Computers in Industry 21 (1993) 1–10
Elsevier

Knowledge Engineering

Configuration of complex products

M.J. Euwe [a] and R.V. Schuwer [b]

[a] *Anderson Consulting, Eindhoven, Netherlands*
[b] *Faculty of Industrial Engineering and Management Science, Eindhoven University of Technology, Eindhoven, Netherlands*

Received April 16, 1992; accepted June 26, 1992

Making a configuration of a complex product in the course of a sales process is a difficult matter. A sales-support information system (SSIS) can be a useful help for salesmen. An SSIS should give support in two problem areas. The first area concerns the problem of getting a *valid* product specification, i.e. a specification of a product which falls within the product assortment. The second area concerns the problem of getting the product that is the best solution for the problem of the customer. Such an SSIS must be able to communicate in three different languages. These languages are called user-language, functional language and technical language. In this paper the problem of specifying a product in technical language is treated. With the building of a prototype of such an SSIS it is demonstrated that it is possible to give a salesman sufficient support for the problem of getting a valid product specification. The proposed solution for the second problem area has not been processed into the prototype.

Keywords: Configuration; Configuration system; Sales process; Knowledge-base system

1. Introduction and problem statement

A great number of industrial products can be assembled out of a number of components, where one out of various options needs to be selected for each component. The end product is so to speak the "sum" of several components. Specification of the product is mainly the specification of each of the components. Examples of such products are computer systems, medical test systems, kitchens, trucks etc. When buying such a product, a customer can select the options which give him a product that best fulfils his specific demands.

In the remainder of this paper the following terminology will be used. The components out of which a product is assembled are called *features*. For each feature one or more choices can be made. Such a choice for a feature is called an *option. A variant* (or product variant) is a combination of selected options which is allowed and which forms a complete product. In general, not all combinations of options are allowed in a product. The rules which denote which combinations of options are obliged and which combinations are forbidden are called *restrictions*. Three different kinds of restrictions can be distinguished: An *exclusion* denotes a forbidden combination of two or more options within a product specification. An *inclusion* denotes an obliged combination of two sets of options. That is, when one set of options is chosen, the other set must be chosen as well (but not necessarily the other way round). A *coupling* denotes two or more sets of options which must always be chosen together (A coupling can easily be denoted by several inclusions. For reasons of overview and compactness this kind of restriction is introduced.). In spite of these restrictions the number of possible variants of a product can often be large in the case of industrial products. A *generic product specification* is the combination of the set of features and options and its corresponding restrictions describing a whole range of possible products [1]. Figure

Correspondence to: Ir. drs. R.V. Schuwer, Eindhoven University of Technology, TUE/TBDK/I&T, Den Dolech 2, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

0166-3615/93/$06.00 © 1993 – Elsevier Science Publishers B.V. All rights reserved

FEATURE	OPTION
CPU	8086 80286 80386
Co-processor	8087 80287 80387
Memory	640K 2M 4M 8M

RESTRICTIONS	
Exclusion:	IF CPU(80286) THEN NOT Memory(8M)
Inclusion:	IF Co-processor(8087) THEN CPU(8086)
Coupling:	CPU(8086) IF AND ONLY IF Memory(640K)
......	

IBM-compatible PC

Fig. 1. Example of a generic product specification.

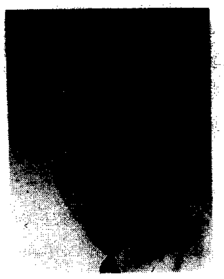
1 gives an example of a generic product specification for a PC.

1.1. Problem area

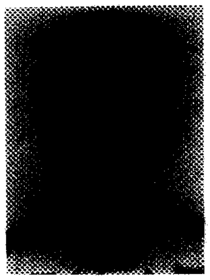
Because of the enormous number of variants it may be very difficult during the selling process to determine the variant which is most suited for the customer. For all types of products it is known out of which options for which features a choice has to be made. The decision not to choose a certain feature can be one of the options. As soon as options have been selected for all features the product is completely specified. In the process of selecting options two difficulties can be distinguished:

(i) Because of the restrictions not all combinations of options are allowed. This is the problem of getting a *valid* product specification. A valid specification is a specification, which meets the restrictions with respect to the possibilities of combining options. Because of the great number of possible combinations of options it is very difficult for a salesman to combine those options correctly. Therefore he might end up with an order for a product which cannot be assembled, due to some overlooked restrictions.

(ii) During the selection of options the re-



Mark Euwe holds a Bachelors degree in Computer Science from Venlo Institute for Technology and a Masters degree in Industrial Engineering and Management Science from Eindhoven University of Technology. He is currently working as a management consultant for Andersen Consulting (Eindhoven, The Netherlands) and specializes in the fields of production control and logistics.



Robert Schuwer received his Masters degree in Mathematics at the Catholic University of Nijmegen and his MSc in Computing Science at the Eindhoven University of Technology. He is now working as an assistant professor at the Faculty of Industrial Engineering and Management Science at the Eindhoven University of Technology. His current research interests are applications of knowledge-base systems and the added value that can be gained in using them.

quirements of the customer must be taken into account. The relations between those requirements and the options can be very complex. Most of the time the customer will formulate his requirements in terms of characteristics of use. At the end the variant must be specified in characteristics of the product, i.e. in features and options. The salesman must translate between those two different sets of characteristics. In this process he will rely on his experience. This gives rise to the possibility that, because of a lack of knowledge of the salesman, the customer will buy a product which does not fully conform to his requirements. This can lead to dissatisfaction with the customer so that in the future the customer will go to a competitor. It can also lead to higher costs of guarantee, because the product may be used for applications for which it is not completely suited.

A *sales-support information system* (SSIS) should support the salesman and the customer during the process of selecting options in both problem areas. In this paper a research of the feasibility of such an SSIS is described. First a sketch of an SSIS is given. From this sketch an implementation of a prototype SSIS has been made to validate the feasibility of such an SSIS. At the end of this paper the experiences with this prototype are described.

2. Reflection of a sales dialogue

The customer considers buying a product as searching for a solution for a problem within his organization. The goal of a sales dialogue is to offer the customer the right solution. What is "right" depends on the problem of the customer. The possible solutions are described in terms of features and options. The problem of the customer is not formally described, but exists only in the mind of the customer. Therefore there has to be an exchange of information between the customer and the salesman to determine the right solution. A closer look will learn, that there are three different levels on which a customer may specify his requirements:

(i) The customer specifies his problem by stating the *use* (the why and the how) of the product. The language in which this specification is made is defined *user-language*. The salesman translates parts of the problem into certain parts of the solution (i.e. specific options). In the case of selling a kitchen the customer can state that the family exists of two persons who both have a full-time job. The salesman translates this piece of information into the option "microwave combination oven" of the feature "oven".

(ii) The customer knows which *functional requirements* result from his problem and states those requirements. The language in which functional requirements are expressed is defined *functional language*. The salesman translates those requirements into a part of the solution (i.e. specific options). In the case of selling a kitchen the customer realizes that, with both persons having a full-time job, there is a need to quickly warm up meals. The customer tells the salesman that the oven must be able to warm up meals quickly. The salesman translates this piece of information into the option "microwave combination oven" of the feature "oven".

(iii) The customer knows the relations between the problem and the solution and states the specific options. This language is defined *technical language*. The salesman chooses those options. In the kitchen-case the customer can realize that his situation demands for quickly warmed up meals and that a microwave combination oven is best suited to do this. He tells the salesman that the kitchen must have such an oven. The salesman chooses the option "microwave combination oven" of the feature "oven".

The three different languages can be considered as three levels of knowledge of the customer. The more knowledge the customer has of the product, the more he will state his requirements in terms of technical language and the less the salesman has to translate. As a rule a combination of the three languages is used. The reason for this is that not all features have the same technical complexity. For instance, the feature "dishwasher" is part of the technical language in specifying a kitchen. Most customers, however, will state this requirement as-is. This however is not the case for the capacity of the extractor in relation to the amount of air which must be refreshed per minute. For this reason it is not possible to let the customer just point out the required options.

During the process of selling a product, communication is possible at different levels of language. An SSIS must be able to handle this. This is especially important because of the fact that it might annoy the user to have a dialogue at a too low level of knowledge.

The rest of this paper is concerned with specifying a product using the technical language. The next chapter will describe the problems of specification using technical language.

3. Problems of specifying in technical language

The ultimate goal of the specification process is to obtain a valid product specification. The implementation of this process can occur in two different ways [2]:

(i) The user specifies a product and the complete specification is checked on validity. When it turns out that the specification is not valid, then the user must adapt the specification. This new specification must be checked again on validity. Eventually this will yield a valid specification.

(ii) The user specifies the product in an interactive way. The system guarantees the validity of the complete specification. To put it another way: the system guides the process of specification in such a way that in the end it is impossible to have a non-valid specification.

We will elaborate the second way of specifying. This way of specifying a product is much more efficient than the first one. It would not be realistic to make a complete specification only to discover that somewhere during the process a mistake has been made.

We will first introduce some terminology to describe the process of specification. The *degrees of freedom* of a feature X are the options which are (still) allowed to choose for the feature X. A feature with only one degree of freedom is called a *definite* feature. A feature with more than one degree of freedom is called an *indefinite* feature. A *selection step* consists of making an arbitrary indefinite feature definite. The *specification process* consists of a sequence of selection steps which transforms a situation in which all features are indefinite into a situation in which all features are definite.

After each selection step the number of indefinite features decreases with at least one and therefore the number of definite features is increased with the same number. It is possible that with one selection step more than one features are determined due to restrictions between options. Consider the example in Fig. 2.

In the figure a square represents a degree of freedom for a feature. The complete specification

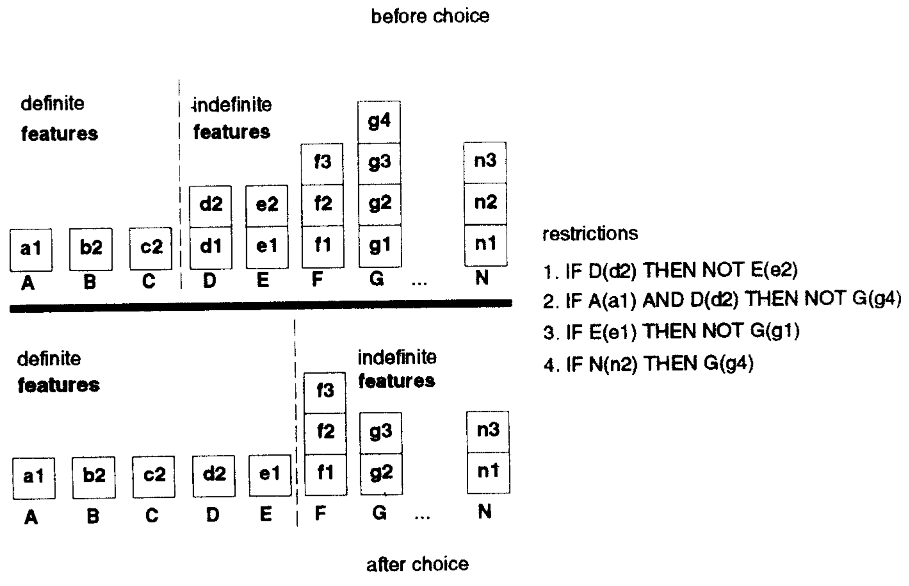


Fig. 2. Selection step in the specification process.

consists of N features. Features A, B, and C are already definite. The user wants to make feature D definite and chooses option d2. This choice results in feature E to be implicitly definite too (because of restriction 1). Another result of the choice of d2 is the change in the number of degrees of freedoms for other indefinite features. In the example this applies to features G and N. Consider in particular the repercussions of restrictions 2 and 4. Because of the choice of d2 restriction 2 is applied. This results in the exclusion of option g4. Restriction 4 is equivalent to "IF NOT G(g4) THEN NOT N(n2)". Because of this chain reaction eventually option n2 is also excluded.

During the specification process two complications can occur:
(i) the problem of a total exclusion;
(ii) the problem of an excluded option which is required by the customer.
We will go deeper into these two problems.

3.1. The problem of a total exclusion

It must be guaranteed that the user will never get stuck during the process. This is the situation where the user has to modify choices that were made in previous selection steps, because otherwise it is not possible to obtain a valid product in the end. An example of this situation is illustrated in Fig. 3.

In this example it is assumed that feature A has the options a1 and a2; feature B has the options b1, b2 and b3; feature C has the options c1 and c2. Due to restriction 1 option c1 is not allowed anymore because of the choice of a1. Due to restriction 2 option c2 is not allowed anymore because of the choice of b2. Feature C now has zero degrees of freedom. At least one of the choices for feature A or B must be changed.

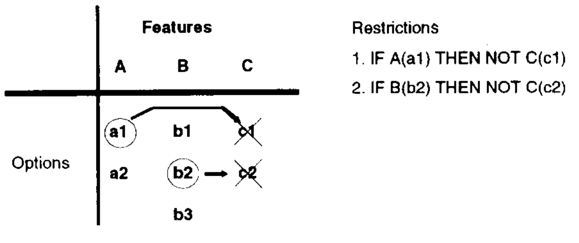


Fig. 3. A total exclusion.

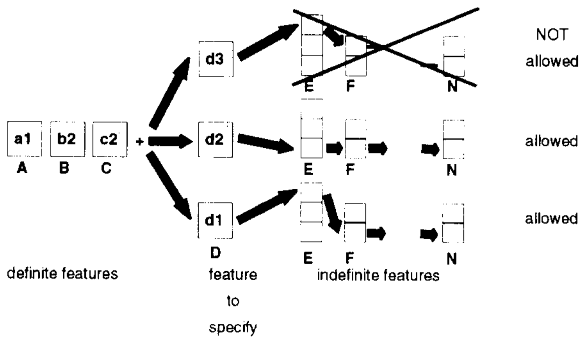


Fig. 4. Determination of allowed options.

These undesirable situations are called *total exclusions* and can be characterised by the existence of one or more features with zero degrees of freedom. To prevent this situation the user should only be allowed to choose for options bl and b3 for feature B when he has chosen a1.

The algorithm that guides the specification process should rule out situations in which a feature has zero degrees of freedom. This can be transformed to the demand that the algorithm should only allow the choice of those options that will not lead to a total exclusion. The system should therefore distinguish between allowed options (not leading to a total exclusion) and non-allowed options (leading to a total exclusion). An algorithm to decide which options are allowed is the following:

Let X be the feature to specify in the next selection step. Let $x_1, \ldots, x_n$ be the options for feature X. Let O denote the set of options that are already chosen. For each option x_i let O_i be the union of O and the set $\{x_i\}$. Decide if there exists a valid specification, starting with the set O_i (for each i). If such a specification exists, then x_i is allowed, else x_i is not allowed.

In plain English: if a variant exists that contains the current specification and option x_i then option x_i is allowed.

Figure 4 illustrates this algorithm. In this figure, the product consists of the features A up to and including N. Features A, B, and C are already determined. The user wants to specify feature D. It must be determined whether d1, d2 and d3 are allowed. With options d1 and d2 a valid specification can be made with certain choices for features E to N. This has been symbolical denoted with the arrows. Therefore, options d1 and d2 are allowed. With option d3 such a valid specification cannot be found, so d3 is not allowed and will not be shown to the user.

In the remainder of this paper, the set O of already determined options will be called the *current specification*.

3.2. The problem of an excluded option which is required by the customer

In Fig. 2, one can see that a choice for feature D has consequences for the degrees of freedom for features G and N. In general, the user does not know the effect of his choice on other features. For instance, it is possible that the user, when specifying feature N, notices that the preferred option has already been excluded due to the choice at feature D. The user may find that his choice at feature D is more important and therefore just accept the situation. However, if the user finds the excluded option at feature N more important than the choice at feature D, then he has to modify his previously made choice.

The necessity of altering a previously made choice can cause a chain reaction of changes in the specification. Therefore, it is necessary to avoid those situations as much as possible. This problem could be tackled by showing the user the consequences of his choices in terms of excluded options. When, for example, the user is confronted with the exclusion of options for feature N resulting from his choice for feature D, then he can decide to first determine feature N, before determining feature D. *The order in which the features are determined does in fact reflect the preferences of the user for the different options.* The user must realize this, because otherwise he will be confronted with the problems mentioned before.

Showing the consequences of a selection step comes to showing the excluded options that are the result of the selection step. In the example of Fig. 2 two options for feature G and one option for feature N are shown to the user. A simple algorithm to determine which options are ex-

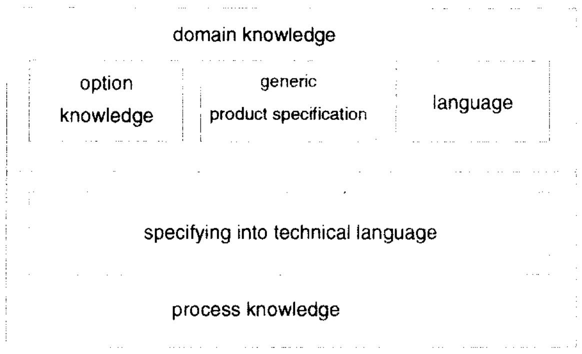

domain knowledge
option knowledge
generic product specification
language
specifying into technical language
process knowledge

Fig. 5. Model of a sales support information system (SSIS).

cluded because of a selection step is the following:

> Determine the set of allowed options for all features which have not been determined yet. Compare this set with the set of allowed options before the selection step. The difference of these sets is the set of the excluded options because of the selection step.

4. Design

A model of the design of an SSIS for the specification of a product into technical language is presented in Fig. 5. The SSIS will be considered a knowledge base system. This is a computer program in which as good as possible a separation has been made between the application-independent inference rules and the application-specific knowledge [3]. This last type of knowledge is implemented into a knowledge base (i.e. a database and a rulebase). This explains the name "knowledge base system".

The border between application-independent and application-specific is determined by the product where the system is used for. The application-independent inference rules contain the knowledge that is always used in the specification of a product (e.g., the algorithm for the determination of allowed options). This type of knowledge is called *process knowledge*. Application-specific knowledge is the knowledge about the specific type of product for which a specification has to be found (e.g., the exclusions). This type of knowledge is called *domain knowledge* [4].

Separating these two types of knowledge has the advantage that changes in the domain-knowledge module can be processed into the system without affecting the process-knowledge module. This demand is important when changes to the product occur often. Such changes can be caused by marketing considerations or engineering changes. Another advantage of separating the types of knowledge is the possibility to use the same process knowledge for different sets of domain knowledge. When, for example, a domain-knowledge module for kitchens is replaced by a domain-knowledge module for X-ray apparatus, then it is possible to configure X-ray apparatus without any changes to the system.

This modular approach also gives the possibility to work with so-called *local catalogs*. A local catalog contains the (sub)set of features and options which are locally offered (mostly determined by the country where the system is used). This is valuable when restrictions to the products depend on the country where the product is sold.

In the system a number of modules can be distinguished. We will take a closer look into those modules.

4.1. Domain-knowledge

Generic product specifications. The contents of a generic product specification has been described in Section 1.

Option knowledge. This module contains all relevant data for each option (e.g., "price", "name", "id#").

Language. When the system is to be used in several countries, the aspects which depend on the language are included in this module. This gives the possibility to easily exchange this module when the system is transferred to another country. The module contains messages that appear on the screen or the printer.

4.2. Process knowledge

Specifying to technical language. This module contains the necessary algorithms that control the specification process.

The specification process can be described as follows: The system permits the user to make selection steps in any order. In the remainder this is called *flexible specification*. The user selects an undefinite feature to make it definite. The system then calculates which options are still possible for the not-yet-determined features. Exclusions are shown to the user, who accepts or modifies his choice. After making a choice the user selects a new feature to determine. This is repeated until a complete, valid specification is made. Because of the algorithms used, it is guaranteed that this process ends. The Appendix gives an elaborated example for a part of this process.

5. The prototype

The prototype system is implemented on an IBM PS/2 model 50 with the programming language LPA-Prolog. This Prolog implementation is according to the Edinburgh syntax. The choice for Prolog is made because of its suitability for the realisation of a knowledge base system.

With the prototype flexible specification is possible. By using the algorithms mentioned before, it is guaranteed that the process always leads to a valid specification and that the process does not get stuck.

Building and using the prototype gave a clearer insight into several aspects:

5.1. Flexible specification

Until now it has been assumed that the system only contains one generic product specification, which itself contains a lot of variants. In practice, however, a product family will often be described by more than just one generic product specification. For instance in the case of kitchens, product specifications exists for all different manufacturers, i.e. Miele, SieMatic, Bruynzeel, etc. It may be obvious that the total number of variants is the sum of the number of variants within the different generic product specifications. Intuitively it is clear that a large number of variants asks for more processing effort from the system than a small number of variants. Therefore the first steps within the specification process should concentrate on features that decrease the number of variants considerably. (Most of the time these features will decrease the number of generic product specifications that are still valid.) This means that flexible specification should initially be limited to those types of features. When these features are specified, the remaining features can be specified in the same flexible manner. So a hierarchy in the specification process exists in a situation with a high number of variants.

5.2. Show consequences of a selection step

After making a selection step, the options excluded by that selection are shown to the user. Exclusion of an option can have two causes:

(i) The option forms part of a generic product specification that is not relevant anymore: the product specification cannot be chosen because of the selection step.

(ii) Because of the selection step a restriction causes the exclusion of the option.

The prototype shows both types of excluded options. Showing the first type of options can lead to a great number of data on the screen. This is especially true at the beginning of the specification process when each selection step may result in the exclusion of one or more generic product specifications. Referring to the remarks in Section 5.1, the system should initially only show excluded options of the set of features that is used to discriminate between product specifications. When this set of features is determined the system should show the other exclusions.

5.3. Response times

The calculation of the consequences of a selection step may take a long time. In the current prototype, an extreme case may take five minutes. This is due to the fact that determining the set of allowed options is done in an inefficient manner. The following algorithm speeds up the calculation:

> Partition the set of features in a number of subsets, using the following criterion:
>
> > Two features are element of the same subset if and only if they are a part of the same restriction.
>
> With this criterion, features that are not part of a restriction form a subset with only one element. As was mentioned before, an option x is allowed if there exists a valid specification starting with the union of the current specification and the set containing x. With the partitioning of the set of options, this demand can be replaced by the following demand:
>
> > For each subset, there must exist a valid subspecification, i.e. a specification satisfying all restrictions that are influenced by the features in the subset. When such valid subspecifications exist for all subsets, they can simply be combined into one specification. This specification is valid because no restrictions apply to two elements of different subsets. (This was the criterion on which the subsets were based.)

Determining a valid specification can therefore be replaced by the determination of valid subspecifications. It is however possible to sharpen the last algorithm:

> In the last selection step it is guaranteed that a valid specification can be obtained. This means that valid subspecifications exist for all subsets. Therefore, in determining whether or not an option is valid, it is sufficient to only look for a valid subspecification for the subset in which the feature is contained.

Using this algorithm in the prototype should definitely reduce the response times.

6. Conclusions

The goal of this research was to test the feasibility of a sales-support information system. Building a prototype of such a system showed that it is indeed possible to develop a computer program that:

(i) supports flexible specification of a product in an interactive way;

(ii) shows the consequences of each choice;

(iii) guarantees the validity of the resulting product specification;

(iv) guarantees that the specification process does not get stuck.

References

[1] E.A. van Veen and J.C. Wortmann, "Generic bills of material in assemble-to-order manufacturing", *Int. J. Prod. Res.*, Vol. 25, No. 11, 1987, pp. 1645–1658.

[2] E. van Veen, *Modelling Product Structures by Generic Bills-of-Material*, (Thesis, Eindhoven University of Technology, 1991), Elsevier, Amsterdam, 1992.
[3] N. Mars, "Onderzoek van niveau: Kennistechnologie in wording", (High-level research: The growth of knowledge technology), *Informatie*, Vol. 30, No. 2, 1988, pp. 84–90 (in Dutch).
[4] M.A.W. Theunissen, Kennis over kennis, (Knowledge about knowledge), Thesis, Hogeschool Eindhoven, 1987 (in Dutch).

Appendix

This appendix gives an example of a dialogue between a user and the SSIS. We hope that this will give a better notion of the ideas of this paper. Therefore, we will not specify the technical user-interface details (such as the use of pull-down menus, windowing techniques, use of a mouse, layout of the screen etc.).
The product consists of the following features and options: feature A with options a1, a2; feature B with options b1–b3, feature C with options c1–c5; and feature D with options d1, d2. Figure A.1 gives a schematic overview of the product in the style of Fig. 2.

The following restrictions apply to the product:

1	IF A(a1)	THEN C(c1)
2	IF A(a2) AND B(b2)	THEN NOT C(c3)
3	IF C(c2)	THEN NOT B(b2)
4	IF B(b2)	THEN D(d1)

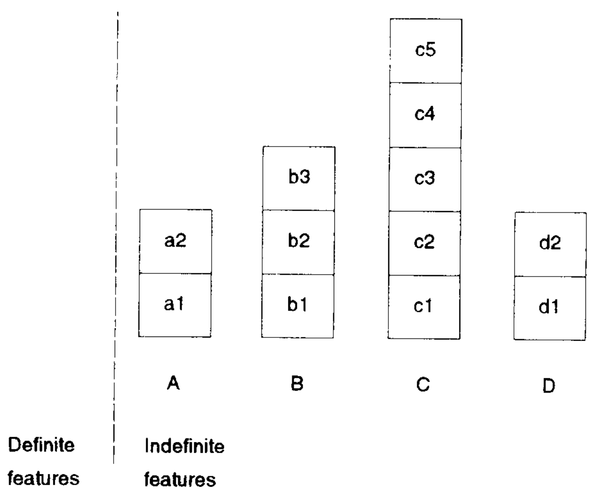


Fig. A.1. Situation at the start of the specification process.

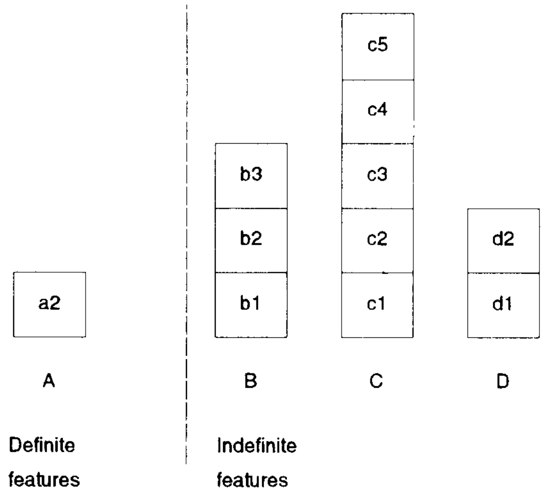


Fig. A.2. Situation after specifying feature A.

The following notations are used:

[text]	denotes considerations of the user or calculations of the system;
→	denotes the input by the user or the messages of the system to the user;
→( )	denotes a description of the input by the user or of the messages of the system to the user;
S	system;
U	user.

After each choice, confirmed by the user, a figure will give an overview of the definite and indefinite features with the allowed options.

The following dialogue can take place:

S → Feature to specify: ABCD
U → (selects feature A to specify)
S → Options to choose from: a1 a2
U → (selects option a1)
S [calculates the options that become non-allowed because of the choice of a1. Restriction 1 gives: c1 is obliged. Therefore, the other options for feature C are not allowed.]
→ Not allowed because of your choice:
Feature C, option c2
Feature C, option c3
Feature C, option c4
Feature C, option c5
Feature C definite, option c1
U [considers that he has some preference for option c2 and has no preference for neither a1 nor a2]
→ (selects option a2)

S [No restrictions can be found that results in non-allowed options because of this choice. The choice is confirmed by the system, so the set of definite features is extended. Eventually somewhere on the screen an overview of the choices is placed. Figure A.2 gives an overview of the current situation]
S → Feature to specify: BCD
U → (selects feature B to specify)
S → Options to choose from: b1 b2 b3
U → (selects option b2)
S [Restriction 2 gives: c3 is not allowed. The reverse of restriction 3 gives: c2 is not allowed. Restriction 4 gives: d1 is obliged, so d2 is not allowed.]
→ Not allowed because of your choice:
Feature C, option c2
Feature C, option c3
Feature D, option d2
Feature D definite, option d1

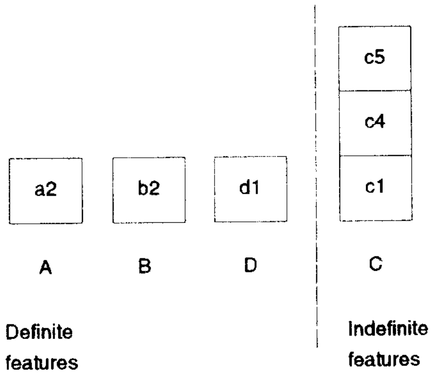


Fig. A.3. Situation after specifying feature B.

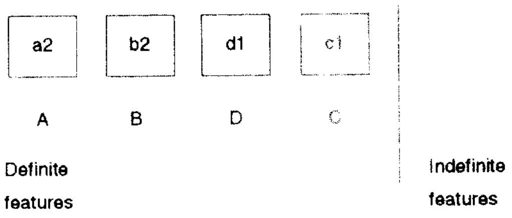


Fig. A.4. Final situation.

U [considers, that he has a greater preference for b2 than for c2, c3, and d2, so he accepts the exclusion of c2, c3, and d2.]
→ (confirms his choice for option b2)
S [Extends the set of definite features with features B and D and decreases the allowed options for the indefinite feature C. Figure A.3 gives an overview of the current situation.]
→ Feature D has also become definite (option d1).
S → Feature to specify: C
U → (selects feature C to specify)
S → Options to choose from: c1 c4 c5
U → (selects option c1)
S [No restrictions can be found that results in non-allowed options because of this choice. The choice is confirmed by the system, so the set of definite features is extended. Because there are no indefinite features left, the process is finished. Figure A.4 gives an overview of the final situation.]
→ Thank you for your order!