relative to an overall **mask-pool** (an aggregate resource), and the defined REQUIRED-RESOURCE-FUN determines the specific mask subpool (also an aggregate resource) that represents copies of the particular mask that is required[5]. An additional secondary resource requirement, designating capacity from the pool of expose machine operators, is also included to illustrate a situation where a secondary resource is explicitly specified. In this case, the REQUIRED-RESOURCE-FUN simply returns the resource designated in the relation (implying in this example that we need a qualified operator but don't care which one it is).

{{**constant-quantity-spec**
IS-A: **resource-requirement-spec**
QUANTITY:
REQUIRED-QUANTITY-FUN: get-constant-quantity }}

{{**batch-capacity-spec**
IS-A **resource-requirement-spec**
REQUIRED-QUANTITY-FUN: calc-prod-unit-size }}

Figure 3-12: Subtypes of **resource-requirement-spec**

---

[5]We assume in this example that mask information is associated with product descriptions, which are accessible through the description of the production unit designated by the operation

```
{{expose-operation
    ...
    SECONDARY-RESOURCES:
        mask-pool
            {INSTANCE constant-quantity-spec
                QUANTITY: 1
                REQUIRED-RESOURCE-FUN: get-product-dependent-mask-pool }
        expose-operator-pool
            {INSTANCE constant-quantity-spec
                QUANTITY: 1
                REQUIRED-RESOURCE-FUN: get-designated-resource }
    ... }}
```

Figure 3-13: Procedural specification of secondary resource requirements

This procedural specification of resource requirements raises a broader issue concerning the modeling of manufacturing processes. It is obvious that decisions as to the level of generality at which manufacturing processes should be defined are not always clear cut and there are always tradeoffs to be considered (in this case regarding specification of secondary resource requirements). The most we can expect from any representational framework is a set of primitives that can be straightforwardly customized according to peculiarities of the specific application.

3.4. Specifying operation duration constraints

Another aspect of the definition of manufacturing activities is specification of constraints relating to operation duration. These constraints are, of course, quite often a function of the particular resource that is designated as the operation's primary resource (e.g. a function of the operating speed of the machine). However, since all operation descriptions are constrained to designate a single primary resource and process abstractions are assumed to be based on appropriate abstractions of required resources, association of duration constraints with operations is perfectly natural, serving to further define the manufacturing activity being modeled.

One important concern influencing the specification of a representation of operation duration constraints is the modeling perspective adopted relative to resource *setup* activities. Utilization of a resource in the context of a specific production activity is typically predicated on the resource being in a particular state. A machine must be configured according to the characteristics of the manufacturing operation that requires it. An AGV must be at same location as a production unit before a transport operation involving that production unit can be carried out. In situations where a resource is not in the state required by the operation, then preparatory actions that bring the resource to the required state must necessarily be performed prior to execution of the production operation. We refer generally to such preparatory actions as resource setup activities.

From a modeling standpoint, there are different assumptions we might adopt regarding the issue of setup activities. On the one hand, we might assume an explicit representation of all relevant aspects of the state of resources over time, express the resource requirements of operations directly in terms of required state values, and dynamically construct and schedule the sequence of setup operations necessary to satisfy these constraints. Such an approach to dealing with state-dependent setup constraints is described in [7].

On the other hand, our primary objective in modeling setup activities in the context of manufacturing systems is determination (or estimation) of setup duration. In light of this, the approach outlined above

would seem most advantageous in situations where there is considerable interaction between the setup activities of different resources and this interaction has a significant effect on the total setup duration. In manufacturing environments, this is typically only the case at micro levels (e.g. a shared AGV within a flexible manufacturing cell). The duration of a machine setup activity, for example, is typically not affected by any setup activities being performed relative to other resources. Given this fact, and the fact that we are interested in large-scale simulation and scheduling, we assume it reasonable to operate with a simpler, implicit model of operation-dependent setup activities. More specifically, we assume that resource setup activities can be adequately modeled as adjustments to the durations of operations that require the setup. This is accomplished by defining the state of a resource at any given point in time to be a function of the operation last performed, which in turn provides a basis for defining setup durations.

The use of this implicit model of resource setup activities is not without some loss of generality:

- By modeling setup activities as adjustments to operation durations, it is assumed that resource setup activities must occur immediately prior to the operation requiring the resource, which need not always be the case.
- While it is possible to differentiate between the portion of an operation's duration that is devoted to setup and the portion that represents its actual execution (which is, in fact, done in the representation presented below), it becomes somewhat cumbersome to differentiate at finer levels (e.g. between that portion of the setup duration that requires the presence of the production unit and that portion that can proceed without its presence).

One final comment regarding representational objectives is in order before considering the details of the representation of operation duration constraints. We assume that the framework for specification must provide a basis for use of decision procedures that perform some amount of lookahead from the current state or rely on some amount of scheduling. This implies an ability to specify and derive expected durations as well as actual durations.

Given these assumptions, we define the concept of a **duration-spec** (Figure 3-14). **Duration-specs** are intended to encapsulate all parameters and calculations relevant to determination of an operation's duration. Two generic methods, common to all duration specs, define basic mechanisms for determining durations and a framework for specialization according to domain-specific parameters and calculation methods:

- *get-expected-duration* - This method returns the "expected" total duration of the associated operation (including any required setup time). The actual method is defined quite simply as:

```
(defun calc-expected-duration (op message
                               &optional (prev-ops nil))
  (declare (ignore message)
  (+ (call-method op 'get-expected-setup-duration prev-ops)
     (call-method op 'get-expected-run-duration)))
```

  The definitions of the methods supporting this calculation, *get-expected-setup-duration* and *get-expected-run-duration*, will obviously vary across different types of **duration-specs**. We will consider the nature of this variability in Section 3.6 as the definitions of different types of operations are elaborated.

- *get-actual-duration* - This method returns the "actual" total duration of the associated operation according to a specified PROBABILITY-DISTRIBUTION. It operates by first obtaining the "expected" duration, which provides a basis for defining the absolute parameters of the specified distribution, and then returning a sample drawn over this distribution. A pre-defined set of possible types of probability distributions is assumed[6] and the parameters necessary to specify a distribution of any one of these types appear as specifiable attributes of the

[6]which are exactly those types of distributions that are supported by SIMPAK.

duration-spec (in Figure 3-14 this is alluded to by "STANDARD-DEVIATION SPREAD ..."). Values filling these distribution parameter slots are assumed to be relative to the expected duration.

{{**duration-spec**
IS-A: **conceptual-object**
get-expected-duration: calc-expected-duration
get-expected-setup-duration:
get-expected-run-duration:
get-actual-duration: calc-actual-duration
DURATION-SPEC-OF:
PROBABILITY-DISTRIBUTION:
range: (TYPE instance **distribution-object**)
STANDARD-DEVIATION:
SPREAD:
... }}

Figure 3-14: The **duration-spec** definition

Duration-specs are associated with operations via the **has-duration** relation. It is defined as an inheritance relation to make aspects of the attached **duration-spec** immediately accessible to the operation description. The **has-duration** relation and its inverse are defined in Figure 3-15.

{{**has-duration**
IS-A: **relation**
DOMAIN: (TYPE instance **utilization-operation**)
RANGE: (TYPE instance **duration-spec**)
INVERSE: **duration-spec-of**
INCLUSION: **is-a-inclusion-spec** }}

{{**duration-spec-of**
IS-A: **relation**
DOMAIN: (TYPE instance **duration-spec**)
RANGE: (TYPE instance **utilization-operation**)
INVERSE: **has-duration** }}

Figure 3-15: The **has-duration/duration-spec-of** relations

3.5. Operation Status Information

We associate a STATUS attribute with the definition of **operation** (with possible values of *pending*, *inprocess*, or *completed*) to indicate the current state of a given instantiated operation at any point in time. Similarly, we associate an EXECUTION-INTERVAL with each instantiated operation, to delineate its actual (or planned) start and end times. The range of EXECUTION-INTERVAL is constrained to be an instance of a **production-activity-interval** (Figure 3-16). This specialization of **calendar-interval** introduces two additional attributes: RUN-DURATION and SETUP-DURATION.

{{**production-activity-interval**
IS-A: **calendar-interval**
SETUP-DURATION:
RUN-DURATION: }}

Figure 3-16: The **production-activity-interval** definition

Within the basic representational framework, it is assumed that the model updating associated with the initiation and termination of operations properly maintains these two state variables (as well as state variables relating to the available capacity of resources - see Section 4.2). However, in the context of a specific manufacturing environment, it may be desirable or necessary to extend the representation of current state in various ways, and hence extend the basic model updating that is performed as control decisions are initiated. Implementation of the example in Figure 3-9, for instance, would require extension of the current state representation to include an association of a NBR-OF-CHIP-LAYERS-COMPLETED attribute with production units, and extension of the model updating that must take place upon termination of specific operations.

To enable this capability, two methods, *update-on-op-start* and *update-on-op-end* are also associated with operations. These methods (which by default are noops) are called after the basic model updating has been performed in the cases of operation initiation and termination respectively, and thus provide a mechanism for model extension.

3.6. Operation Descriptions

Given the above primitives for organizing and describing manufacturing activities, we now put the pieces together and consider representation of manufacturing activities themselves. Figure 3-17 depicts the generic definition of **operation**. It contains those attributes and relations previously discussed and identifies three additional attributes:

- an *instantiate-operation* method, which provides the mechanism for instantiating a given operation and establishing relationships with previously instantiated operations,
- an OPERATES-ON attribute, which indicates the object of the operation, and
- a STATISTICS attribute, which serves as a repository for statistics relating to the current instantiations of a given prototype operation.

{{**operation**
IS-A: **activity**
instantiate-operation:
update-on-op-start:
update-on-op-end:
POSSIBLE-SUB-OPERATIONS:
POSSIBLE-SUB-OPERATION-OF
SUBPLAN-ENTRY-OPERATION:
INITIAL-OPERATION-OF:
SUBPLAN-EXIT-OPERATION:
FINAL-OPERATION-OF:
SUB-OPERATIONS:
SUB-OPERATION-OF:
FIRST-SUB-OPERATION:
FIRST-SUB-OPERATION-OF:
LAST-SUB-OPERATION:
LAST-SUB-OPERATION-OF:
TYPE:
range: (OR *and or atomic*)
POSSIBLE-SUCCESSORS:
POSSIBLE-PREDECESSORS:
SUCCESSOR:
PREDECESSOR:
STATUS:
range: (OR *pending inprocess completed*)
EXECUTION-INTERVAL:
range: (TYPE instance **production-activity-interval**)
OPERATES-ON:
STATISTICS:
range: (TYPE instance **operation-stats-report**) }}

Figure 3-17: The **operation** definition

We distinguish between two general types of operations:

- **utilization-operations** - Utilization operations are defined to be those operations whose execution involves utilization of resources (machines, operators, tools, etc.). We can further distinguish three subtypes in this case, depending on the nature of the object being OPERATED-ON, and the type of transformation being modeled:
  - **manufacturing-operation** - Manufacturing operations model activities that operate on production units, transforming their contents and making progress toward the production of final products.
  - **transport-operation** - Transport operations model activities that change the location of production units, specifying movement from one stationary resource to another.
  - **resource-support-operations** - Resource support operations are defined to model those activities required to support the resources required by manufacturing operations (i.e. resource maintenance, repair and setup activities). They operate on specific resources.
- **control-operations** - Control operations are defined to model operations that change and regulate the flow of products through the manufacturing system. These operations do not constitute productive work on the contents of production units but rather act to reconfigure production units and control their movement.

We consider the representation of each of these types of operations in turn in the following subsections.

3.6.1. Utilization Operations

The definition of **utilization-operation** (Figure 3-18) specializes **operation** through the introduction of previously discussed relations pertaining to specification of resource requirements and duration constraints. Further specialization of **utilization-operation** into the subtypes **manufacturing-operation**, **transport-operation**, and **resource-support-operation** (Figure 3-19) imposes appropriate constraints on the type of value that may fill the OPERATED-ON slot and the HAS-DURATION relation. In the case of **transport-operation**, additional attributes relating to location are also defined. We assume for the time being that stationary resources and process-related "control points" (see Section 3.6.2) designate possible production unit locations.

{{**utilization-operation**
IS-A: **operation**
HAS-DURATION:
PRIMARY-RESOURCE:
SECONDARY-RESOURCES: }}

Figure 3-18: The **utilization-operation** definition

{{**manufacturing-operation**
IS-A: **utilization-operation**
OPERATES-ON:
range: (TYPE instance **production-unit**)
HAS-DURATION:
range: (TYPE instance **mfg-op-duration-spec**) }}

{{**resource-support-operation**
IS-A: **utilization-operation**
OPERATES-ON
range: (TYPE instance **resource**)
HAS-DURATION:
range: (TYPE instance **res-support-duration-spec**) }}

{{**transport-operation**
IS-A: **utilization-operation**
OPERATES-ON
range: (TYPE instance **production-unit**)
HAS-DURATION:
range: (TYPE instance **trans-op-duration-spec**)
INITIAL-LOCATION:
range: (OR (TYPE instance **stationary-resource**) (TYPE instance **control-point**))
FINAL-LOCATION:
range: (OR (TYPE instance **stationary-resource**) (TYPE instance **control-point**)) }}

Figure 3-19: Types of **utilization-operations**

Figure 3-20 further refines the **utilization-operation** type hierarchy by defining three different types of

resource-support-operation. These definitions do not, by themselves, provide any differentiating characteristics. They are defined for purposes of easily associating default information within the model of a specific manufacturing system. For example, a single probability distribution for determining operation durations might be appropriate for all repair operations.

{{**repair-operation**
IS-A: **resource-support-operation** }}

{{**maintenance-operation**
IS-A: **resource-support-operation** }}

{{**setup-operation**
IS-A: **resource-support-operation** }}

Figure 3-20: Types of **resource-support-operations**

The above definitions of **manufacturing-operation**, **transport-operation**, and **resource-support-operation** presume corresponding specializations of basic **duration-spec** introduced in Section 3.4. We consider these specializations in the following paragraphs.

Figure 3-21 lists the definitions of **duration-spec** specializations relevant to **manufacturing-operations**. The **mfg-op-duration-spec** is distinguished by the introduction of a generic *get-expected-setup-duration* method. Setup duration in this case is assumed to be a function of the setup-matrix associated with the required primary resource (the concept of **setup-matrix** will be discussed in Section 4). The method defined simply consults this structure. The **mfg-op-duration-spec** subtype is further specialized into **batch-op-duration-spec** and **piece-dependent-duration-spec** to define two basic types of *get-expected-run-duration* calculations. These specializations distinguish respectively cases where operation run time is independent of the number of items in the production unit (e.g. "wash" operations in semi-conductor manufacturing environments), and cases where operation run time is a function of the number of items in the production unit (e.g. wafer "etching" operations). Both specializations add an appropriate calculation parameter.

{{**mfg-op-duration-spec**
IS-A: **duration-spec**
get-expected-setup-duration: calc-resource-setup-duration }}

{{**batch-op-duration-spec**
IS-A: **mfg-op-duration-spec**
get-expected-run-duration: calc-batch-op-run-duration
RUN-DURATION: }}

{{**piece-dependent-duration-spec**
IS-A: **mfg-op-duration-spec**
get-expected-run-duration: calc-piece-dep-run-duration
DURATION-PER-PIECE: }}

Figure 3-21: mfg-op-duration-specs and its subtypes

Definitions of *get-expected-run-duration* can certainly be more complex. If, for example, we were defining an iterative manufacturing process, as in Figure 3-9, then it might be appropriate to differentiate

run duration parameters according to aspects of the current production state. Supposing this to be the case, we could extend the example of Figure 3-9 by defining "level-dependent" variants of **batch-op-duration-spec** and **piece-dependent-duration-spec**, substituting *get-expected-run-duration* methods that derive the current level of the production unit being OPERATED-ON by the operation, and interpret the contents of the parameter slot as a list of <level value> pairs.

Figure 3-22 defines a specialization of **duration-spec** relevant to **transport-operations**. In this case, the both *get-expected-setup-duration* and *get-expected-run-duration* calculation methods are defined in terms of distances between source and destination locations. Setup duration is defined to be the time necessary for the resource to travel from the FINAL-LOCATION of the last transport operation performed to the INITIAL-LOCATION of the current operation. Run duration is similarly defined in terms of the INITIAL-LOCATION and FINAL-LOCATION of the current operation. In both cases, a distance is retrieved from the defined DISTANCE-MATRIX and multiplied by the DURATION-PER-DISTANCE-UNIT parameter.[7] Note that the same DISTANCE-MATRIX will be applicable to all instances of **trans-op-duration-specs**. Only the DURATION-PER-DISTANCE-UNIT can vary across instances.[8]

{{**trans-op-duration-spec**
IS-A: **duration-spec**
get-expected-setup-duration: calc-trans-op-setup-duration
get-expected-run-duration: calc-trans-op-run-duration
DISTANCE-MATRIX:
DURATION-PER-DISTANCE-UNIT: }}

Figure 3-22: The **trans-op-duration-spec** definition

Finally, we define the **res-support-duration-spec** (see Figure 3-23) for describing duration constraints associated with **resource-support-operations**. In this case, the defined *get-expected-setup-duration* and *get-expected-run-duration* methods are defined in turns of specified RESPONSE-TIME and SUPPORT-OP-DURATION parameters respectively.

{{**res-support-duration-spec**
IS-A: **duration-spec**
get-expected-setup-duration: calc-support-prep-duration
RESPONSE-TIME:
get-expected-run-duration: calc-support-op-duration
SUPPORT-OP-DURATION: }}

Figure 3-23: The **res-support-duration-spec** definition

---

[7]Within the implementation, the distance matrix is actually implemented as a hash table with a default value of 0. This exploits the fact that the matrix is symmetric and economizes the amount of storage required.

[8]One obvious alternative to the use of a distance matrix is to simply retrieve the location values of the appropriate stationary resources and/or control points, and directly compute the distance. We believe this approach is perfectly reasonable (and perhaps preferable) when it is possible. However, since the modeling framework assumes that the locations of all entities are defined relative to the locations of stationary resources and control points, this alternative is not possible in manufacturing environments where there are no stationary resources. In such cases, the distance matrix provides a framework for defining suitable approximations.

3.6.2. Control Operations

In contrast to the various types of utilization operations discussed above, control operations are concerned with the reconfiguration of production units traveling through the manufacturing system and regulation of their movement. Reconfiguration of production units during the production process is a common phenomenon in many manufacturing environments. Production units may be merged into larger units or split into smaller units at various stages due to material handling constraints (e.g. pallet capacities in an automated manufacturing cell). Alternatively, yield problems at a particular inspection point in the manufacturing process may dictate that a production unit be split and defective elements routed through a sequence of repair operations. Depending on the specific control policies in force, the remaining portion of the production unit may be sent on in the manufacturing process or held until defective elements are repaired and can be reunited. A third example of reconfiguration arises in the context of complex setup procedures. In wafer fabrication, selected elements of a given production unit are sometimes "sent ahead" through a particular process for purposes of verifying that the manufacturing equipment is correctly calibrated. In such cases, further movement of the remainder of the production unit is contingent on the outcome of the send ahead process. Control operations provide a basis for modeling these sorts of production unit reconfiguration activities.

Figure 3-24 provides the prototypical definition of a **control-operation**. The definition specializes **operation** by designating an associated CONTROL-POINT, which specifies the locus of the control operation (see the definition of **control-point** below). As stated above, a control operation OPERATES-ON a given production unit. Control operations do not require resources, and either occur instantaneously or have an indefinite duration that is depends entirely on the execution of other activities in the manufacturing system.

{{**control-operation**
IS-A: **operation**
CONTROL-POINT:
range: (TYPE instance **control-point**)
OPERATES-ON:
range: (TYPE instance **production-unit**) }}

Figure 3-24: The **control-operation** definition

As indicated above, the distinguishing characteristic of a control operation is its **control-point** (see Figure 3-25). A **control-point** designates a particular QUEUE (or store) of production units, and has a well-defined LOCATION within the manufacturing system. A particular **control-point** is seen as the locus of a specific process-related control activity (e.g. a place where production units are reconfigured to accommodate the material transport constraints on a specific manufacturing subprocess). A production unit enters the queue of a given control point when it is necessary to perform this control activity (specified by a control operation that OPERATES-ON that production unit and designates the control point in question). For purposes of simulating model behavior [5], a **control-point** is ascribed responsibility for managing control operations. To this end, a **control-point** has an associated *execute-op* method, which defines the overall semantics of executing a control operation (i.e. the changes it implies with respect to the state of the model). This method, in turn, relies on any parameters specified in the specific control operation being executed (see below) as well as a specified *control-policy*. The *control-policy* associated with a control point defines the specific decision procedure to be applied in reconfiguring and releasing the production units that reside in the control point's QUEUE.

{{**control-point**
execute-op:
control-policy:
LOCATION:
range: (LIST integer integer)
QUEUE:
range: (SET (TYPE instance **production-unit**)) }}

Figure 3-25: The **control-point** definition

Three basic types of control operations are distinguished (depicted in Figure 3-26):

- **split-operation** - This type of control operation involves reformulation of the production unit designated by OPERATES-ON into two or more smaller production units. In this case, a NEW-PRODUCTION-UNIT-SUCCESSORS attribute is introduced to designate the first operation to be performed on all but one of the newly created production units. It is assumed that the POSSIBLE-SUCCESSORS relation designates a continuing route for the one remaining production unit resulting from the split. The *control-policy* contained in the associated CONTROL-POINT specifies a procedure for splitting the production unit. This procedure may reflect a decision policy as the name implies (e.g. a partitioning of an *n* element production unit into a 2 element "send ahead" sub-unit and an *n-2* element "hold back" sub-unit) or model observed behavior of the manufacturing system (e.g. a yield function that partitions the elements of the production unit into "good" and "bad" sub-units).

- **join-operation** - This type of control operation involves the merger of the production unit designated by OPERATES-ON with one or more other production units currently in the QUEUE of the operation's associated CONTROL-POINT. Here, the CONTROL-POLICY contained in the associated CONTROL-POINT is a procedure that designates the circumstances under which production units residing in the queue can be joined and proceed past the control point (e.g. rejoining the above mentioned "good" sub-unit with the "bad" sub-unit once it has been repaired and enters the queue). As is the case with **utilization-operations**, the POSSIBLE-SUCCESSORS relation designates the continuing route of the single production unit produced by this operation.

- **wait-operation** - This type of control operation serves to delay any further movement of the production unit designated by OPERATES-ON until a specific causal condition involving a separate production unit becomes true. As indicated in the discussion of connection specs in Section 3.2, this is operationalized by posting triggering "message" with the **wait-operation** when the condition becomes true. To this end, a MESSAGE-LIST is associated with each wait operation. Note, that the there is no CONTROL-POLICY associated with the CONTROL-POINTS of **wait-operations**. In this case, there is no decision-making that must take place; when the triggering message is received, the production unit is released.

{{**split-operation**
IS-A: **control-operation**
NEW-PRODUCTION-UNIT-SUCCESSORS:
CONTROL-POINT:
range: (TYPE instance **split-control-point**) }}

{{**join-operation**
IS-A: **control-operation**
CONTROL-POINT:
range: (TYPE instance **join-control-point**) }}

{{**wait-operation**
IS-A: **control-operation**
MESSAGE-LIST:
CONTROL-POINT:
range: (TYPE instance **wait-control-point**) }}

Figure 3-26: Types of control operations

Correspondent to these three basic types of control operations, three types of control points are also defined: **split-control-point**, **join-control-point**, and **wait-control-point** (see Figure 3-27). These types of control points are distinguished by the presence of distinct *execute-op* methods.

{{**split-control-point**
IS-A: **control-point**
execute-op: split-pu-transition-fun }}

{{**join-control-point**
IS-A: **control-point**
execute-op: join-pu-transition-fun }}

{{**wait-control-point**
IS-A: **control-point**
execute-op: release-pu-transition-fun }}

Figure 3-27: Types of control points

To provide a basis for modeling process-related control activities at different levels of abstraction, an **abstract-split-operation** and an **abstract-join-operation** are additionally defined (see Figure 3-28). As can be seen, these types of abstractions are associated both the properties of the corresponding basic control operation and the properties of a **manufacturing-operation**. This is due to the fact that an abstract control operation will often encapsulate a subprocess consisting of both types of more primitive operations.

{{**abstract-split**
IS-A: **split-operation manufacturing-operation** }}

{{**abstract-join**
IS-A: **join-operation manufacturing-operation** }}

Figure 3-28: Abstract control operations

The reader is referred to [5] for further details regarding the interpretation of control operations and examples of their use.

4. Modeling Resources

In this section we consider representation of the resources required to perform manufacturing activities. We first address issues relating to the development of hierarchical descriptions of resources. We then consider, in turn, representation of the various constraints that affect resource allocation. This will provide us with a basis for subsequent development of descriptions of the specific types of resources encountered in manufacturing environments.

4.1. Hierarchical Descriptions of Resources

The framework for hierarchical specification of manufacturing processes described in Section 3.1 is motivated by a desire to enable reasoning about resource allocation at different levels of precision. We have seen in the representation of **operations** that resource requirements of a given operation are uniformly designated as percentages of specific resources, regardless of the position of the operation in the hierarchical model. At abstract levels, it is thus assumed that resource requirements are expressed in terms of aggregate resources. It has already been stated that the determination of appropriate process abstraction levels in any particular manufacturing environment is largely a function of the utility of various resource abstractions from an allocation perspective. Resource abstractions (defined as aggregate resources) are intended to encapsulate and isolate meaningful sets of control decisions.

Reasoning about resource allocation at multiple levels of abstraction implies a representation of aggregate resources that is interpretable from two distinct perspectives:

- *aggregate resource as an allocatable entity* - In the context of reasoning about resource allocation at a given abstract level in the hierarchy, aggregate resources define the entities to be allocated. As such, the representation must provide appropriate abstractions of the allocation constraints associated with the sub-resources that the aggregate resource abstracts (e.g. capacity constraints, availability constraints, setup constraints, etc.).
- *aggregate resource as a set of constituent sub-resources* - In moving across levels of abstraction in the model, aggregate resources define the set of resources relevant to the more detailed resource allocation decisions that must be made. In this regard, the representation must define the relationship of the aggregate resource to its constituents, as well as any knowledge relevant to sub-resource allocation (e.g. allocation preferences among the alternatives abstracted by a given aggregate resource).

These representational requirements motivate our approach to modeling resources.

As implied above, we assume that the basic objective in developing hierarchical resource descriptions is specification of groups of *functionally* related resources. A group of resources is defined to be functionally related if either

1. they provide manufacturing alternatives relative to a given process step or

2. they are configured for consecutive utilization within a multi-step process.

Hierarchical resource descriptions based on this organizing principle are, of course, not unrelated to the actual configuration of manufacturing system being modeled. Indeed, the design of manufacturing systems is typically also motivated by functional grouping objectives (e.g. resources are partitioned into work areas which support particular production processes and/or process steps). Thus, the types of abstractions defined above often correspond very directly to identifiable structural components of the actual manufacturing system. Generally speaking, the set of resource abstractions of interest in constructing a hierarchical model will be a superset of those implied by a structural decomposition of the actual manufacturing system.

Consideration of just those resource abstractions (or functional groupings) that have structural counterparts in the actual manufacturing system leads to a hierarchical organization of resources that partitions the resources defined at any given level of abstraction into mutually exclusive resource sets at the next higher level. Such an organization offers considerable advantages from the standpoint of maintaining descriptions of current available capacity (see Section 4.2 below), and we rely on mutually exclusive resource partitions to provide the "backbone" of any defined organization of resources. We introduce the **sub-resources/sub-resource-of** relation pair (Figure 4-1) for purposes of specifying such partitions. These relations are defined to associate aggregate resources with their constituent sub-resources (and vice versa) under the assumption of a mutually exclusive hierarchical partitioning. Thus, a resource can be a SUB-RESOURCE-OF at most one higher level aggregate. An additional PCTG-OF-AGGREGATE-CAPACITY attribute is associated with the **sub-resource-of** relation to further specify the relationship between a resource and its aggregate. We defer consideration of the semantics of this information until Section 4.2 below, where resource capacity constraints are discussed.

{{**sub-resources**
IS-A: **relation has-parts**
DOMAIN: (TYPE instance **resource**)
RANGE: (SET (TYPE instance **resource**))
INVERSE: **sub-resource-of** }}

{{**sub-resource-of**
IS-A: **relation part-of**
DOMAIN: (TYPE instance **resource**)
RANGE: (TYPE instance **resource**)
INVERSE: **sub-resources**
PCTG-OF-AGGREGATE-CAPACITY: }}

Figure 4-1: The **sub-resource-of/sub-resources** relations

From the standpoint of reasoning about resource allocation, hierarchical resource descriptions based solely on the use of the **sub-resources/sub-resource-of** relation pair can prove insufficient in some situations. To illustrate this, we consider an example of their use. Figure 4-2 depicts a work area of machines that supports a particular production process step within a hypothetical manufacturing system. The work area is composed of three different types of machines: type A machines, type B machines, and type C machines. Each machine type is assumed to possess distinct operating characteristics, and thus take variable amounts of time to perform specific operations. Given this work area configuration, we can identify three levels of detail at which control/allocation decisions might be modeled:

- at the work area level, where the characteristics of constituent machines are appropriately aggregated (e.g. operation durations reflect averages over all constituent machine types) and allocation decisions are based on the overall capacity of the work area,

- at the identical machine group level, where tradeoffs between machine types (e.g. varying operation durations, relative reliability) can be factored into allocation decisions, and allocation decisions are based on the respective capacities of each machine group, and
- at the individual machine level, where specific machine assignments are made to pending operations, and tradeoffs between sequencing decisions (e.g. to minimize setup time) can be precisely evaluated.

Assuming appropriate abstractions of constraints such as capacity (which will be considered in following sections), these three levels of description would be represented within the relational framework defined above as indicated in Figure 4-3.

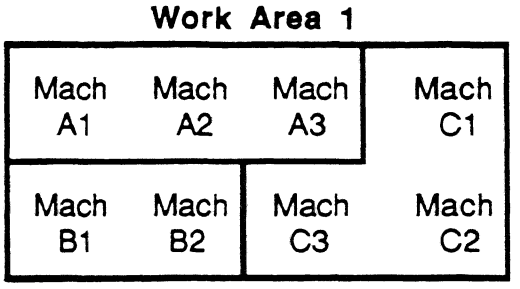

Figure 4-2: A work area of machines supporting a particular process step

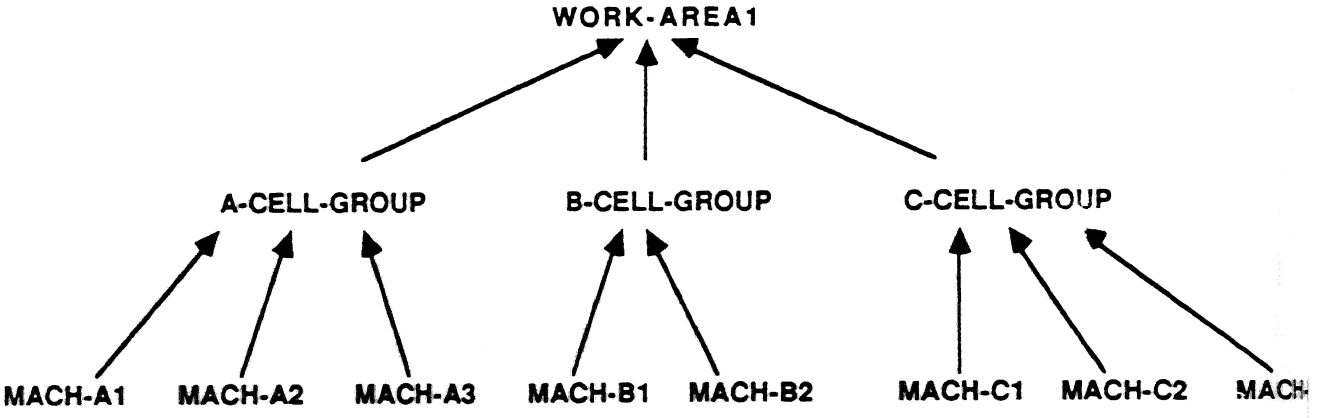

Figure 4-3: Hierarchical representation of the work area

However, suppose that allocation of resources in the work area is also constrained by the types of products that are being manufactured. Figure 4-4 introduces a set of such constraints, which dictate that the process step for product family P1 can only be performed on machine types A and C, that the process step for product family P2 can only be performed on machine types A and B, and that the process step for product family P3 can only be performed on machine type B. Given these additional constraints, the work area level of description in Figure 4-3 no longer provides a meaningful basis for reasoning about resource allocation. Saying this another way, designation of **work-area1** as a required resource of an abstract operation in a specific production process is always misleading. Interpreting **work-area1** as an

allocatable entity, its capacity constraints reflect all resources of the work area while in the context of any specific production process, only a portion of the work area's total capacity is actually relevant. Similarly, **work-area1** indicates all three machine groups as its constituents, while in the context of any specific production process, only a subset of these groups are allocation alternatives.

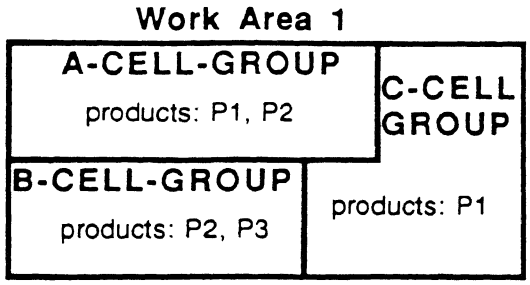

Figure 4-4: A work area of machines with overlapping capabilities

The problem, of course, stems from the fact that the constituent resources of the work area have overlapping capabilities. We thus extend our representational framework to allow definition of aggregate resources that designate overlapping sets of constituent resources. We associate a TYPE with each resource, which can be either *disjoint-aggregation*, *overlapping-aggregation*, or *atomic* (see Figure 4-18), and define an additional set of relations for constructing resource hierarchies that include overlapping aggregations. A given overlapping aggregation is related to the smallest disjoint aggregation that "contains" it (and vice versa) via the **overlapping-sub-resource-of/overlapping-sub-resources** relation pair (Figure 4-5). An overlapping aggregation is related to its constituents (and vice versa) via the **grouped-sub-resources/grouped-in** relation pair (Figure 4-6). Finally, an overlapping aggregation is related to the overlapping aggregations with which it shares constituents via the **overlaps-with** relation (Figure 4-7).

{{**overlapping-sub-resources**
IS-A: **relation has-parts**
DOMAIN: (TYPE instance **resource**)
RANGE: (SET (TYPE instance **resource**))
INVERSE: **overlapping-sub-resource-of** }}

{{**overlapping-sub-resource-of**
IS-A: **relation part-of**
DOMAIN: (TYPE instance **resource**)
RANGE: (TYPE instance **resource**)
INVERSE: **overlapping-sub-resources** }}

Figure 4-5: The **overlapping-sub-resource-of/overlapping-sub-resources** relations

---

{{**grouped-sub-resources**
IS-A: **relation has-parts**
DOMAIN: (TYPE instance **resource**)
RANGE: (SET (TYPE instance **resource**))
INVERSE: **grouped-in** }}

{{**grouped-in**
IS-A: **relation part-of**
DOMAIN: (TYPE instance **resource**)
RANGE: (SET (TYPE instance **resource**))
INVERSE: **grouped-sub-resources** }}

Figure 4-6: The **grouped-sub-resources/grouped-in** relations

{{**overlaps-with**
IS-A: **relation**
DOMAIN: (TYPE instance **resource**)
RANGE: (TYPE instance **resource**)
INVERSE: **overlaps-with** }}

Figure 4-7: The **overlaps-with** relation

The augmented resource hierarchy in the case of our work area example is shown in Figure 4-8. Using this hierarchy, the resources designated as operation requirements at the work area level of precision now become **P1-cell-group**, **P2-cell-group** and **P3-cell-group** within the production process descriptions associated with products P1, P2 and P3 respectively.

4.2. Modeling Capacity Constraints

In both the above discussion of hierarchical resource descriptions and the approach to specifying resource requirements of operations presented in Section 3.3, we have made the assumption that allocation of a resource to a specific activity does not necessarily imply its total unavailability to other activities. Rather unavailability is assumed to be a function of the resource's capacity constraints, and allocation of the resource to an operation implies the unavailability of some percentage of the resource (i.e. a reduction of its *available capacity*) until the operation requiring it subsequently terminates. We can certainly identify types of resources whose allocation requires this perspective (e.g. an oven that can simultaneously accommodate several production units). Moreover, reasoning about resource allocation at abstract levels almost always involves partial allocation of resources. In this section we consider representation of the capacity constraints that govern allocation of resources, and the means by which these constraints are used to maintain descriptions of current available capacity.

4.2.1. Capacity-related Attributes

Generally speaking, we define the CAPACITY of a resource to be the number of items that the resource can process simultaneously. We use the term "items" here, in part, to reflect the fact that different types of **utilization-operations** (i.e. those that require resources) operate on different types of entities. Manufacturing and transport operations manipulate production units; resource support operations manipulate other resources. However, we also wish to define the notion of capacity in a manner that is independent of the number of activities that might be simultaneously supported. For example, the

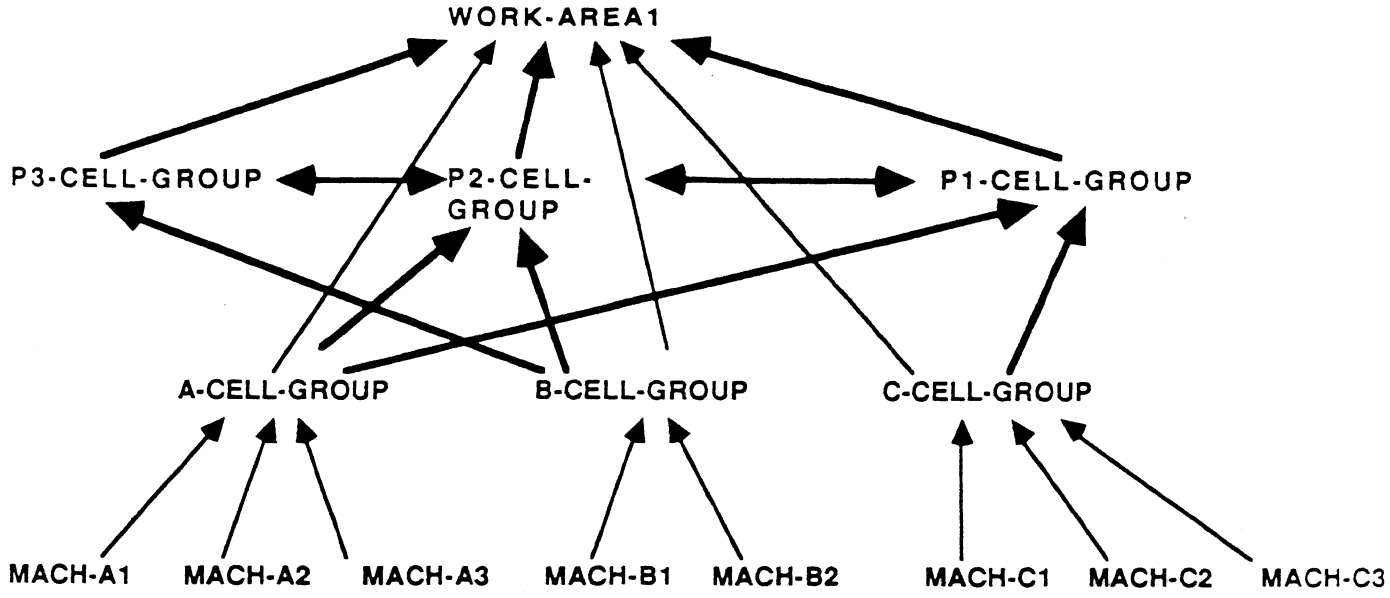

Figure 4-8: Augmented hierarchical representation of the work area

production unit designated by a specific manufacturing (or transport) operation actually represents a group of products that are moving together through the manufacturing system, and it is these products that are actually transformed (or moved) during execution of the operation. The capacities of resources required by this operation are thus defined in terms of the number of products that each resource can simultaneously support. In the case of a resource required by resource support operations, capacity is the number of resources that can simultaneously be supported (e.g. a machine repairman would have a capacity of 1; the overall pool of repair personnel would have a capacity equal to the number of people in the pool).

A resource may also have constraints on the manner in which capacity can be allocated. Consider, for example, a "wash" operation in the context of semi-conductor manufacturing, where wafers are soaked for some period of time in a chemical bath. The bath might have the capacity to hold 50 wafers. However, once the bath has been allocated to a particular production unit (or batched set of production units), it is unavailable for other use for the duration of this wash operation, regardless of the number of wafers actually undergoing the operation. Such constraints are modeled by associating a BATCH-SIZE with each resource, which defines the multiple of capacity units in which capacity must be allocated. Thus, in the above example, we would specify a BATCH-SIZE of 50 for the bath. Supposing, at a higher level, the definition of an aggregate resource that represents a group of three of these baths, we would specify quantities of 150 and 50 as the aggregate resource's CAPACITY and BATCH-SIZE respectively.

4.2.2. Available Capacity

Of central importance in reasoning about resource allocation, of course, is a representation of the state of availability of resources over time. In our terms, this corresponds to a representation of that portion of resource's total CAPACITY that is available for allocation at any point in time.

Given the CAPACITY and BATCH-SIZE constraints defined above and the specification of resource requirements in Section 3.3, computation of a resource's available capacity is straightforward. Suppose

we designate the available capacity of resource R at time t as $avail\text{–}cap_{R,t}$, and assume an initial value $avail\text{–}cap_{R,t0}=capacity_R$. Then the change in $avail\text{–}cap_{R,t}$ as a result of R's allocation to operation op at time $t1$ is simply

$$avail\text{–}cap_{R,t1} = avail\text{–}cap_{R,t1-1} - max(cap\text{–}req_{op}, batch\text{–}size_R).$$

Similarly, the change in $avail\text{–}cap_{R,t}$ upon termination of op at time $t2$ is

$$avail\text{–}cap_{R,t2} = avail\text{–}cap_{R,t2-1} + max(cap\text{–}req_{op}, batch\text{–}size_R).$$

The important issue from the standpoint of representation here concerns support for reasoning about the future. If resource allocation decisions are to be made in a strictly time ordered manner (e.g. in the context of a forward simulation) with no anticipation of or expectations about future system behavior, then maintenance of a scalar available capacity value for each resource (as described above) is sufficient. In our representation of resources, we associate a CURRENT-CAPACITY attribute which has precisely these semantics. If, on the other hand, resource allocation decisions are to be contemplated in advance of their occurrence (i.e. either schedules are developed or some amount of look ahead search is performed) and the results used to guide factory operations, then a representation that depicts the evolution of each resource's available capacity over some future planning horizon is additionally required. We thus also associate an AVAILABLE-CAPACITY structure with each resource, which provides this representation of anticipated resource utilization. A resource's AVAILABLE-CAPACITY is represented as an ordered sequence of **capacity-intervals** (Figure 4-9), with each interval indicating the activities that are anticipated to be consuming capacity within its temporal scope and the capacity that remains available. Further details of this representation and its use in scheduling may be found in [8].

{{**capacity-interval**
IS-A: **calendar-time-interval**
CAPACITY:
CONSUMERS:
range: (SET (TYPE instance **utilization-operation**)) }}

Figure 4-9: The **capacity-interval** definition

4.2.3. Relating capacity constraints at different levels

Use of a hierarchical model of the manufacturing system raises an additional issue relative to resource capacity constraints: that of maintaining consistency in the available capacity of resources defined at different levels of abstraction[9]. If a machine breaks down, for example, the loss in available capacity must be reflected not only in the description of the broken machine, but also in the descriptions of every aggregate resource that "contains" the machine as a sub-resource.

Propagation of changes in available capacity through different levels in the resource hierarchy requires knowledge of the percentage of capacity that each sub-resource contributes to the overall CAPACITY of a given aggregate resource. In the case of disjoint abstractions, this information is encoded with the instance of the **sub-resource-of** relation linking a given sub-resource to the abstraction. Recalling this relation's definition in Figure 4-1, we see that PCTG-OF-AGGREGATE-CAPACITY is defined as an attribute of the relation. We assume that the value of PCTG-OF-AGGREGATE-CAPACITY is a value between 0 and 1, and that the sum of the values associated with all sub-resources of a given abstraction equals 1. Given this information, a change of n units in the available capacity of a subresource R is defined to result in a

---

[9]Here we are using the term "available capacity" in a general sense to refer to either the CURRENT-CAPACITY or AVAILABLE-CAPACITY representation defined above.

change of

$$\frac{n}{capacity_R} \times pctg\text{-}of\text{-}aggregate\text{-}capacity_R$$

in the available capacity of the related aggregate resource.

In the case of overlapping abstractions, where sub-resources are related to the abstraction via the **grouped-in** relation (Figure 4-6), we define a similar updating scheme. However, since a resource can belong to more than one overlapping abstraction, we cannot associate PCTG-OF-AGGREGATE-CAPACITY with the relation itself. We must instead associate this information with the specific values in the range of the relation as meta-information. For this purpose, we define a **pctg-of-capacity-spec** (Figure 4-10), whose PERCENTAGE attribute has the same interpretation as defined above. The **pctg-of-capacity-spec** is also used to describe the **overlaps-with** relations associated with overlapping abstractions (in this context reflecting the percentage of resources shared by two overlapping abstractions). When the available capacity of an overlapping abstraction changes this information is used to update the descriptions of each aggregate resource that OVERLAPS-WITH the overlapping abstraction. Specification of PCTG-OF-AGGREGATE-CAPACITY information relative to relations between an overlapping abstraction and its "containing" disjoint abstraction is done in the same manner as in the case of the **sub-resource-of** relation, since the **overlapping-sub-resource-of** relation used to define such linkages (Figure 4-5) is also a one-to-one relation. Note, however, that change is propagated across **overlapping-sub-resource-of** relations only if the overlapping abstraction constitutes the lowest level of precision at which the system is reasoning. Otherwise, change will be propagated to the containing disjoint abstraction through **sub-resource-of** relations.

{{**pctg-of-capacity-spec**
IS-A: **conceptual-object**
PERCENTAGE: }}

Figure 4-10: The **pctg-of-capacity-spec** definition

Determination of appropriate "percentage of aggregate capacity" values depends on the nature of the set defined by a given aggregate resource. If the aggregate resource represents a set of manufacturing alternatives, then percentages are straightforwardly defined (i.e. the CAPACITY of the aggregate is simply the sum of the capacities of its constituents). The situation is more ill-defined in cases where the aggregate resource represents a set of consecutively utilized resources. Here the CAPACITY of the aggregate may be dominated by the CAPACITY of one or more "bottleneck" resources. We have no compelling methodology for specification of percentages in this case, and rely instead on estimates based on performance characteristics of the actual manufacturing system.

4.3. Modeling Resource Setup Constraints

We associate a **setup-matrix** with each resource as a means of specifying setup duration constraints. The object defines a MATRIX of durations and a *retrieve-duration* method for accessing the structure. The function implementing the method is assumed to be defined in terms of two parameters: the operation requiring the setup and the set of operations last utilizing the resource.

We distinguish between two basic types of **setup-matrix**, a **config-dependent-setup-matrix** and a **location-dependent-setup-matrix**. In the case of **config-dependent-setup-matrix**, setup duration is assumed to be a function of the difference in the resource configuration state implied by the last operation performed on the resource and the resource configuration state implied by the current operation. The

retrieve-duration method derives the value of the "state-defining" attribute for each operation (e.g. the type of product contained in the production unit OPERATED-ON by each operation) and uses these values as indices into the SETUP-MATRIX. The matrix itself defines durations for each possible index pair.

In the case of **location-dependent-setup-matrix**, the matrix is precisely the same distance matrix that was defined for **trans-op-duration-spec** in Section 3.6. The *retrieve-duration* method derives "location" indices in the same manner as previously specified.[10].

{{**setup-matrix**
IS-A: **object**
retrieve-duration:
MATRIX: }}

{{**config-dependent-setup-matrix**
IS-A: **setup-matrix**
retrieve-duration: get-stationary-resource-setup }}

{{**location-dependent-setup-matrix**
IS-A: **setup-matrix**
retrieve-duration: get-mobile-resource-setup }}

Figure 4-11: setup-matrix definitions

4.4. Modeling Work Shift Constraints

Work shifts specifications provide a framework for representing the periods of time during which specific resources are to be operational. More generally, work shift specifications provide a solution to representation of *time varying constraints* - constraints which vary in nature over different intervals of time. For example, circumstances may dictate some degree of overtime over a specific period of time, even though shorter operational period is typically adhered to.

A work shift specification defines intervals of operation for a resource (i.e. work shifts) over a specific temporal horizon. The work shift specifications relevant to a specific resource are organized as a rooted tree structure, with each subtree defining an alteration to its parent specification over some portion of the temporal scope of the parent specification. In other words, work shift specs are defined as calendar intervals (which delineate their temporal scope), and the **current-alterations** and **alteration-of** relations used to define the tree structure are equivalent respectively to the "contains" and "during" temporal relations of Allen.

---

[10]The comments made in Section 3.6 regarding use of actual location values as an alternative to the matrix are equally appropriate here.

{{**current-alterations**
IS-A: **relation**
DOMAIN: (TYPE instance **work-shift-spec**)
RANGE: (SET (TYPE instance **work-shift-spec**))
INVERSE: **alteration-of** }}

{{**alteration-of**
IS-A: **relation**
DOMAIN: (TYPE instance **work-shift-spec**)
RANGE: (TYPE instance **work-shift-spec**)
INVERSE: **current-alterations** }}

Figure 4-12: The **current-alterations/alteration-of** relations

{{**work-shift-spec**
IS-A: **calendar-interval**
CURRENT-ALTERATIONS:
ALTERATION-OF:
SHIFTS: }}

Figure 4-13: The **work-shift-spec** definition

A specialization of the **work-shift-spec**, the **work-shift-spec-root**, is defined to represent the root of a given resource's work shift specification tree. It designates, through an additional SPECIALIZATION-OF relation, the name of another work shift specification tree, to be used over periods of time not covered by the root's CURRENT-ALTERATIONS (or if the root has no current alterations). This provides a basis for association of default work shifts with larger areas of the factory (i.e. cell groups) which can be selectively modified for particular subresources in appropriate circumstances. Indeed, the framework is designed to accommodate control policies which dynamically alter work shift specifications according to characteristics of the current production state.

A method *compile-shifts* is also associated with the **work-shift-spec-root**. This method transforms the specification into an array representation (stored in the COMPILED-SHIFTS slot of the resource) that is more efficient from a computational standpoint. Thus, it is assumed that alterations to a given resource's specification are followed by a recompile.

{{**work-shift-spec-root**
IS-A: **work-shift-spec**
compile-shifts: compile-shifts
SPECIALIZATION-OF:
START-TIME: 0
END-TIME: *time-infinite* }}

Figure 4-14: The **work-shift-spec-root** definition

Work-shift-specs contain descriptions of work shifts. A **shift** is defined as an **hours-of-day-interval** with an associated **work-week**. A **work-week** is, in turn, defined as a **days-of-week** interval.

{{**shift**
IS-A: **hours-of-day-interval**
WORK-WEEK: }}

{{**work-week**
IS-A: **days-of-week-interval** }}

Figure 4-15: Shift and **work-week** definitions

Some sample subtypes and instances are given below.

{{**8to4-shift**
IS-A: **shift**
START-TIME: 28800
END-TIME: 57800 }}

{{**mon-fri-wwk**
INSTANCE: **work-week**
START-TIME: 0
END-TIME: 4 }}

{{**5day-1st-shift**
INSTANCE: **8to4-shift**
WORK-WEEK **mon-fri-wwk** }}

Figure 4-16: Shift examples

4.5. Modeling Resource Breakdowns

Another attribute of a resource is its failure characteristics. In modeling resource failure characteristics, we focus on representing two parameters:

- time to next failure - This parameter relates to the frequency at which the associated resource breaks down.
- amount of capacity lost - This parameter relates to the amount of utilization capacity lost as a result of any given breakdown of the resource. It provides a means of modeling "partial" breakdowns of aggregate resources.

These parameters are defined for any given resource through association of a **breakdown-spec** (see Figure 4-17). The **breakdown-spec** provides a framework for specifying probability distributions relative to both of these parameters, and defines a *get-actual-breakdown-parameters* method which returns a sample drawn from each distribution. A second method called *get-expected-breakdown-parameters* is also defined for use in reasoning about the future. It simply returns the specified mean of each distribution.[11]

[11]It should be noted that, unlike the specification of operation duration constraints, the parameters of resource breakdown distributions are defined in absolute terms and therefore can be stored directly with the SIMPAK distribution object.

{{**breakdown-spec**
IS-A: **conceptual-object**
get-breakdown-characteristics: get-mean-time-to-failure get-mean-capacity-lost
get-actual-breakdown-parameters: calc-time-to-failure calc-capacity-lost
TIME-TO-FAILURE-DISTRIBUTION:
range: (TYPE instance **distribution-object**)
CAPACITY-LOST-DISTRIBUTION:
range: (TYPE instance **distribution-object**) }}

Figure 4-17: The **breakdown-spec** definition

One additional issue regarding resource breakdowns is specification of the **repair-operations** required to return a failed resource to an operational state. To this end, we also associate a REPAIR-PLAN with each resource, which specifies the "root" operation of a hierarchical description of the resource repair process.

4.6. Ascribing control responsibility to resources

Analogous to the treatment of **control-points** in Section 3.6.2, control responsibility is also ascribed to resources for purposes of simulating the behavior of the manufacturing system. More specifically, each resource is defined to be responsible for managing three associated queues:

- PENDING-OPERATIONS - which contains a set of utilization operations that are waiting to utilize capacity of the resource,
- INPROCESS-OPERATIONS - which contains the set of operations that are currently utilizing capacity of the resource, and
- COMPLETED-OPERATIONS - which contains the set of operations that have just finished utilizing capacity of the resource.

The semantics underlying the use of these queues (i.e. how the state of the queues change over time) are specified in terms of three basic transition methods: *begin-operation, end-operation*, and *preempt-operation* (the last of which occurs as a result of a resource failure).

Note that of these possible state transitions, only *begin-operation* involves any decision-making (i.e. which of the PENDING-OPERATIONS should be started at a given point, if any). As was done with **control-points**, we encapsulate the specific decision procedure to be used in this case as the resource's *control-policy*. This *control-policy* is applied by the *begin-operation* method when the latter is enabled.

We refer the reader to [5] for further details of these control methods and their use in the context of simulation.

4.7. Resource Descriptions

Having now addressed the major representational issues vis a vis resources, we can now complete the picture by considering the description of resources themselves. Figure 4-18 defines those attributes that are common to all resources. Note that all attributes discussed in previous sections do not appear in this basic definition of **resource**. These missing attributes are characteristic of only certain types of resources and will be introduced as various specializations of **resource** are considered below. The only attributes not previously mentioned are

- STATISTICS, which serves as a repository for statistics relating to utilization of the resource, and
- SCHEDULING-LEVEL and SIMULATION-LEVEL, which are discussed in Section 4.8 below.

{{**resource**
IS-A: **physical-object**
TYPE:
range: (OR disjoint-aggregation overlapping-aggregation atomic)
SUB-RESOURCES:
SUB-RESOURCE-OF:
OVERLAPPING-SUB-RESOURCES:
OVERLAPPING-SUB-RESOURCE-OF:
GROUPED-SUB-RESOURCES:
GROUPED-IN:
OVERLAPS-WITH:
CAPACITY:
CURRENT-CAPACITY:
AVAILABLE-CAPACITY:
range: (SET (TYPE instance **capacity-interval**))
BREAKDOWN-SPEC:
range: (TYPE instance **breakdown-spec**)
REPAIR-PLAN:
range: (TYPE is-a **repair-operation**)
STATISTICS:
range: (TYPE instance **resource-stats-report**)
SCHEDULING-LEVEL:
range: (OR t nil)
SIMULATION-LEVEL:
range: (OR t nil)
DESCRIPTION: }}

Figure 4-18: The **resource** definition

4.7.1. Stationary Resources

Stationary resources are those resources that have a fixed location within the manufacturing system. At the atomic level, stationary resources include individual machines and work stations. At abstract levels, stationary resources are functional work areas composed of collections of machines and/or work stations.

As previously stated, stationary resources will always be designated as primary resource requirements for **manufacturing-operations** if they are present in the manufacturing system being modeled. This being the case, the definition of **stationary-resource** (Figure 4-19) includes the previously discussed methods, queues, and control policy that enable control decisions to be made (i.e. *begin-operation*, *end-operation*, *preempt-operation*, PENDING-OPERATIONS, INPROCESS-OPERATIONS, COMPLETED-OPERATIONS and *control-policy*. We also associate shift constraints (i.e. WORK-SHIFT-SPEC and COMPILED-SHIFTS) with **stationary-resources**, again a consequence of their role as primary resources. Given the nature of setup in the context of a **stationary-resource**, its associated SETUP-MATRIX is constrained to be configuration dependent. Finally, an attribute indicating a stationary resource's LOCATION is introduced.

{{**stationary-resource**
IS-A: **resource**
begin-operation: start-op-transition-fun
end-operation: end-op-transition-fun
preempt-operation: breakdown-transition-fun
control-policy:
PRIMARY-RESOURCE-FOR:
LOCATION:
range: (LIST integer integer)
SETUP-MATRIX:
range: (TYPE instance **config-dependent-setup-matrix**)
WORK-SHIFT-SPEC:
range: (TYPE instance **work-shift-spec-root**)
COMPILED-SHIFTS:
PENDING-OPERATIONS:
range: (SET (TYPE instance **utilization-operation**))
INPROCESS-OPERATIONS:
range: (SET (TYPE instance **utilization-operation**))
COMPLETED-OPERATIONS:
range: (SET (TYPE instance **utilization-operation**)) }}

Figure 4-19: The **stationary-resource** definition

Stationary-resource is specialized into **work-cell** and **cell-group** to distinguish between atomic and aggregate stationary resources.

{{**work-cell**
IS-A: **stationary-resource**
TYPE: *atomic* }}

{{**cell-group**
IS-A: **stationary-resource**
TYPE:
range: (OR disjoint-aggregation overlapping-aggregation) }}

Figure 4-20: atomic and aggregate stationary resources

Finally, with respect to **cell-groups**, we distinguish between the two basic types of functionally-related resource groups defined in Section 4.1: **parallel-cell-groups**, which delineate manufacturing alternatives, and **serial-cell-groups**, which represent resources configured for consecutive utilization.

{{**parallel-cell-group**
IS-A: **cell-group** }}

{{**serial-cell-group**
IS-A: **cell-group** }}

Figure 4-21: Parallel and serial cell groups

4.7.2. Mobile Resources

Mobile resources are those resources that do not have a fixed location in the manufacturing system, but rather move (or are moved) between locations over time. Mobile resources encompass human resources, transport devices, and tools. Since each of these types of resource has fairly unique characteristics, the definition of **mobile-resource** (Figure 4-22) serves only to distinguish between mobile and stationary resources and does not introduce any additional attributes.

{{**mobile-resource**
IS-A: **resource** }}

Figure 4-22: The **mobile-resource** definition

The **human-resource** subtype of **mobile-resource** (Figure 4-23) defines those resources that comprise the human work force. Depending on the characteristics of a given manufacturing environment, human resources might be defined as either primary or secondary resources from the standpoint of allocation. More precisely, we assume that, in the absence of stationary resources, human resources will constitute the primary resource requirements of both **manufacturing-operations** and **resource-support-operations**. Given this potential role, a **human-resource** is defined with the methods and attributes accorded to resources that can participate in control decisions, and shift constraints are also associated. Finally, its SETUP-MATRIX is constrained to be location dependent.

{{**human-resource**
IS-A: **mobile-resource**
begin-operation: start-op-transition-fun
end-operation: end-op-transition-fun
preempt-operation: breakdown-transition-fun
PRIMARY-RESOURCE-FOR:
SECONDARY-RESOURCE-FOR:
SETUP-MATRIX:
range: (TYPE instance **location-dependent-setup-matrix**)
WORK-SHIFT-SPEC:
range: (TYPE instance **work-shift-spec-root**)
COMPILED-SHIFTS:
PENDING-OPERATIONS:
INPROCESS-OPERATIONS:
COMPLETED-OPERATIONS: }}

Figure 4-23: The **human-resource** definition

Human-resource is specialized into **operator** and **operator-group** to distinguish between atomic and aggregate human resources (Figure 4-24).

{{**operator**
IS-A: **human-resource**
TYPE: *atomic* }}

{{**operator-group**
IS-A: **human-resource**
TYPE:
range: (OR disjoint-aggregation overlapping-aggregation) }}

Figure 4-24: atomic and aggregate human resources

The **transport-device** subtype of **mobile-resource** (Figure 4-25) delineates mechanical devices (as opposed to human resources) that are used to transport production units from one location to another. A **transport-device** can only be required by **transport-operations** (see Section 3.6), and in this context will always be the primary resource required. Given this role as a primary resource, a **transport-device** is also defined so as to enable its participation in control decisions.

{{**transport-device**
IS-A: **mobile-resource**
begin-operation: start-op-transition-fun
end-operation: end-op-transition-fun
preempt-operation: breakdown-transition-fun
PRIMARY-RESOURCE-FOR:
PENDING-OPERATIONS:
INPROCESS-OPERATIONS:
COMPLETED-OPERATIONS: }}

Figure 4-25: The **transport-device** definition

A final subtype of **mobile-resource** is **tool-resource** (Figure 4-26). A tool resource can only be defined as a secondary resource, and thus has no role in the control of operations. As with other types of resources, we specialize **tool-resource** into **tool** and **tool-group** (not depicted) to distinguish between atomic and aggregate tool resources respectively.

{{**tool-resource**
IS-A: **mobile-resource**
SECONDARY-RESOURCE-FOR: }}

Figure 4-26: The **tool-resource** definition

4.8. Regulating the level of precision of decision-making

Given a hierarchical model of resources defined according to the representational primitives put forth in the preceding sections, it is straightforward to superimpose a framework for regulating the level of precision at which specific resources are reasoned about. We simply associate a "marker" with each resource indicating whether or not the resource resides at the desired level of precision. In the case of primary resources, the marker is used to determine whether descent to a more detailed description of manufacturing operations (and required resources) is appropriate (since our representation assumes that levels of manufacturing process descriptions mirror the levels of description defined relative to primary resources). In the case of secondary resources, the marker simply bounds the level of precision at which their allocation is considered. Recognizing that it may be desirable to differentiate between the level of precision at which resource allocation decisions are anticipated (i.e. through scheduling) and the level at which control decisions are actually modeled (i.e. through simulation), we actually associate two such markers with each resource: a SCHEDULING-LEVEL and a SIMULATION-LEVEL. In both cases, the value is constrained to be *t* or *nil*.

5. Modeling Products, Demands and Production Units

We now turn attention to representation of the entities that are manipulated and transformed by the manufacturing system. As indicated in Section 3, manufacturing activities are defined to OPERATE-ON **production-units**. Production units represent collections of **products** that are manufactured together and production units are created in response to product **demands**. As such, their representation necessarily relies on representations of related product and demand information. The modeling of these three types of entities is the focus of this section.

Before proceeding, one general comment regarding our modeling perspective is in order. We are interested in models of products and demands from the standpoint of simulation and scheduling, and will address representation issues from this constrained viewpoint. Consideration of other modeling perspectives (e.g. product design, process planning, accounting) would obviously lead to much more comprehensive representations of products and demands.

5.1. Product Descriptions

We refer to the objects actually produced by the manufacturing system, either as final outputs of the system or as input materials to more complex objects, as **products**. Within our modeling framework, product descriptions are intended to provide a basis for organizing information about products that is relevant to its manufacture.

One central characteristic of a product in this regard, of course, is the manufacturing process by which it is realized. We have already addressed the details of representing manufacturing processes (Section 3), and a given product's PRODUCTION-PLAN is defined in this manner. More precisely, this product attribute designates the "root" operation of a hierarchically described prototype manufacturing process (specified according to the representation defined in Section 3). Typically, there is commonality in the manufacturing processes associated with different products. For example, in the semiconductor domain the manufacturing processes of two different types of wafers might vary only in the particular types of masks that are required for exposing operations (see example in Section 3.3). We take commonality of basic manufacturing process as a basis for organizing product descriptions into a type hierarchy. We define the notion of PRODUCT-FAMILY to represent a set of products whose manufacture follows the same basic process, and assume that intermediate levels in the product type hierarchy designate specific product families. Production plans are associated at the level in the type hierarchy at which PRODUCT-FAMILY attributes are defined, and are inheritable by all specific product types defined at lower levels. Individuating characteristics required for interpretation of a PRODUCTION-PLAN relative to a specific product type are associated with the definition of that product type. Such a type hierarchy is illustrated in Figure

5-1.

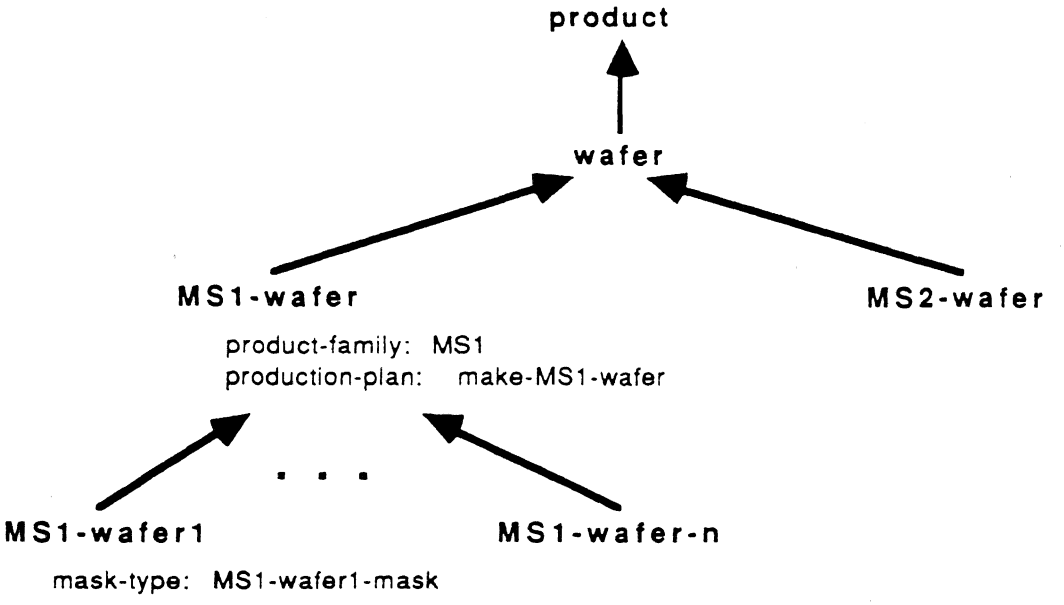

Figure 5-1: An example product type hierarchy

With respect to compositional relationships between products, which are important from the standpoint of mapping product demands to production requirements, we can identify two general cases:

- subassembly/assembly relationships - Much of manufacturing consists of the production of products which then become component materials for the production of composite products. In such cases, demands for composite products must be decomposed into demands for component products to determine production requirements.
- aggregate/disaggregate relationships - In other situations, manufacturing consists of the production of products which are then disaggregated into other products (e.g. manufacturing wafers and then dicing them into chips). In these cases, demands for final products must be aggregated into demands for internally manufactured products to determine production requirements.

These two situations can be treated uniformly by viewing the relationships from a material requirements perspective. Production of a composite product requires the prior production of each component material (product). Similarly, production of products through disaggregation requires the prior production of the material (product) to be disaggregated.

We define the **material-requirements/material-requirement-for** relation pair (Figure 5-2 to express these product relationships. In the case of **material-requirement-for**, we assume that a **quantity-spec** (Figure 5-3) is attached as meta-information to each value in the range of the relation, indicating the "amount of material" that is required for production of the range product. In cases where the **material-requirement-for** relation is used to relate a component product to a composite product, QUANTITY simply designates the number of that component that is required for production of the composite product. In cases where **material-requirement-for** is used to relate an aggregate product to a disaggregated final product, QUANTITY indicates the percentage of the aggregate product required to produce one disaggregated product. Figure 5-4 illustrates the use of the **material-requirements** and

material-requirement-for relations to express inter-product relationships.

{{**material-requirements**
IS-A: **relation**
DOMAIN: (TYPE instance **product**)
RANGE: (SET (TYPE instance **product**))
INVERSE: **material-requirement-for** }}

{{**material-requirement-for**
IS-A: **relation**
DOMAIN: (TYPE instance **product**)
RANGE: (SET (TYPE instance **product**))
INVERSE: **material-requirements** }}

Figure 5-2: The **material-requirements/material-requirement-for** relations

---

{{**quantity-spec**
IS-A: **conceptual-object**
QUANTITY: }}

Figure 5-3: The **quantity-spec** definition

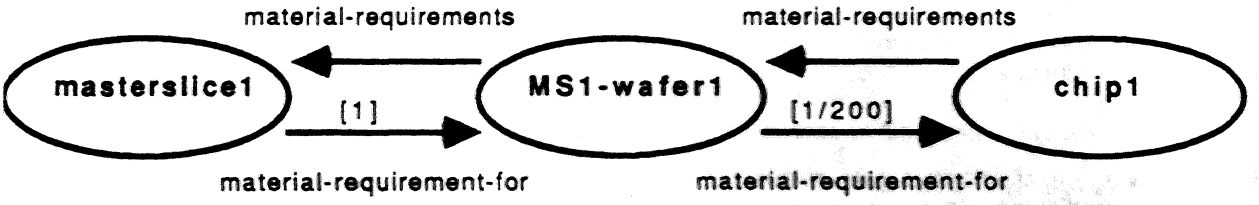

Figure 5-4: An example of inter-product relationships

We assume that the **material-requirements** relation defines a conjunctive set of requirements, and thus situations where material alternatives exist must be modeled by defining separate "material requiring" products. This assumption seems reasonable in the case of assemblies, since subcomponents are typically rigidly defined during product design and we would expect experimental designs to be separately named. And in situations of production by disaggregation, where there is often flexibility in the mapping of internally manufactured products to the disaggregated final products, we claim that the assumption is not as restrictive as it might first appear. Consider an example from the semiconductor manufacturing domain, where top level demands are for chips and the production of chips (in our terms) requires wafers as materials. Typically there is a one to one correspondence between chip types and wafer types, but some wafer fabrication technologies provide the flexibility to manufacture different chip types on the same wafer. This implies the possibility of alternative material requirements for a given chip type and, given the above modeling assumption, the need for multiple product definitions. However, given the magnitudes of the quantities of chips that are typically ordered, there would seem to be little to be

gained (at the expense of considerable additional complexity) from the unconstrained placement of different chip types on a given wafer. More realistically, multiple chip type wafers would be defined relative to typical chip relationships (e.g. the types of chips required for a particular device), which map quite naturally to the definition of specific end products (e.g. computer1-chip-set).

In situations where the production of a product requires the prior production of input materials, some indication of the time required to produce the end product once input materials are available is needed to properly establish the manufacturing requirements (e.g. due dates) relative to intermediate products. We associate an AVERAGE-LEAD-TIME with each product description which contains this information.

A final general attribute of relevance to a product's manufacture is its PRODUCTION-QUANTITY. Products are often manufactured in specific quantities due either to economies of scale or physical constraints relating to their manufacture (e.g. product movement). A product's PRODUCTION-QUANTITY simply indicates the batch size[12]. This information constrains the instantiation of the **production-units** that are actually manipulated by the manufacturing system.

Figure 5-5 lists the complete definition of **product**. In addition to the characteristics just discussed, we specify a PRODUCED-BY relation, which associates a product type description with currently instantiated production units that are composed of products of that type. This relation will be discussed in Section 5.3 below.

{{**product**
IS-A: **physical-object**
PRODUCT-FAMILY:
PRODUCTION-PLAN:
range: (TYPE is-a **operation**)
PRODUCTION-QUANTITY:
MATERIAL-REQUIREMENTS:
MATERIAL-REQUIREMENT-FOR:
AVERAGE-LEAD-TIME:
PRODUCED-BY: }}

Figure 5-5: The **product** definition

5.2. Demand descriptions

We introduce the concept of a **demand** (see Figure 5-6) to represent an obligation for delivery of products that the manufacturing system has committed to. Generally speaking, A **demand** specifies REQUESTS for quantities of specific products to be satisfied within a REQUESTED-PRODUCTION-INTERVAL. Demand REQUESTS are mapped into PRODUCTION-REQUIREMENTS, which represent the total set of product requests that the manufacturing system must satisfy (i.e. both intermediate and end products). A demand's PRODUCTION-REQUIREMENTS, in turn, guide the formation for **production-units** for release to the manufacturing system.

From the standpoint of scheduling and control of the manufacturing system, demands also specify information relating to their relative priority. We identify two such attributes (assumed to be a function of the demand's INITIATOR):

---

[12]Of course the PRODUCTION-QUANTITY will be 1 in cases where products are manufactured individually

- PRIORITY-CLASS - The partitioning of demands into a discrete set of priority classes is a commonly employed method for specifying the relative importance of the associated demand. Priority classes provide one basis for differentiating among alternative scheduling and control policies. We assume the possible values of this attribute to be domain-dependent (e.g. *hot*, *red-hot*, etc.).
- TARDY-COST - Tardy cost is defined to be the cost penalty per time unit for late delivery of the requested products. Tardy costs provide an additional basis for trading off alternative resource allocation decisions.

{{**demand**
IS-A: **conceptual-object**
REQUESTS:
PRODUCTION-REQUIREMENTS:
INITIATOR:
REQUESTED-PRODUCTION-INTERVAL:
range: (TYPE instance **calendar-interval**)
PRIORITY-CLASS:
TARDY-COST:
CONTRACT-DATE: }}

Figure 5-6: The **demand** definition

We specialize **demand** into three subtypes, according to the nature of the demand's INITIATOR:

- **customer-order** - Customer orders represent external demands that have been placed on the manufacturing system
- **engineering-work-order** - Engineering work orders represent internally generated demands relating to testing of product manufacturing processes and product changes.
- **stock-order** - Stock orders represent internally generated demands for purposes of maintaining inventory levels.

Given our modeling interests (i.e. simulation and scheduling), the motivation for this demand type hierarchy is to provide a basis for default specification of PRIORITY-CLASS and TARDY-COST values. In a broader modeling context (e.g. encompassing order administration and accounting model interpretations), these definitions would necessarily require elaboration.

{{**customer-order**
IS-A: **demand**
INITIATOR:
range: (TYPE instance **customer**) }}

{{**engineering-work-order**
IS-A: **demand**
INITIATOR:
range: (TYPE instance **engineer**) }}

{{**stock-order**
IS-A: **demand**
INITIATOR:
range: (TYPE instance **stock-manager**) }}

Figure 5-7: Specializations of **demand**

Specific demand requests and production requirements are uniformly represented as **production-requests** (Figure 5-8). A **production-request** specifies a request for some QUANTITY of a specific PRODUCT. INITIATOR requests are linked to demands via the **requests/request-of** relation pair (See Figure 5-9). The production requirements that are implied by the specified INITIATOR requests are related to the demand via the **production-requirement-of/production-requirements** relation pair (see Figure 5-10). Note that **production-requirement-for** is defined as an inheritance relation, extending the definition of **production-request** in the context of production requirements to include REQUESTED-PRODUCTION-INTERVAL, PRIORITY-CLASS, and TARDY-COST. In this case, the value of the associated demand's REQUESTED-PRODUCTION-INTERVAL is interpreted as a default which may be overridden during determination of production requirements (e.g. the due dates of component materials of an assembly must allow time for the assembly). A **satisfied-by** relation is also introduced in the definition of **production-request** for purposes of relating production requirements to **production-units**. This relation is discussed below in Section 5-16.

{{**production-request**
IS-A: **conceptual-object**
REQUEST-OF:
PRODUCTION-REQUIREMENT-OF:
SATISFIED-BY:
PRODUCT:
range: (TYPE instance **product**)
QUANTITY: }}

Figure 5-8: The **production-request** definition

{{**requests**
IS-A: **relation**
DOMAIN: (TYPE instance **demand**)
RANGE: (SET (TYPE instance **production-request**))
INVERSE: **request-of** }}

{{**request-of**
IS-A: **relation**
DOMAIN: (TYPE instance **production-request**)
RANGE: (TYPE instance **demand**)
INVERSE: **requests** }}

Figure 5-9: The **requests/request-of** relations

{{**production-requirements**
IS-A: **relation**
DOMAIN: (TYPE instance **demand**)
RANGE: (SET (TYPE instance **production-request**))
INVERSE: **production-requirement-of** }}

{{**production-requirement-of**
IS-A: **relation**
DOMAIN: (TYPE instance **production-request**)
RANGE: (TYPE instance **demand**)
INVERSE: **production-requirements**
INCLUSION: {instance **inclusion-spec**
TYPE: slot
SLOT-RESTRICTION: requested-production-interval tardy-cost priority-class} }}

Figure 5-10: The **production-requirements/production-requirement-of** relations

In Figure 5-11, we graphically illustrate the representation of a particular demand at a point after production requirements have been determined. In this example, a request for chips (whose production is assumed to be defined as a wafer dicing activity) leads to an additional wafer production requirement.[13]

5.3. Production unit descriptions

Production units model the objects that are actually manipulated by the manufacturing system. They are defined to represent sets of one or more product instances which travel through the system and are manufactured together. As we have previously stated, production units draw part of their definition from characteristics of their member products and part of their definition from the demands that they are designated to satisfy. With respect to grouping products into production units, the only constraint we impose is that all member products belong to the same product family (i.e. all member product types share the same PRODUCTION-PLAN). With respect to associating production units with demands, we allow a

[13]Of course, the decision to include chips as a production requirement and model the corresponding dicing activity is up to the model builder, who must specify the mapping from requests to production requirements. Specification of this mapping is considered below in Section 5.4.

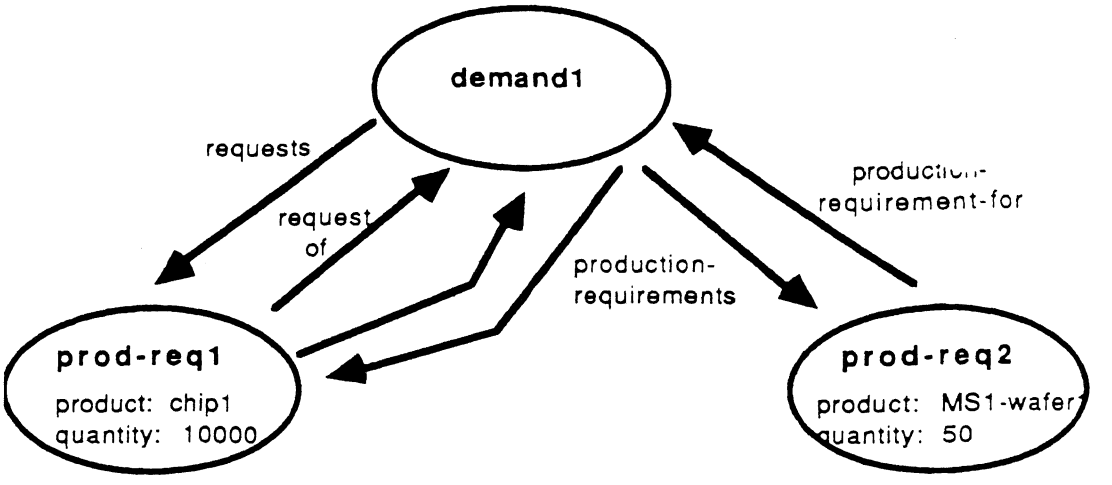

Figure 5-11: An example demand with associated production requirements

production unit to contribute to the satisfaction of more than one production request. Both of these assumptions are less restrictive than is required in many manufacturing environments. However, the actual policy employed to configure production units can be specified to match the characteristics of the specific manufacturing system being modeled (see Section 5.4).

In addressing the representation of production units, we first define relations for use in associating production units to products and demands, and then present their complete definition.

5.3.1. Relating production units to products

We define the **produces/produced-by** relation pair (see Figure5-12 to associate respectively a production unit with the *type(s)* of products that the production unit will ultimately "produce", and a product type with all currently instantiated production units that are "producing" products of that type. The **produces** relation is defined as an inheritance relation which provides a production unit with the prototype PRODUCTION-PLAN required for its manufacture. Since we impose the constraint that all product types grouped within a given production unit must share the same PRODUCTION-PLAN, inheritance of this information in situations where production units are comprised of multiple product types presents no problems (i.e. inheritance from any of the product types designated by the **produces** relation will yield the same value).[14]

[14]The possibly of multiple associated product types does however introduce some complications relative to further exploitation of **produces** as an automatic inference mechanism. We have seen in Sections 3 that various aspects of manufacturing activities may rely on product-dependent information (e.g. the specification of secondary resources), and one might consider redefinition of the **produces** inheritance semantics to make this information automatically inferrible from the production unit. However, in situations where multiple product types are associated, the desired inference might be (1) the value associated with any of the product types, (2) a list of the values associated with each product type, or (3) some value that is a function of the values associated with each product type. The first two cases can be managed through appropriate setting of the *$inherit-all* inheritance control switch, but the third case is not an inference that is supported by the CRL inheritance mechanism. An approach to modeling this last type of situation is considered below relative to the problem of relating demands to production units.

{{**produced-by**
IS-A: **relation**
DOMAIN: (TYPE is-a **product**)
RANGE: (SET (TYPE instance **production-unit**))
INVERSE: **produces** }}

{{**produces**
IS-A: **relation**
DOMAIN: (TYPE instance **production-unit**)
RANGE: (SET (TYPE is-a **product**))
INVERSE: **produced-by**
INCLUSION: {instance **inclusion-spec**
TYPE: slot
SLOT-RESTRICTION: production-plan } }}

Figure 5-12: The **produced-by/produces** relations

The definition of the **produces/produced-by** relation pair in terms of *instances* of production units and *types* of products is motivated by a desire to provide flexibility with respect to the manner in which materials and material flows are modeled. In many modeling contexts, product instances have no distinguishing characteristics from a manufacturing perspective, and it is sufficient to model the contents of production units and inventories of materials as numerical quantities. In modeling turbine blade production, for example, there is little to be gained by representing each individual blade in the system. On the other hand, there are modeling contexts which do require an explicit representation of product instances. Consider the problem of evaluating various scrapping policies in the wafer fabrication domain. One set of possible policies might be based on the number of times a given wafer has undergone rework, which would require a model that represents individual wafers and maintains a REWORK-COUNT with each one. The **has-members/member-of** relation pair is defined to support such situations.

{{**member-of**
IS-A: **relation**
DOMAIN: (TYPE instance **product**)
RANGE: (TYPE instance **production-unit**)
INVERSE: **has-members** }}

{{**has-members**
IS-A: **relation**
DOMAIN: (TYPE instance **production-unit**)
RANGE: (SET (TYPE instance **product**))
INVERSE: **member-of** }}

Figure 5-13: The **member-of/has-members** relations

5.3.2. Relating production units to demands

The **satisfies-request/satisfied-by** relation pair (see Figure 5-14) provides primitives for relating production units to the production requirements they are intended to satisfy (and vice versa). As with the earlier defined **produces** relation, we define **satisfies-request** as an inheritance relation, in this case to allow automatic inference of demand-dependent information that is relevant to the control of production units. The attributes REQUESTED-PRODUCTION-INTERVAL, PRIORITY-CLASS, and TARDY-COST are contributed

to the definition of production units through this relation.

{{**satisfied-by**
IS-A: **relation**
DOMAIN: (TYPE is-a **production-request**)
RANGE: (SET (TYPE instance **production-unit**))
INVERSE: **satisfies-request** }}

{{**satisfies-request**
IS-A: **relation**
DOMAIN: (TYPE instance **production-unit**)
RANGE: (SET (TYPE is-a **production-request**))
INVERSE: **satisfied-by**
INCLUSION: {instance **inclusion-spec**
TYPE: slot
SLOT-RESTRICTION: requested-production-interval priority-class tardy-cost } }}

Figure 5-14: The **satisfied-by/satisfies-request** relations

---

It is important to note that the inheritance of values through the **satisfies-request** relation is only meaningful in modeling contexts where the mapping from production units to production requirements is constrained to be one-to-one. Since such a mapping represents common practice in many manufacturing environments the relation is defined in this manner. However, if production units are allowed to contribute to the satisfaction of more than one production requirement, then only attributes (and not values) can be inherited through the **satisfies-request** relation and methods must be provided to derive the values. To this end, we add the methods *get-requested-production-interval*, *get-priority-class* and *get-tardy-cost* to the definition of **production-unit**. These methods, if they exist, are invoked to establish values by *instantiate-production-unit* (discussed below).

The mapping from production requirements to production units specified by the **satisfied-by** relation is always assumed to be one-to-many. Thus, we assume attachment of a **quantity-spec** (recall Figure 5-3) as meta-information to each value designated in the range of a given **satisfied-by** relation to indicate the portion of the production requirement to be satisfied by the associated production unit

Figure 5-15 provides an example of production unit, production request, and product relationships.

5.3.3. Other Attributes

In addition to the information that is attributed to production units by virtue of relationships to specific product types and production requests, productions units have defining characteristics of their own. These characteristics are identified in the definition of **production-unit** contained in Figure 5-16, along with the relations and methods identified above. In the following paragraphs, we describe these additional attributes.

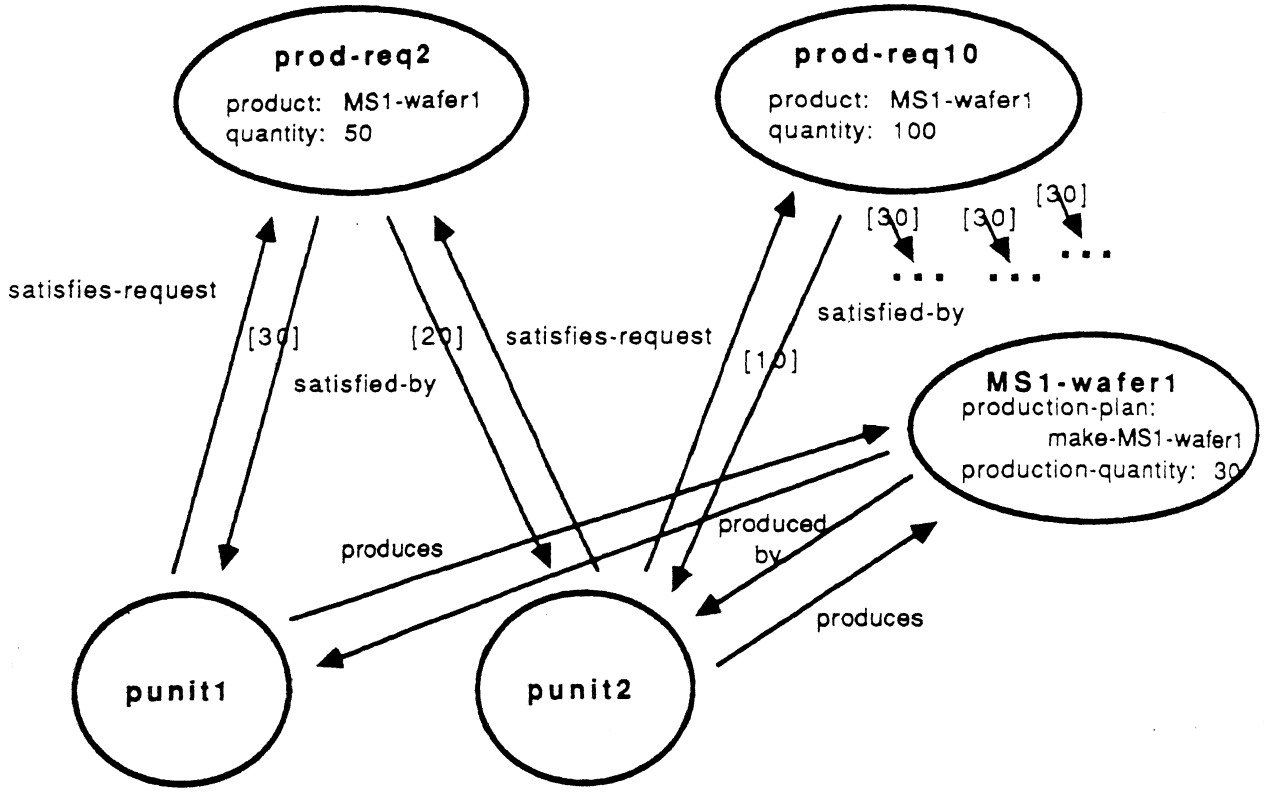

Figure 5-15: Production unit, production request, and product relationships

---

{{**production-unit**
IS-A: **physical-object set**
instantiate-production-unit: create-p-unit
get-requested-production-interval:
get-priority-class:
get-tardy-cost:
PRODUCES:
HAS-MEMBERS:
QUANTITY:
SATISFIES-REQUEST:
INSTANTIATED-PLAN:
range: (TYPE instance **operation**)
CURRENT-STATE:
HAS-SUB-UNITS:
SUB-UNIT-OF:
STATISTICS:
range: (TYPE instance **punit-stats-report**) }}

Figure 5-16: The **production-unit** definition

One obvious characteristic of a production unit that is important from the standpoint of coordination is its QUANTITY, which is simply the number of products that the production unit contains. As we have seen in

Section 3, this attribute can be important in interpreting various aspects of the production unit's prototype PRODUCTION-PLAN (e.g. operation duration constraints). Production units also have attributes related to their current manufacturing state. One attribute that provides such information is the production unit's INSTANTIATED-PLAN. As we have previously discussed, interpretation of the prototype production plan as the production unit travels through the manufacturing system (or prior to its manufacture if advance schedules are to be developed) results in the creation of a network of instantiated operations. At any point in time, a production unit's instantiated production plan, or more specifically the instantiated root operation that is actually stored as the value of INSTANTIATED-PLAN, contains a STATUS and EXECUTION-INTERVAL that reflect basic aspects of the production unit's current state. We also define and associate a CURRENT-STATE attribute with the production unit itself, to provide a framework for user-defined extensions to the representation. In the wafer fabrication example presented in Figure 3-9 of Section 3.2, for instance, CURRENT-STATE might be defined to take on values such as *level1*, *level2*, etc.

There are often situations that dictate a reconfiguration of production units at some point during the manufacturing process. For example, the outcome of a particular "test" operation may indicate that some percentage of the production unit must be re-routed for rework operations. As we have already seen in Section 3.6.2, such situations are modeled within prototype production plans through the use of various split and join operations. Instantiation of these operations in the context of specific production units, however, requires an ability to model the relationship between an initial production unit and any decomposition of that unit that has transpired. To this end, we define the **has-sub-units/sub-unit-of** relation pair (see Figure 5-17). The **sub-unit-of** relation is defined to allow inheritance of all "parent" production unit attributes except those relating to its composition.[15]

{{**has-sub-units**
IS-A: **relation**
DOMAIN: (TYPE instance **production-unit**)
RANGE: (SET (TYPE instance **production-unit**))
INVERSE: **sub-unit-of** }}

{{**sub-unit-of**
IS-A: **relation**
DOMAIN: (TYPE instance **production-unit**)
RANGE: (TYPE instance **production-unit**)
INVERSE: **has-sub-units**
INCLUSION: {INSTANCE **inclusion-spec**
TYPE: **slot**
SLOT-RESTRICTION: (NOT quantity has-members produces) } }}

Figure 5-17: The **sub-unit-of/has-sub-units** relation pair

The following two attributes complete the definition of **production-unit** listed in Figure 5-16:

- *instantiate-production-unit*, which is a method that is called to instantiate a particular production unit, and
- STATISTICS, which serves as a repository for statistics relating to the behavior of production units.

---

[15]The value of the **produces** relation is excluded here because of the possibility that a subcomponent could consist of only a subset of the types of products "produced" by the parent. Recall, however, that all product types appearing in a given production unit must share the same production plan, so there is no complication in inheriting the INSTANTIATED-PLAN.

5.4. Mapping demands to production units

As suggested in the above discussion of **demands**, **products**, and **production-units**, the activity of mapping a set of current demands into a set of production units to be manufactured is conceptualized within the modeling framework as a two step process:

1. the set of demand requests is first translated into a set of production requirements, and
2. a set of production units is then formulated whose manufacture will satisfy the set of production requirements.

Given a manufacturing model, the first step is well-defined; it simply involves interpretation of the material requirements associated with the product specified in each demand request (i.e. the explosion of product "bills of materials" to determine all production requirements). The second step, alternatively, is much more a function of the characteristics and policies practiced in a given manufacturing environment. For example, in a computer board assembly and test facility where products are manufactured individually, the mapping from production requirements to production units is straightforward and invariant. On the other hand, in environments where products are produced in batches (e.g. wafer fabrication, turbine blade production), more complex possibilities exist (e.g. mapping multiple production requirements to a single production unit) and a variety of policies are practiced. Furthermore, production unit formation typically involves consideration of current inventories (i.e. it may be possible to directly satisfy some portion of the production requirements), and, in some cases, consideration of tradeoffs between manufacturing or sub-contracting.

To provide a framework for modeling the production unit formation practices of a specific manufacturing facility (or, more generally, for analyzing the effects of different production unit formation strategies on overall system performance) , a **demand-manager** object is defined (Figure 5-18). This object specifies two methods which govern the mapping of demands to production units in accordance with the above conceptualization. The first, *generate-production-requirements*, is a generic, pre-defined procedure. The second, *generate-production-units*, is a method to be constructed by the model builder (i.e. a decision-making policy much like the *control-policies* associated with resources and control points). From the standpoint of providing an experimental environment, we also assume that the **demand-manager** contains a *demand-generator* and a related set of DEMAND-GENERATOR-PARAMETERS.

{{**demand-manager**
generate-demands: demand-generator-fun
DEMAND-GENERATOR-PARAMETERS:
generate-production-requirements: explode-material-requirements-fun
generate-production-units:
CURRENT-DEMANDS:
range: (SET (TYPE **instance demand**)) }}

Figure 5-18: The **demand-manager** definition

---

6. Final Remarks

Our aim in this paper has been to define a modeling framework that supports realistic simulation and scheduling of large-scale manufacturing systems. In this regard, we believe the modeling primitives presented in the preceding sections provide a structural base for constructing interpretable models that reflect the full complexity of a particular manufacturing facility. Before closing, we briefly consider some aspects of this modeling framework that remain underspecified.

The modeling framework as presented does not provide a complete semantics with respect to material flow within a manufacturing facility. In particular, the framework does not explicitly address the issue of

inventory management, which impacts both the demand management process (i.e. whether manufacturing is actually required to satisfy production requirements) and the coordination of assembly processes. On the other hand, extension of the framework to address this issue seems straightforward. All that is missing is the notion of "inventories"; stores into which products (or component products) are moved after they are manufactured and from which products are drawn to satisfy production requirements (or to enable assembly processes). Inventories can be modeled in a manner similar to the **control-points** of Section 3.6.2. If the products comprising production units are explicitly modeled, then the structure maintained by an inventory is similarly a queue (of products in this case). Otherwise, a set of scalar quantities is maintained.

The specification of control policies is another aspect of the modeling framework that requires further consideration. From the standpoint of both expressibility and flexibility, a more declarative framework for expressing control decision procedures would be preferable to the current use of attached methods. One general approach to this problem is to view a control policy as a specification of a set of decision-making preferences, and focus on developing an interpretable representation of preferences. In this regard, the constraint representation defined in [3] is directly relevant.

Acknowledgements

The modeling framework presented in this paper has evolved through the collaborative efforts of the OPIS scheduling group. Don Kosy, Claude LePape, Nicola Muscettola, Peng Si Ow, and Chris Young, in particular, have each contributed substantially to its development. Thanks also to Don Kosy for his helpful comments on earlier drafts of this document.

References

[1] Cleveland, G.A.
The Temporal Database of the HSTS Space Telescope Observation Scheduler.
ISL Working Paper, Robotics Institute, Carnegie Mellon University, 1989.

[2] Dean, T.
Temporal Imagery:An Approach to Reasoning about Time for Planning and Problem Solving.
PhD thesis, Yale University, Computer Science Department, October, 1985.

[3] Fox, M.S., and S.F. Smith.
ISIS: A Knowledge-Based System for Factory Scheduling.
Expert Systems 1(1):25-49, July, 1984.

[4] *Knowledgecraft Reference Manual*
Carnegie Group Inc., Pittsburgh, PA, 1986.

[5] Kosy, D.
A Simulation Kernel for OPIS Models of Factory Operation.
ISL Working Paper, Intelligent Systems Laboratory, Carnegie Mellon University, January, 1989.

[6] LePape, C. and S.F. Smith.
Management of Temporal Constraints for Factory Scheduling.
In C. Rolland, M. Leonard, and F. Bodart (editors), *Proceedings IFIP TC 8/WG 8.1 Working Conference on Temporal Aspects in Information Systems (TAIS 87).* Elsevier Science Publishers, held in Sophia Antipolis, France, May, 1987.

[7] Muscettola, N. and S.F. Smith.
State-Based Scheduling: An Architecture for Space Telescope Observation Scheduling.
In *1989 NASA Conference on Space TeleRobotics.* Pasadena, CA, January, 1989.

[8] Smith, S.F.
A Constraint-Based Framework for Reactive Management of Factory Schedules.
In M. Oliff (editor), *Proceedings 1st International Conference on Expert Systems and the Leading Edge in Production Management.* Charleston, SC, May, 1987.