8.4 Methods for Configuration Problems

In the preceding sections we considered example knowledge systems for configuration tasks and developed a computational model to compare and analyze configuration domains. This section revisits our model of configuration to discuss knowledge-level and symbol-level analysis and then presents sketches of methods for configuration. Our goal in this section is not to develop an "ultimate" configuration approach, but rather to show how variations in the requirements of configuration tasks can be accommodated by changes in knowledge and method.

As in the case of classification tasks, most of the complexity in configuration tasks is in the domain-specific knowledge rather than in the search methods. Furthermore, most of the remaining complexity in the methods is in implementations of general search techniques. In this section the methods are stripped down to basics so we can see the assumptions they depend on in the configuration domain.

8.4.1 Knowledge-Level and Symbol-Level Analysis of Configuration Domains

Knowledge about configuration is used by the submodels of our model. We begin our knowledge-level analysis by considering variations in the knowledge from the domains of our case studies. The submodels define major categories of knowledge for configuration in terms of its content, form, and use.

The Parts Submodel: Knowledge about Function and Structure

The parts submodel contains representations and knowledge about what the available parts are, what specifications they satisfy, and what requirements they have to carry out their functions. The simplest parts submodel is a catalog, which is a predetermined set of fixed parts. However, in most configuration domains the set of parts includes abstractions of them, organized in an abstract component hierarchy also called a functional hierarchy. Such hierarchies are used in mapping from initial specifications to partial configurations and in mapping from partial configurations to additional required parts. In most configuration domains these hierarchies represent functional groupings, where branches in the hierarchy correspond to specializations of function. In the last section we saw several examples of functional hierarchies in the computer configuration applications for XCON, M1, and COSSACK.

At the knowledge level an abstraction hierarchy guides problem solving, usually from the abstract to the specific. At the symbol level a parts hierarachy can be implemented as an index into the database of parts. There are several other relations on parts that are useful for indexing the database.

The determination of required parts is a major inference cycle in configuration. This process expands the set of selected parts and consumes global resources. The **required-parts relation** indicates what additional parts are required to enable a component to perform its role in a configuration. For example, a disk drive requires a disk controller and a power supply. These parts, in turn, may require others. These relations correspond to further requirements, not to specializations of function. In contrast, power supplies are required parts for many components with widely varying functions. Power supplies perform the same function without regard to the function of the components they serve.

We say that parts are bundled when they are necessarily selected together as a group. Often parts are bundled because they are manufactured as a unit. Sometimes parts are manufactured together because they are required together to support a common function. For example, a set of computer clocks may be made more economically by sharing parts. In other cases parts are made together for marketing reasons. Similarly, bundling components may reduce requirements for some resource. For example, communication and printing interfaces may be manufactured on a single board to save slots.

In some domains, parts are bundled because they are logically or stylistically used together. In configuring kitchen cabinets, the style of knobs and hinges across all cabinets is usually determined by a single choice. Parts can also be bundled for reasons not related to function or structure. For example, a marketing group may dictate a sales policy that certain parts are always sold together.

Some part submodels distinguish special categories of "spanning" or "dummy" parts. Examples of spanning parts from the domain of kitchen cabinets are the space fillers used in places where modular cabinets do not exactly fit the dimensions of a room. Examples of filler parts from computer systems are the conductor boards and bus terminators that are needed to compensate electrical loads when some common component is not used. Examples from configuring automobiles include the cover plates to fill dashboard holes where optional instruments were not ordered. Dummy and filler parts are often added at the end of a configuration process to satisfy modest integrity requirements.

Another variation is to admit parameterized parts, which are parts whose features are determined by the given values of parameters. A modular kitchen cabinet is an example of a parameterized part. Parameters for the cabinet could include the choice of kind of wood (cherry, maple, oak, or alder), the choice of finish (natural, red, golden, or laminate), and the choice of dimensions (any multiple of 3 inches from 9 inches to 27 inches).

Finally, parameterization can be used to describe properties of parts and subsystems, as in the VT system. Global properties of a configuration such as power requirements, weight, and cost are usually treated as parameters because they depend on wide-ranging decisions. In the VT system, essentially all the selectable features of the elevator system and many intermediate properties are represented in terms of parameters.

The Arrangement Submodel: Knowledge about Structure and Placement

The arrangement submodel expresses connectivity and spatial requirements. There are many variations in the requirements for arrangement submodels. Some configuration domains do not require complex spatial reasoning. For example, VT's elevator configurations all use minor variations of a single template for vertical transport systems. M1 uses a port-and-connector model for logical connections but its configurations are spatially uncommitted. Spatial arrangement of antibiotics is not relevant for MYCIN therapy recommendations.

Arrangements do not always require distance metrics or spatial models. In configuring the simplest personal computers, the slots that are used for different optional parts are referenced and accounted for by discrete names or indexes. A configuration task need only keep track of which slots are used without bothering about where they are.

Figure 8.34 illustrates several examples of specialized arrangement models. The simplest of these is the port-and-connector model. This model was used in the M1 and COSSACK appli-

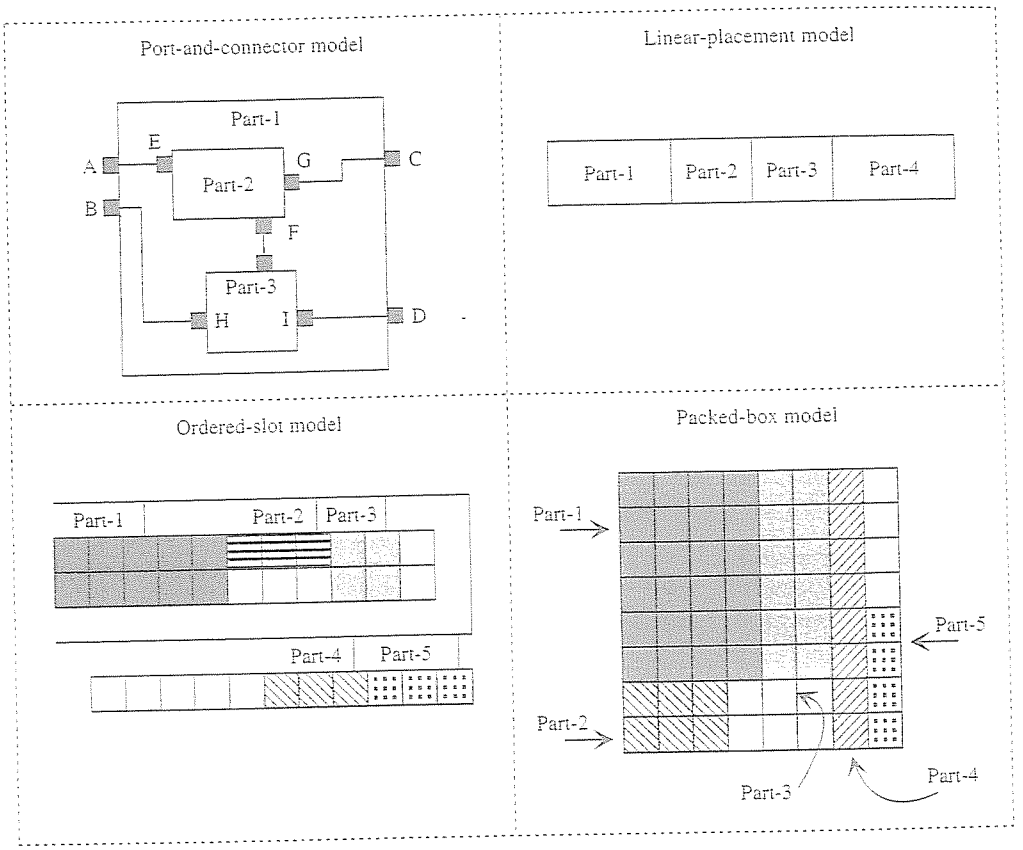

FIGURE 8.34. Alternative models of arrangement for configuration problems. Different arrangement models lend themselves to different resource allocation strategies.

cations discussed earlier. In this model, ports are defined for all the configurable parts. Ports have types so that different kinds of ports have different properties. Ports are resources than can be assigned to at most one connector. COSSACK implemented a port-and-connector model in a frame language, combining its use for specifying the arrangement of parts with indexing on constraints that specified what parts were compatible or required. The linear-placement model organizes parts in a sequence. A specialization of this model was used in XCON for specifying the order of electrical connections on a computer bus. In that application, roughly speaking, it is desirable to locate parts with a high interrupt priority or a high data rate nearer the front of the bus. Location on the bus was a resource that was consumed as parts were arranged. The ordered-slot model combines a linear ordering with a spatial requirement. In the version in Figure 8.34, parts are ordered along a bus, starting with Part-1 and ending with Part-4. Parts in the first row can be up to two units deep, and parts in the second row can be only one unit deep. In this example, both space and sequence are resources that get consumed as parts are arranged. The packed-box model emphasizes the efficient packing of parts. In this model, space is a consumable

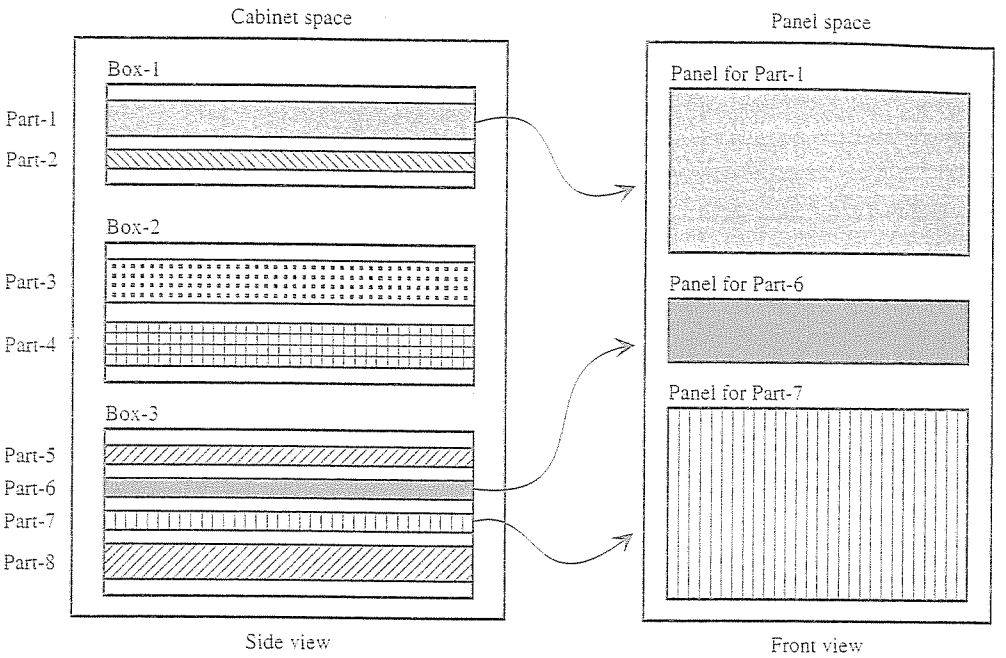

FIGURE 8.35. Another variation on models of arrangement for configuration problems. In this model, parts are arranged in boxes and boxes are arranged in cabinets. In addition, some parts require control panels and there is a limited amount of panel space. Panels for parts must be located on the cabinet that contains the parts. In this model, space in boxes, space in cabinets, and space for panels are all resources consumed during configuration. Because of the connections between parts and panels, the resource allocation processes are interdependent.

resource. Constraints on packing may also include requirements about certain parts being adjacent to one another.

All the variations in Figure 8.34 represent physical containment. **Physical-containment relations** form a hierarchy whose branches indicate which parts are physically contained in others. For example, a cabinet may contain a circuit board, which in turn may contain a processor and some memory. Contained parts do not necessarily carry out specializations of their container's function, nor are they necessarily required parts for their container.

COSSACK uses a port-and-connector model to account for the use of slots. It also keeps track of memory resources. XCON's arrangement models are the most complex in our example systems. Figure 8.35 illustrates another complication in arrangement models. In this example, there are three interlinked tasks in arrangement: the location of boards inside boxes, the location of boxes inside cabinets, and the location of panels on the front of cabinets. The tasks are interlinked because the control panels for particular boards need to be located on the cabinet containing the corresponding boards.

XCON also admits complex arrangement conditions and interactions with nonspatial resources. For example, a rule for allocating slots inside boxes says: "a box can accommodate

five 4-slot backplanes; the exception is that a 9-slot backplane may not occupy the space reserved for the second and third 4-slot backplanes." This example combines electrical requirements with space allocation.

Knowledge about Decision Ordering and Interactions

The difficulty of configuration problems arises from a three-horned dilemma, where the three horns are combinatorics, horizon effects, and threshold effects. The combinatoric difficulty is that the number of possible configurations increases multiplicatively with the number of parts and arrangements. In the configuration domains discussed in this chapter, there are far too many possible configurations to instantiate them all in complete detail. Most decisions need to be made on the basis of partial information about the possible configurations. This is where the horizon effect comes in. At each stage of reasoning, some interactions are not yet visible either because not enough inferences have been made or because not enough commitments have been made. When information is partial, a natural approach is to rely on estimates. This is where the threshold effect comes in. A small difference in specifications or descriptions can lead to large, widespread changes. Because of their approximate nature, estimation functions can be off by small amounts. If these small errors are near critical thresholds, then the estimates may not be trustworthy.

To reiterate the argument in short form, there are too many configurations to develop them in detail. Making choices on the basis of partial information is subject to horizon effects. Estimates at the horizon based on partical information are subject to threshold effects.

The approaches that we consider for coping with these effects assume that there are predictable or typical patterns of interaction and that knowledge about these patterns is available. The following discussion is concerned with the dimensions of these patterns.

The first horn of the dilemma, combinatorics, is dealt with by two major techniques: abstraction and decision ordering. Thus, the use of functional abstractions in reasoning about configuration amounts to hierarchical search and has the usual beneficial effects. Controlling the order of decisions can reduce the amount of backtracking in search. In the context of configuration, knowledge-level justifications of decision ordering complement the symbol-level graph analyses. In XCON and VT, the reduction of backtracking is characterized in terms of dependent and independent decisions, where each decision is carried out only after completing other decisions on which it depends.

For example, XCON defers cabling decisions to its last subtask. Three assumptions rationalize this late placement of cabling decisions. The first assumption is that for any configuration, cabling is always possible between two components. This assumption ensures that the configuration process will not fail catastrophically due to an inability to run a cable. The second assumption is that all cables work satisfactorily, for any configuration. Thus, a computer system will work correctly no matter how long the cables are. The third assumption is that cable costs do not matter. Cable costs are so low when compared with active components that they will not influence an overall cost significantly. As long as these assumptions hold, XCON would never need to backtrack (even if it could!) from a cable decision and there is no need to consider cable issues any earlier in XCON's process. If there were some special cases where cables beyond a certain length were unavailable or would prevent system operations, a fix in terms of other component choices or arrangements would need to be determined and XCON would be augmented with look-ahead rules to anticipate that case.

Much of the point of the SALT analysis in the elevator domain is to identify patterns of interactions. VT's data-driven approach is a least-commitment approach and is analogous to constraint satisfaction methods that reason about which variable to assign values to next. Least commitment precludes guessing on some decisions when refinements on other decisions can be made without new commitments. Deadlock loops correspond to situations where no decision would be made. To handle cases where decisions are inherently intertwined so there is no ordering based on linear dependencies, SALT introduces reasoning loops in the parameter network that search through local preferences toward a global optimum.

The partial-commitment approach extends least commitment by abstracting what is common to the set of remaining alternatives and then using that abstraction to constrain other decisions. VT narrows values on some parameters, sometimes on independent grounds, so that components are partially specified at various stages of the configuration process.

In VT, such reasoning can amount to partial commitment to the extent that further inferences are drawn as the descriptions are narrowed.

All efforts to cope with the combinatorics of configuration encounter horizon effects. Since uniform farsightedness is too expensive, an alternative is to have limited but very focused farsightedness. Domain knowledge is used to anticipate and identify certain key interactions, which require a deeper analysis than is usual. This approach is used in XCON, when it employs its look-ahead rules to anticipate possible future decisions. The required conditions for look-ahead can be very complex. In XCON, heuristics are available for information gathering that can tell approximately whether particular commitments are feasible. The heuristics make look-ahead less expensive.

Finally, we come to techniques for coping with the threshold effect. The issue is not just that there are thresholds, but rather that the factors contributing to the threshold variables are widely distributed. Sometimes this distribution can be controlled by bringing together all the decisions that contribute. This helps to avoid later surprises. Knowledge about thresholds and contributing decisions can be used in guiding the ordering of decisions. In general, however, there is no single order that clusters all related decisions. One approach is to anticipate thresholds with estimators and to make conservative judgments. For example, one could anticipate the need for a larger cabinet. This conservative approach works against optimization. In XCON, many of the constraints whose satisfaction cannot be anticipated are soft. This means that although there may be violations, the softness of the constraints reduces the bad consequences. Thus, the extra part may require adding a cabinet, but at least it is possible to add new cabinets.

In summary, domain knowledge is the key to coping with the complexity of configuration. Knowledge about abstractions makes it possible to search hierarchically. Knowledge about key interactions makes it possible to focus the computations for look-ahead to specific points beyond the horizon. Knowledge about soft constraints reduces the effects of threshold violations.

8.4.2 *MCF-1: Expand and Arrange*

In this section and the following, we consider methods for configuration. In principle, we could begin with a method based on a simple generate-and-test approach. By now, however, it should be clear how to write such a method and also why it would be of little practical value. We begin with a slightly more complex method.

Figure 8.36 presents an extremely simple method for configuration, called MCF-1. MCF-1 acquires its specifications in terms of key components, expands the configuration to include all required parts, and then arranges the parts. Although this method is very simple it is also appropriate for some domains. MCF-1 is M1's method, albeit in skeletal form and leaving out the secondary cycle for reliability analysis and revision. It is practical for those applications where the selection decisions do not depend on the arrangement decisions.

MCF-1 has independent auxiliary methods get-requirements, get-best-parts, and arrange-parts. The auxiliary method get-requirements returns all the requirements for a part and returns the initial set of requirements given the token initial-specifications. Given a set of requirements, there may be different alternative candidate sets of parts for meeting a requirement. The method get-best-parts considers the candidate sets of new parts and employs a domain-specific evaluation function to select the best ones. Finally, given a list of parts, the method arrange-parts determines their best arrangement. A recursive subroutine, add-required-parts, adds all the parts required by a component.

MCF-1 relies on several properties of its configuration domain as follows:

- Specifications can be expressed in terms of a list of key components.
- There is a domain-specific evaluation function for choosing the best parts to meet requirements.
- Satisfactory part arrangements are always possible, given a set of components.
- Parts are not shared. Each requirement is filled using unique parts.

The first assumption is quite common in configuration tasks and has little bearing on the method. The second assumption says there is a way to select required parts without backtracking. An evaluation function is called from inside get-best-parts. The third assumption says that arrangement concerns for configurations involve no unsatisfiable constraints, even in combination. This is not unusual for configuration tasks that use a port-and-connector model for arrangements. In such domains, one can always find some arrangement and the arrangement does not affect the quality of the configuration. This assumption is crucial to the organization of MCF-1 because it makes it possible to first select all the required parts before arranging any of them. It implies further that the arrangement process never requires the addition of new parts. A somewhat weaker assumption is actually strong enough for a simple variation of the method: that part arrangement never introduces any significant parts that would force backtracking on selection or arrangement decisions. The last assumption means the system need not check whether an existing part can be reused for a second function. This method would need to be modified to accommodate multifunctional parts.

8.4.3 MCF-2: Staged Subtasks with Look-Ahead

Method MCF-1 is explicit about the order that configuration knowledge is used: first, key components are identified; then required parts are selected; and finally, parts are arranged. As discussed in our analysis of configuration, these decisions may need to be more tightly interwoven. Sometimes an analysis of a domain will reveal that certain groups of configuration decisions are tightly coupled and that some groups of decisions can be performed before others. When this is the case, the configuration process can be organized in terms of subtasks, where each subtask

To perform configuration using MCF-1:

```
        /* Initialize and determine the key components from the
        specifications. */
 1.     Set parts-list to nil.

        /* Get-requirements returns specifications for required parts given
        a part. */
 2.     Set requirements to get-requirements(initial-specifications).
 3.     Set key-components to get-best-parts(requirements).

        /* Add all the required parts to the parts-list. */
 4.     For each key component do
 5.     begin
 6.           Push the key component onto the parts-list.
 7.           Add-required-parts(key-component).
 8.     end

        /* Arrange parts in the configuration. */
 9.     Arrange parts using arrange-parts(parts-list).
 10.    Return the solution.


        /* Recursive subroutine to add parts required by a given
        part. When there are multiple candidates it selects the best one
        get-best-parts. */
 1.     Add-required-parts(part)
 2.     begin
 3.           Set requirements to get-requirements(part).
 4.           If there are some requirements, then
 5.           begin
 6.                 Set new-parts to get-best-parts(requirements).
 7.                 For each of the new-parts do
 8.                 begin
 9.                       Push the new-part onto the parts-list.
 10.                      Add-required-parts(new-part).
 11.                end
 12.          end
 13.    end
```

FIGURE 8.36. MCF-1, a simple method for configuration problems in which there is no interaction between selection and arrangement of required parts.

performs a combination of closely related decisions to refine, select, and arrange components. Subtasks cluster decisions so that most interactions among decisions within a subtask are much greater than interactions with decisions outside of it.

In some domains the partitioning of decisions into subtask clusters does not necessarily yield a decision ordering in which all later decisions depend only on early ones. To restate this in constraint satisfaction terms, there may be no ordering of variables such that values can be assigned to all variables in order without having to backtrack. To address this, knowledge for selective look-ahead is used to anticipate interactions. Such knowledge is domain specific and may involve conservative estimates or approximations that anticipate possible threshold effects. This approach can be satisfactory even in cases where the look-ahead is not foolproof, if the failures involve soft constraints.

This approach is followed in MCF-2. The method description given in Figure 8.37 says very little about configuration in general. Nonetheless, MCF-2 is the method used by XCON, albeit with its domain-specific particulars abstracted.

```
To perform configuration using MCF-2:
        /* Initialize and get the key components from the specifications. */
1.      Set parts-list to nil.
2.      Set requirements to get-requirements(initial-specifications).
3.      Set key-components to get-best-parts(requirements).

        /* Conditionally invoke Subtasks. */
4.      While there are pending subtasks do
5.      begin
6.              Test conditionals for invoking subtasks and choose the best
                one.
7.              Invoke a specialized method for the subtask.
8.      end
9.      Return the solution.

        /* Each specialized method performs a subset of the task. It is
        responsible for refining some specifications, adding some parts, and
        arranging some parts. It incorporates whatever look-ahead is needed.
        */
10.     Method-for-Subtask-1:

        /* Subtask-1: Look ahead as needed and expand some parts. */
        /* Subtask-2: Look ahead as needed and arrange some parts. */


20.     Return.
...
30.     Method-for-Stage-3:
...
```

FIGURE 8.37. MCF-2, a method for configuration problems in which part selection and arrangement decisions can be organized in a fixed sequence of subtasks. The method depends crucially on the properties of the domain. This method tells us very little about configuration in general.

This method is vague. We might caricature it as advising us to "write a configuration system as a smart program using modularity as needed" or as "an agenda interpreter that applies subtasks in a best-first order." Such symbol-level characterizations miss the point that the decision structure follows from an analysis of relations at the knowledge level. This observation is similar to the case of methods for classification. The particulars of the domain knowledge make all the difference. The burden in using MCF-2 is in analyzing a particular domain, identifying common patterns of interaction, and partitioning the decisions into subtasks that select and arrange parts while employing suitable look-ahead.

It is interesting to compare MCF-2 to the general purpose constraint satisfaction methods. One important result is that when the variables of a constraint satisfaction problem (CSP) are block ordered, it is possible to find a solution to the CSP with backtracking limited to the size of the largest block, which is depth-limited backtracking. Although there are important differences between the discrete CSP problems and the configuration problems, the basic idea is quite similar. Subtasks correspond roughly to blocks. When subtasks are solved in the right order, little or no backtracking is needed. (See the exercises for a more complete discussion of this analogy.)

As in MCF-1, MCF-2 does not specify an arrangement submodel, a part submodel, or a sharing submodel. Each domain needs its specially tailored submodels. In summary, MCF-2 employs specialized methods for the subtasks to do special-case look-ahead, to compensate for known cases where the fixed order of decision rules fails to anticipate some crucial interaction leading to a failure. The hard work in using this method is in the analysis of the domain to partition it into appropriate subtasks.

8.4.4 MCF-3: Propose-and-Revise

In MCF-2, there are no provisions in the bookkeeping for carrying alternative solutions, or for backtracking when previously unnoticed conflicts become evident. Figure 8.38 gives MCF-3, which is based loosely on the mechanisms used in VT, using the propose-and-revise approach. Like MCF-2, we can view MCF-3 as an outline for an interpreter of knowledge about configuration. It is loosely based on the ideas of VT and the interpretation of a parameter network. It depends crucially on the structure of the partial configuration and the arrangement of knowledge for proposing values, proposing constraints, noticing constraint violations, and making fixes.

The beauty of MCF-3 is that the system makes whatever configuration decisions it can by following the opportunities noticed by its data-driven interpreter. In comparison with MCF-2, MCF-3 is does not rely so much on heuristic look-ahead, but has provisions for noticing conflicts, backtracking, and revising.

8.4.5 Summary and Review

This section sketched several different methods for configuration, based loosely on the example applications we considered earlier. We started with a method that performed all selection decisions before any arrangement decisions. Although this is practical for some configuration tasks, it is not adequate when arrangements are difficult or are an important determinant of costs. The second method relied on look-ahead. Our third method triggered subtasks according to knowl-

```
To perform configuration using MCF-3:
     /* Initialize and obtain requirements. */
1.   Initialize the list of parts to empty.
2.   Set requirements to get-requirements(initial-specifications).
3.   Set key-components to get-best-parts(requirements).

     /* Apply the domain knowledge to extend and revise the configuration.
     */
4.   While there are open decisions and failure has not been signaled do
5.   begin
6.        Select the next-node in the partial configuration for which a
          decision can be made.
7.        If the decision is a design extension, then invoke method
          propose-design-extensions(next-node).
8.        If the decision is to post a constraint, then invoke method
          post-design-constraints().
9.        If there are violated-constraints, then invoke revise-design().
10.  end
11.  Report the solutions (or failure).
```

FIGURE 8.38. MCF-3, a method based on the propose-and-revise model.

edge about when the subtasks were ready to be applied and used constraints to test for violations when choices mattered for more than one subtask.

Exercises for Section 8.4

Ex. 1 [*10*] *MYCIN's Method.* How does MYCIN's method for therapy recommendation fit into the set of methods discussed in this section? Is it the same as one of them? Explain briefly.

■ Ex. 2 [*R*] *Subtask Ordering.* After studying the XCON system and reading about constraint satisfaction methods, Professor Digit called his students together. "Eureka!" he said. "MCF-2 is really a special case of an approach to constraint satisfaction that we already know: block ordering. In the future, pay no attention to MCF-2. Just make a tree of 'constraint blocks' corresponding to the configuration subtasks and solve the tasks in a depth-first order of the tree of blocks."

(a) Briefly sketch the important similarities and differences between CSP problems and configuration problems.

(b) Briefly, is there any merit to Professor Digit's proposal and his suggestion that there is a relation between traversing a tree of blocks in a CSP problem and solving a set of staged subtasks in a configuration problem?

(c) Briefly relate the maximum depth of backtracking necessary in a CSP to the "no backtracking" goal of systems like XCON.

Ex. 3 [*10*] *Methods and Special Cases.* The methods for configuration are all simple, in that the complexity of configuration tasks is manifest in the domain-specific knowledge rather than