Configuration tasks select and arrange instances of parts from a set. Computational models for configuration are used for build-to-order manufacturing tasks and also for tasks such as planning and therapy recommendation. Incremental decision-making methods for configuration must cope with threshold effects and horizon effects.

8

Configuration

8.1 Introduction

A **configuration** is an arrangement of parts. A **configuration task** is a problem-solving activity that selects and arranges combinations of parts to satisfy given specifications. Configuration tasks are ubiquitous. Kitchen designers configure cabinets from catalogs of modular cupboards, counters, closets, drawers, cutting boards, racks, and other elements. Salespeople for home entertainment systems must design configurations from tape decks, optical disk players, receivers, turntables, and other units that can play recordings from different media, as well as from video screens, speakers, and other units that can present performances to people. Computer engineers and salespeople configure computer systems from various computing devices, memory devices, input and output devices, buses, and software. Dieticians configure diets from different combinations of foods. Configuration problems begin with general specifications and end with detailed specifications of what parts are needed and how they are to be arranged.

A definitional characteristic of configuration, as we shall use the term, is that it instantiates components from a predefined, finite set. This characterization of configuration does not create new parts or modify old ones. Configuration tasks are similar to classification tasks in that instantiation includes the *selection* of classes. However, to solve a classification problem is merely to select one (or perhaps a few) from the predefined set of classes. To solve a configuration problem is to instantiate a potentially large *subset* of the predefined classes. The search space of possible solutions to a configuration problem is the *set of all possible subsets* of components. This is the *power set* of the predefined classes. Furthermore, multiple instantiations of the same part may be used in a solution, and different *arrangements* of parts count as different solutions. Configuration problems are usually much harder than classification problems.

8.1.1 *Configuration Models and Configuration Tasks*

Custom manufacturing and mass manufacturing are often seen as opposites. Custom manufacturing tailors products to the specific and different needs of individual customers; mass manufacturing achieves economies of scale when high-volume manufacturing techniques make it possible to reduce the unit costs of nearly identical products. An industrial goal common to many kinds of manufacturing is to combine the flexibility of custom manufacturing with the economics of mass production. An important strategy for this is to create lines of products that can be tailored to a customer's requirements by configuring customer-specific systems from a catalog of mass-produced parts. This approach has been called an a la carte or build-to-order marketing and manufacturing strategy.

A la carte manufacturing requires special capabilities of an organization. For production, the required capabilities include the efficient means for managing of an inventory of parts, for bringing the parts together that are required for each order, for assembling the parts into products, and for testing the quality of the different assembled products. A well-recognized logistic requirement for this process is to accurately understand the customer's needs and to develop a product specification that meets those needs. This step, which actually precedes the others, is the configuration step. The success and efficiency of the entire enterprise depends on configuration.

Since flexible product lines can involve hundreds or thousands of different, configurable parts, there are many possibilities for errors in the configuration process. Errors in a configuration can create costly delays when they are discovered at assembly time, at testing time, or by a customer. An effective configuration process reduces costs by reducing level of inventory, reducing the amount of idle manufacturing space, reducing the delay before customers' bills are paid, and better ensuring customer satisfaction.

As these factors have been recognized, companies have sought automated and semi-automated configuration systems for routinely creating, validating, and pricing configurations to meet the differing requirements of their customers. One of the first and best known knowledge-based configuration systems is the XCON system for configuring VAX computer systems (McDermott, 1981). Knowledge-based systems for configuration tasks are being built by many organizations.

Configuration is a special case of design. In design the elements of the designed artifact are not constrained to come from a predefined set. They are subject only to the constraints of the manufacturing methods and the properties of the raw materials. The knowledge necessary for configuration tasks is more bounded than the knowledge for general design tasks. In general design tasks, the solution space is more open-ended. For example, designing a pickup truck from scratch is more open-ended than choosing options for a new one. In configuration tasks, much of the "design work" goes into defining and characterizing the set of possible parts. The set of parts must be designed so they can be combined systematically and so they cover the desired range of possible functions. Each configuration task depends on the success of the earlier task of designing configurable components.

From a perspective of knowledge engineering, configuration tasks are interesting because they are synthetic or constructive tasks. They are tasks for which combinatorics preclude the pre-enumeration of complete solutions. Just as not all classification tasks are the same, not all configuration tasks are the same. Configuration models can be used as a basis for different kinds of tasks, not necessarily involving manufacturing and physical parts. For example, "configuring"

therapies can be a part of a larger diagnosis and therapy task. Configuring alternative combinations of steps for a plan can be an important part of a planning process. In such cases we refer to a **configuration model** for a task.

Knowledge-engineering techniques are suitable for classification tasks and configuration tasks because most of the complexity in specifying how to carry out the task is in the domain knowledge rather than the methods themselves. The knowledge-level methods define roles for different kinds of knowledge and these roles help to control the way that the bodies of knowledge interact and guide the search for solutions.

8.1.2 *Defining Configuration*

Selecting and arranging parts is the core of solving a configuration problem. Figure 8.1 presents an example of a configuration that is a solution to a configuration problem. In this example, the configuration has two top-level parts: Part-1 and Part-2. Each of these parts has subparts.

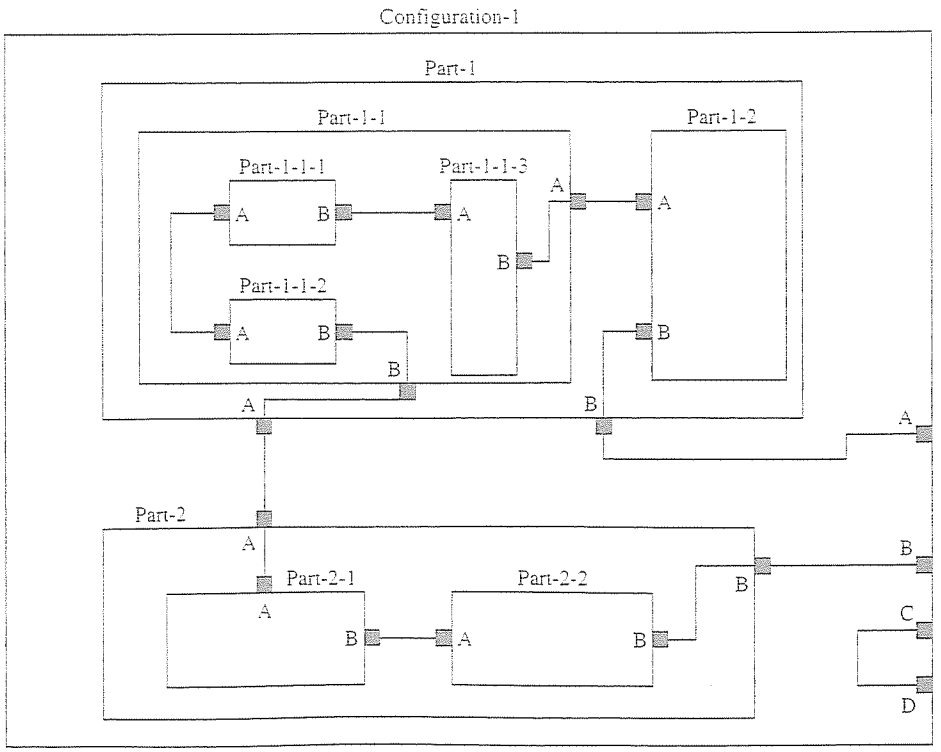

FIGURE 8.1. An example of a configuration using a port-and-connector model. In this figure, all components are shown as boxes and the top-level component is Configuration-1. It has two parts, Part-1 and Part-2. The arrangement of the configuration is indicated by the part-of hierarchy, displayed graphically by the containment of boxes and by the interconnections among parts. Each part has a number of ports, labeled A,B,C, or D. For example, the B port of Part-1-1-1 is connected to the A port of Part-1-1-3.

Configuration domains differ in their representations of arrangements. For example, Figure 8.1 represents arrangements using a **port-and-connector model**. The ports on each part correspond to the different roles that subparts can have with respect to each other. Like all models governing arrangements, a port-and-connector model constrains the ways things can be connected. Components can be connected only in predefined ways. Ports may have type specifications indicating that they can be connected only to objects of a particular kind. Each port can carry only one connection.

Variations in the search requirements and the available domain knowledge for different configuration tasks lead to variations in the methods. Nonetheless, certain regular patterns of knowledge use appear across many different configuration tasks. Configuration methods work through recognizable phases. They map from user specifications to abstract descriptions of a configuration, and they refine abstract solutions to detailed configurations specifying arrangements and further requirements. We distinguish between solution expansion and solution refinement phases, not to imply that they are independent, but to emphasize the distinct meanings of part requirement hierarchies, functional abstraction hierarchies, and physical containment hierarchies. Addition of required components is often a specialized process and can involve goals for both correcting and completing a specification. Refinements of specifications include the consideration of alternative arrangements as well as selection among alternative implementations. Configuration methods must mix and balance several kinds of concerns and different bodies of knowledge.

Figure 8.2 shows the relevant spaces for defining configuration problems. It is a useful starting framework from which we shall consider variations in the following sections. In some domains there is no separate specification language: Specifications are given in the language of

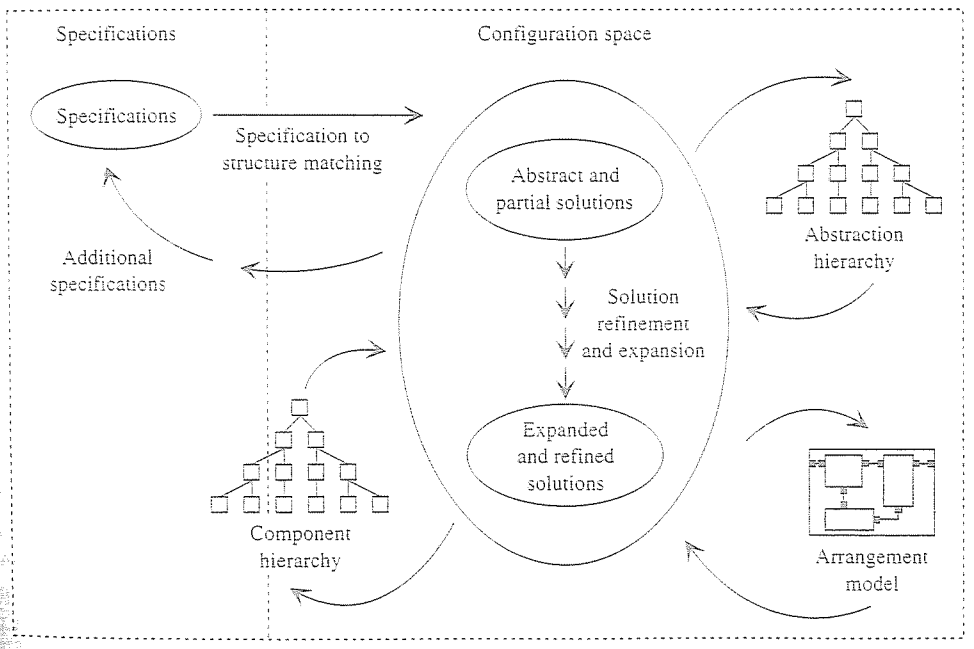

FIGURE 8.2. Spaces in configuration tasks.

parts and arrangements and the task begins with a partial configuration. In that case, we omit the boundary in the figure between the specification space and the configuration space. The outer oval in the configuration space signifies that the representations for partial and expanded solutions are mixed together. Most configuration methods work incrementally on candidates, expanding the candidates to include more parts and refining the specifications to narrow the choices.

By analogy with names for classification methods, the pattern of knowledge use suggested by Figure 8.2 could be called **heuristic configuration** to emphasize the use of heuristic knowledge to guide the generation and testing of possible configurations. It could also be called **hierarchical configuration** to emphasize the importance of reasoning by levels, both component hierarchies and abstraction hierarchies. For simplicity, we just use the term *configuration*.

8.2 Models for Configuration Domains

This section discusses what configuration is and what makes it difficult.

8.2.1 Computational Models of Configuration

To understand configuration tasks we need a computational model of the search spaces and the knowledge that is used. Our model has the following elements:

- A specification language
- A submodel for selecting parts and determining their mutual requirements
- A submodel for arranging parts
- A submodel for sharing parts across multiple uses

We begin by describing these elements generally and simply. We then introduce the "widget" domain as a concrete instantiation of a configuration model. We use this domain to illustrate reasoning and computational phenomena in configuration tasks. Afterward, we draw on the domains from the previous section to extract examples of kinds of knowledge and the use of knowledge.

A **specification language** for a configuration task describes the requirements that configurations must satisfy. These requirements reflect the environment in which the configured product must function and the uses to which it will be put. A specification may also indicate which optimizing or due-process criteria should be used to guide the search, such as minimizing cost or space or preferring some possibilities over others. (See the exercises.)

A functional specification describes capabilities for desired behaviors. For example, rather than saying that a computer configuration needs a "printer," a functional specification might say that the computer needs to be able to print copies. The printing function may be further specialized with qualifiers about speed, resolution, directionality, character sets, sizes of paper, color or black and white, and so on. One advantage of describing systems in terms of functions rather than in terms of specific classes of manufactured parts is that functions can anticipate and simplify the addition of new classes of components to a catalog. When a functional specification language is used, the configuration process must have a means for mapping from function to structure.

A specification language is not defined independently of the other submodels. Some configuration models use the same language for describing specifications and for describing configurations. This approach avoids the function-to-structure mapping by associating each part with a list of key functions. The most common approach is called the **key-component approach**. This approach has two parts: (1) For every major function there is some key component, and (2) all the key components (or abstract versions of them) are included in the initial partial configuration that specifies a configuration. A specification is assumed to include all the key parts. To complete a configuration, a configuration system satisfies the prerequisites of all the key parts it starts with as well as the parts it needs to add to the specification, and it also determines a suitable arrangement for all the included parts.

A **submodel for parts** specifies the kinds of parts that can be selected for a configuration and the requirements that parts have for other parts. Some parts require other parts for their correct functioning or use. For example, computer printed circuit boards may require power supplies, controllers, cables, and cabinets. A part model defines required-component relations, so that when a configuration process considers a part description, it can determine what additional parts are needed. Required parts can be named explicitly or described using the specification language. Descriptions indicate which parts are compatible with each other and what parts can be substituted for each other under particular circumstances.

A **submodel for spatial arrangements** provides a vocabulary for describing the placement of parts and specifies what arrangements of parts are possible. Together, the arrangement model and the specification language form a basis for describing which arrangements are acceptable and preferred. They make it possible to determine such things as where a part could be located in an arrangement, whether there is room for another part given a set of arranged parts, and which parts in a set could be added to an arrangement. Arrangement models constrain the set of possible configurations because they govern the consumption of resources, such as space, adjacency, and connections. These resources are limited and are consumed differently by different arrangements. Arrangement models are a major source of constraints and complexity in configuration domains.

A **submodel for sharing** expresses the conditions under which individual parts can be used to satisfy more than one set of requirements. In the simplest case, part use is mutually exclusive. For example, a cable for one printer cannot be used simultaneously to connect to another printer. For some applications, mutual exclusion is too restrictive, such as with software packages that may use the same memory, but at different times. The following exemplifies a range of categories of use and sharing for configuration systems:

- *Exclusive use*: Components are allocated for unique uses.
- *Limited sharing:* Parts can be shared between certain functions but not across others.
- *Unlimited sharing:* A component can be allocated for as many different purposes as desired.
- *Serial reusability:* A component can be allocated for several different purposes, but only for one purpose at a time.
- *Measured capacity:* Each use of a component uses up a fraction of its capacity. The component can be shared as long as the total use does not exceed the total capacity. The electrical power provided by a supply is an example of a resource that is sometimes modeled this way.

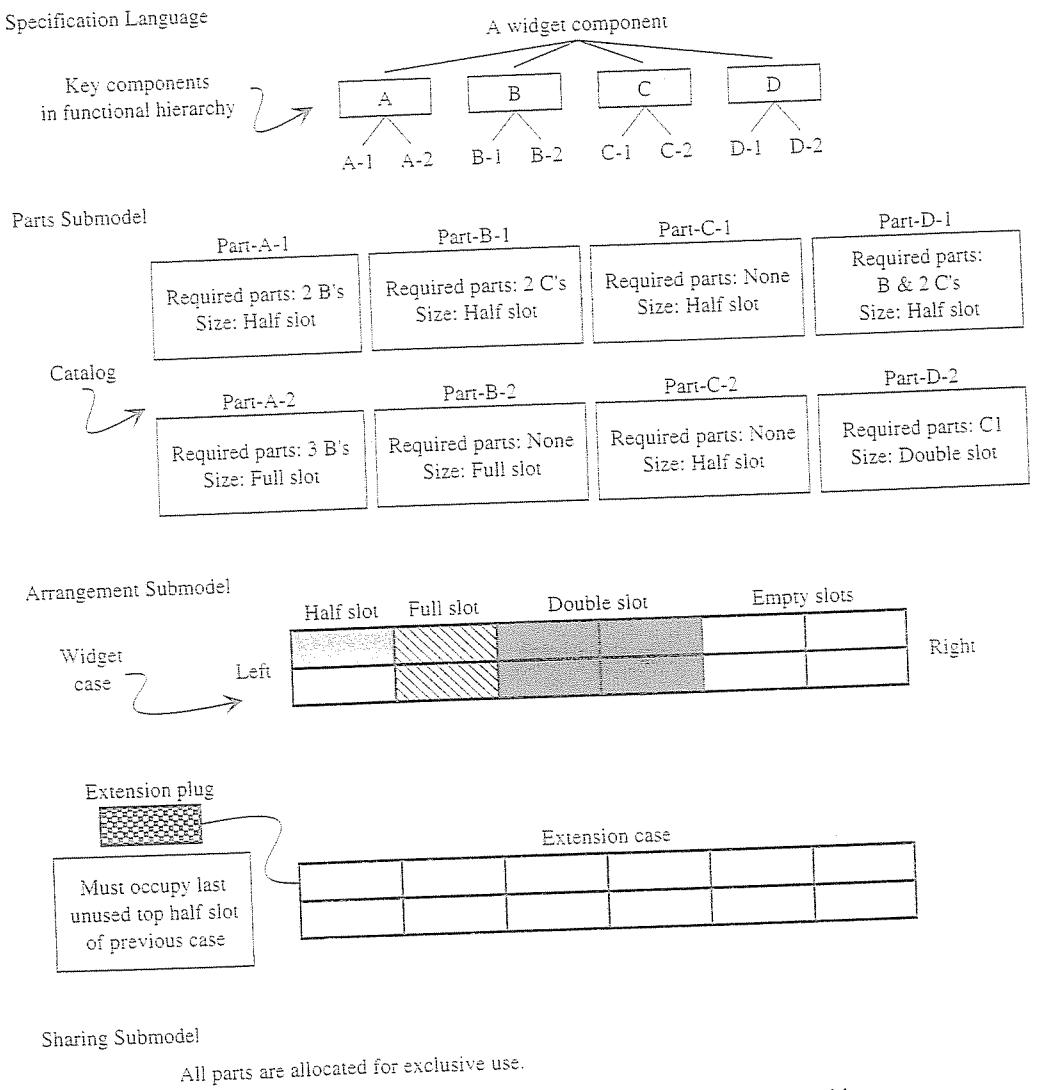

FIGURE 8.3. The widget model, W-1, illustrates phenomena in configuration problems.

A particular domain model may incorporate several of these distinctions, assigning different characteristics to different parts. Sometimes these distinctions can be combined. One way to model how different software packages could share memory would be to post capacity constraints indicating how much memory each package needs. The system allocation for serial usage would be determined as the maximum capacity required for any software package. This approach mixes serial reusability with measured capacity.

In summary, our framework for computational models of configuration has four submodels: a specification language, a submodel for parts and requirements, a submodel for ar-

rangement, and a submodel for component sharing. We return to these models later to describe variations on them and to identify the ways in which they use domain knowledge. In the next section, we instantiate our configuration model for the widget domain and use it to illustrate important phenomena in configuration tasks.

8.2.2 *Phenomena in Configuration Problems*

This section describes reasoning phenomena that arise in configuration problems. These phenomena are illustrated in the context of the sample domain in Figure 8.3. Figure 8.3 presents W-1, a model for configuring widgets. We use W-1 to illustrate configuration phenomena in the following discussion and in the exercises at the end of this section.

Widget requirements are specified in terms of the key generic components A, B, C, and D. The W-1 parts submodel provides a catalog with two choices of selectable components for each generic component. Thus, generic component A can be implemented by either A-1 or A-2, B can be implemented by B-1 or B-2, and so on. Each component requires either a half slot, a full slot, or a double slot. Figure 8.4 presents the rules for the widget model.

When components are arranged in the widget model, there are sometimes unavoidable gaps. When configurations are combined, these gaps are sometimes complementary. For this reason it is not always adequate to characterize the amount of space that a configuration takes up with a single number. Figure 8.5 defines several terms related to measures of space in the widget model. **Occupied space** refers to the number of slots occupied by components, including any required extension plugs. **Trapped space** refers to any unoccupied slots to the left of other slots, including slots to the left of or below an extension plug. **Space** (or total space) is the sum of occupied space and trapped space. The **minimum space** for a specification is the minimum amount of space required by any possible configuration that satisfies the specification.

Required Parts
Each part in a configuration must have all of its required parts, as shown in Figure 8.3.

Arrangement
Components must be arranged in alphabetical order (left to right and top to bottom). Parts with the same letter (e.g., C-1 and C-2) can be mixed in any order.
All identical components (components with the same letter and number) must be in the same widget case.
A full-slot component must occupy a single vertical slot. It cannot be split across slots. A double-slot component must occupy two adjacent full slots in the same widget case.
An upper half slot must be filled before the corresponding lower half slot can be filled.
No full-slot gaps are allowed, except at the right end of a case.
If a case is full, an extension case can be added but it requires using an extension plug in the last upper half slot of the previous case.
When an extension plug is used, no component can be placed in the half slot below the plug.

Sharing
The W-1 sharing submodel requires that distinct parts be allocated to satisfy each requirement.

FIGURE 8.4. Further rules governing the W-1 widget model.

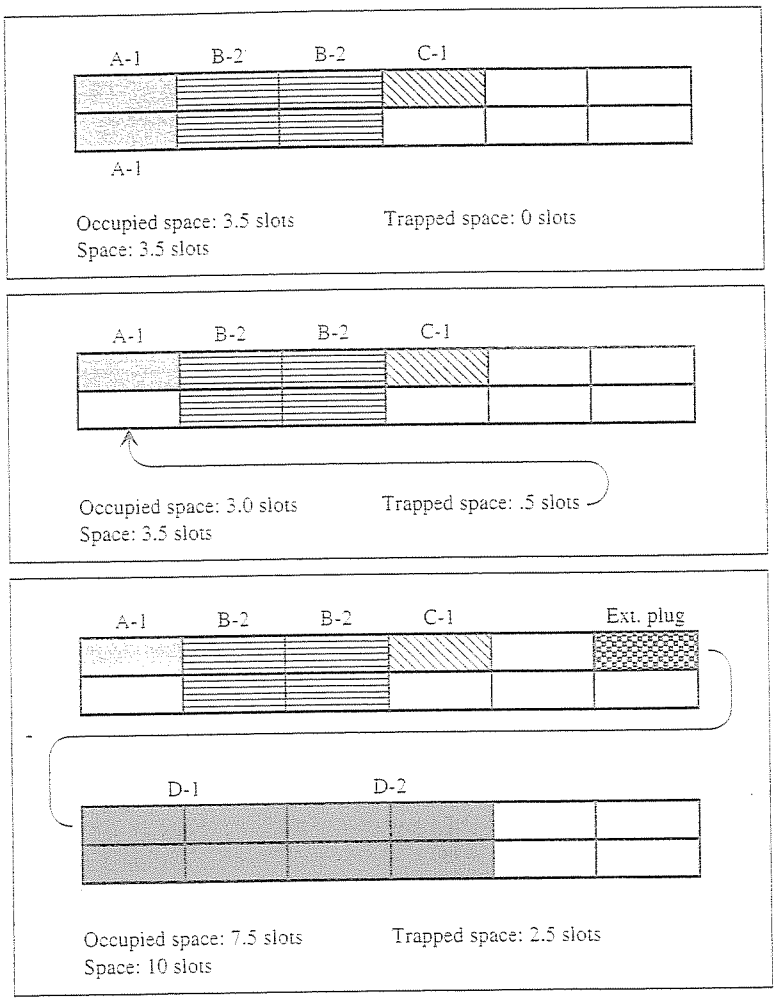

FIGURE 8.5. Examples illustrating the terms total space, occupied space, and trapped space. For most problems, the key factor is the total space, which is called simply "space."

Configuration decisions are made incrementally. Figure 8.6 illustrates stages in a solution process, starting with the initial specifications {A, D}. Under the part-expansion phase, alternatives for required parts for A and D are considered. For requirement A, the candidates are the selectable parts A-1 and A-2. However, A-1 and A-2 both have further requirements. A-1 expands to two B's, which could be implemented as either B-1's or B-2's. If B-1's are chosen, these require C's, which themselves can be implemented as either C-1's or C-2's. This expansion of requirements shows why we cannot use a simple constraint-satisfaction method with a fixed number of variables and domains to solve the configuration task for the top-level parts. Depending on which alternatives are chosen, different required parts will be needed, and different fur-

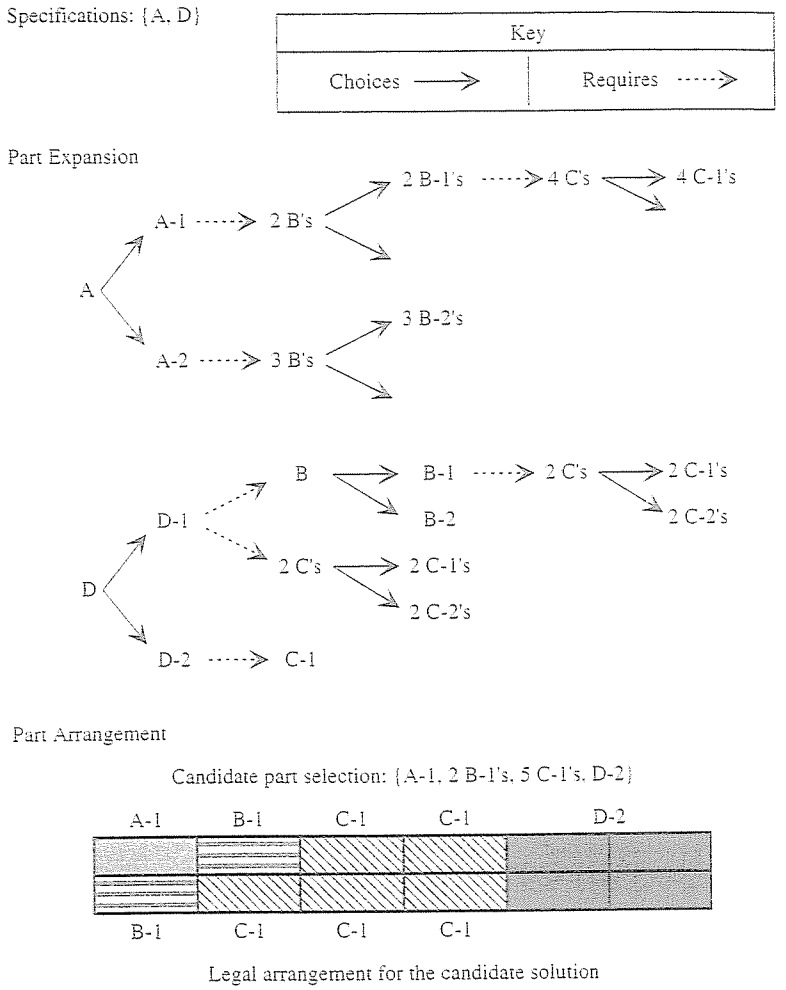

FIGURE 8.6. The dynamic nature of configuration problems. In this example, the initial specifications are {A, D}. The solid arrows indicate where there are alternatives for implementing a part. The dashed arrows indicate places where a part requires further parts. There are several possible solutions for the given specifications. One solution is shown at the bottom. This one fits within a single widget case.

ther requirements will be noted. This dynamic aspect of the problem suggests the use of dynamic and hierarchical constraint methods.

The arrangement model for W-1 lends itself to a sequential layout process, which first places the required A components and then the B components, C components, and D components in left-to-right order. If at any point all of the components of the same type cannot be fitted into the space remaining in a widget case, an extension case is required. If the number of required components of a given type is more than fits into a single case, then the candidate cannot be

arranged according to the rules. Furthermore, when more than one case is required, there must be an available slot in the previous case for its extension plug. At the bottom of Figure 8.6 is one solution to the sample problem, which fits within a single widget case.

Threshold Effects

Configuration problems exhibit **threshold effects**. These effects occur when a small change in a specification causes candidates to exceed certain fixed thresholds, requiring discontinuous and potentially widespread changes to the candidate. For example, in computer configuration domains, overflowing the capacity of a cabinet requires the addition of not only an extension cabinet, but also various components for extending the bus and additional power supplies. In applications where parts have parameters to be determined by configuration, requirements for upgrading the size of components can have ripple effects, so that further parts need to be changed to accommodate changes in weight or size, resulting in shifts to different motor models. Depending on how the decision making is organized in a configuration system, threshold effects can occur in the middle of solving a problem if requirements discovered partway through the process exceed initial estimates of required capacity. In such cases, many early decisions may need to be revised (see Exercise 4).

Figure 8.7 illustrates a threshold being exceeded when the widget specification is changed from {A, D} to {A, 2 D's}. In the two solutions shown, the additional required components exceed the capacity of a single widget case. Furthermore, in the second candidate solution, some of the implementations for a C part were switched to C-2 (rather than C-1) to avoid violating the rule that identical parts must be placed in the same case and to avoid overflowing the extension case. The capacity of a case is a resource. Threshold effects can occur for different kinds of resources. Other arrangement resources include order and adjacency.

Global resources that can be exceeded in different configuration domains include size, power, weight, and cost. These resources present special challenges for allocation and backtracking. They are called **global resources** because they are consumed or required by components throughout a configuration rather than by just one component. In the cases cited, the consumption of the resource is **additive consumption**, meaning that requirements can be characterized in terms of numeric quantities. Requirements from different parts are summed together.

Horizon Effects

When configuration decisions are made incrementally, their evaluation criteria are typically applied in a local context, meaning that they recommend the best solution limiting the evaluation to a small number of factors. This due-process approach is called myopic because it does not try to take into account all of the entailments of a decision caused by the propagation of effects in the model. This leads to a phenomenon called the **horizon effect**. In nautical experience, the curvature of the earth defines a horizon beyond which things are hidden. To see things farther away, you have to get closer to them. In reasoning systems, the "visible horizon" corresponds to things that have been inferred so far. Partway through a configuration process, you cannot see all the consequences of the choices so far because some inferences have not yet been made and because some other choices have yet to be made. At best, you can make estimates from the decisions made so far.

In the idealized case where configuration choices are independent, locally optimal decisions yield globally optimal solutions. For example, if the total cost of a solution is the sum of

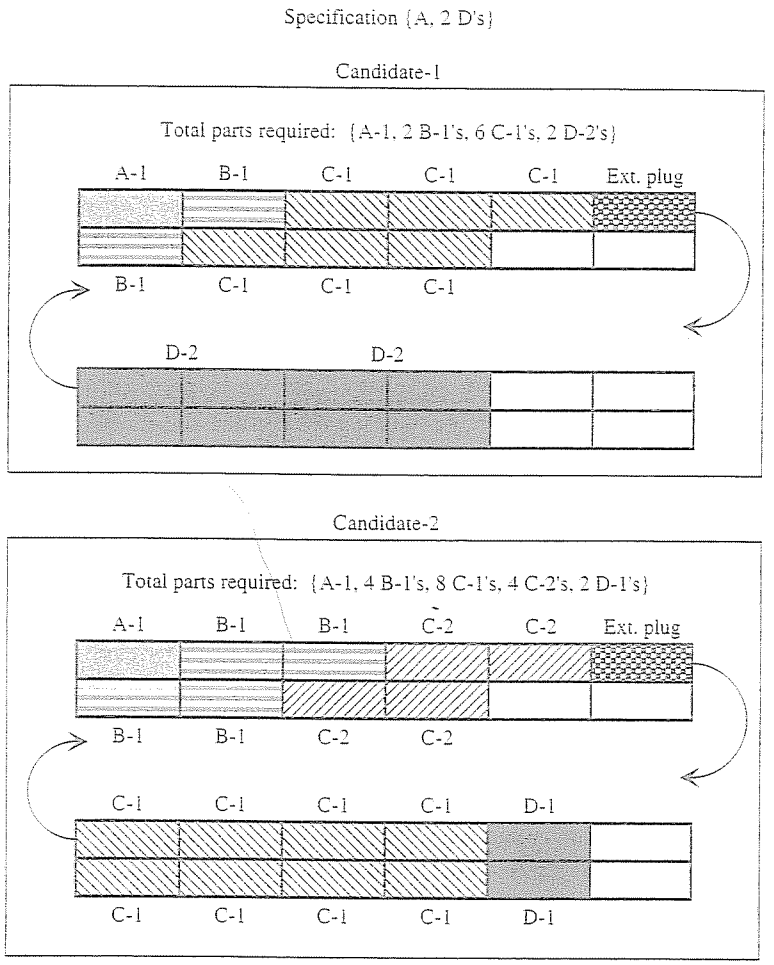

FIGURE 8.7. Threshold effects. This figure illustrates two possible solutions to a widget configuration problem with the specifications {A, 2 D's}. Both candidate solutions exceed the capacity of a single widget case. In more involved examples, exceeding one threshold can lead to a ripple effect of exceeding other thresholds as well, leading to widespread changes in a configuration.

the costs of the independently determined parts, then picking the cheapest part in each case will lead to the cheapest overall solution. If the choice of the next component can be determined by combining the known cost of a partial solution with an estimate of the remaining cost, and the estimate is guaranteed never to be too high, then the A* algorithm can be used as discussed in Section 2.3.

However, in many configuration situations and domains these conditions do not hold and myopic best-first search does not necessarily lead to a globally optimal solution. In each of the following cases, both part A and part B satisfy the local requirements. Part A costs less than part

B and would be chosen by an incremental best-first search that is guided by cost of the part. These cases show why a choice that is locally optimal need not be globally optimal.

Case 1. *Undetected required parts*. The other parts required by part A are much more expensive than the parts required by part B, so the total cost of a complete solution is higher if A is chosen. The further requirements for parts A and B may involve constraints not yet articulated, choices dependent on situations not yet determined, or design variables not yet introduced.

Case 2. *Unanticipated arrangement conflicts*. The parts required by part A consume slightly more of some resource than the parts required by part B. If A is chosen, a threshold will be crossed by later decisions leading to additional expensive cabinets, cases, or whatever. If B had been chosen, the threshold would not have been crossed and the overall solution would have been cheaper (see Exercise 4).

Case 3. *Undetected sharability*. There are requirements not yet considered for a part that is compatible with B but not with A. If B is a sharable component, then choosing B at this stage will make it unnecessary to add parts later on, leading to an overall savings. This phenomenon is especially likely to occur in domains that include multifunctional components, such as multifunctional computer boards that incorporate several independent but commonly required functions (see Exercise 5).

All three cases exhibit the horizon effect. The local evaluation fails to lead to an optimal solution because the indicators fall past the "observable horizon" defined by the commitments and inferences made so far.

If computational complexity was not an issue, the effects of the first case could be mitigated by determining the problematic effects of the required components for a part before committing to a part. Thus, no commitment would be made to choose A until the effects of all of its required components are known. The second case could be mitigated by considering the effects on all resources consumed by a component. The effects in the third case could be mitigated by considering all possible sharings of a component before committing to the selection.

To summarize (with tongue in cheek), the effects of incremental decision making and myopic optimization can be mitigated by performing decision making as far past the immediate horizon as needed. Unfortunately, exponential computational resources are required to look so far ahead. In the following knowledge and symbol-level analyses, we discuss more realistic ways to cope with threshold and horizon effects.

8.2.3 *Summary and Review*

This section models configuration in terms of four elements: a specification language, a model for selecting parts and determining their mutual requirements, a model for arranging parts, and a model for sharing parts across multiple functional requirements. Parts models can be simple catalogs of parts. They can also include hierarchical relations based on functionality, required parts, and bundled packaging. Arrangement models express what physical arrangements of parts are possible and how some arrangements of parts preclude other arrangements. Arrangement models range from complex spatial models to simple named resource models where parts fit in any of a finite number of separately accounted spaces. Sharing models indicate when parts chosen to sat-

isfy one requirement can be used to satisfy others as well. Variations in sharing include exclusive use, open use, metered use, and serial reuse.

Search spaces for configuration domains can be large. Configurations themselves are represented in terms of fairly complex symbol structures that indicate the selection, arrangement, and parameterization of their parts. Most configuration tasks are carried out incrementally, building up descriptions of candidate solutions from descriptions of their parts.

Although configuration domains differ in the complexity of their submodels and the degree to which decisions in each submodel are dependent on decisions in others, they typically exhibit threshold effects where small changes in local requirements can cause discontinuous effects when overall capacities are exceeded. These threshold effects can arise in any of the submodels. Threshold effects and horizon effects for large interconnected problems create a challenge in ordering and managing configuration decisions. The next section presents case studies of configuration systems.

Exercises for Section 8.2

Ex. 1 [*1-05*] *Global Resources.* Decisions about global resources are often problematic in configuration tasks.

(a) Why? What are some examples of global resources from the configuration domains in this chapter?

(b) Briefly describe some alternative approaches for making decisions about global resources.

Ex. 2 [*10*] *Skeletal Configurations.* Professor Digit is a consultant to a company that configures computer systems for its customers. He argues that the number of possible computer configurations may be high but that the number of "really different" configurations sought by customers is actually quite low. He proposes that a knowledge system for configuration would be simpler if it proceeded as follows:

1. Search a library of configurations and find the one closest to the initial configuration proposed for the customer, taking into account which components are key components.
2. Add new key components from the customer's initial configuration that are missing, and remove any extra key components and their required parts from the matching library configuration.
3. Revise the configuration to accommodate the customer's needs.

(a) Briefly describe the main assumptions about the configuration task on which the viability of Professor Digit's proposal depends.

(b) Sam, a graduate student working with Professor Digit, says this proposal creates a tension in the solution criteria for cases in the library between conservative and perhaps inefficient configurations that are easy to extend and ones that are minimal for a fixed purpose. Do you agree? Explain briefly.

(c) Suppose the computer configuration domain is subject to many threshold effects. Briefly, what are the implications for Professor Digit's proposal?

Ex. 3 [*25*] *Configuring Widgets.* The text defines a configuration model, W-1, for widgets. We assume that the following cost figures apply for widget parts.