Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998), *12,* 301–305. Printed in the USA.
Copyright © 1998 Cambridge University Press 0890-0604/98 $12.50

POSITION PAPER

Defining configuring

DAVID C. BROWN
AI in Design Group, Computer Science Department, Worcester Polytechnic Institute, Worcester, MA 01609, U.S.A.

(Received October 31, 1997; Accepted February 5, 1998)

1. INTRODUCTION

This paper is intended to serve as part of the context in which the other papers in this special issue should be read. Its main goal is to revisit the basic definition of the configuration task, on which many people depend, to show some of its flaws, and to point out how it shapes thinking about the problem. We are concerned about characterizing the reasoning processes used to produce a configuration.

2. THE DEFINITION

The most commonly used definition of the configuration task:

> "Given: (A) a fixed, pre-defined set of components, where a component is described by a set of properties, ports for connecting it to other components, constraints at each port that describe the components that can be connected at that port, and other structural constraints; (B) some description of the desired configuration; and (C) possibly some criteria for making optimal selections."
>
> "Build: One or more configurations that satisfy all the requirements, where a configuration is a set of components and a description of the connections between the components in the set, or, detect inconsistencies in the requirements."

was given by Mittal and Frayman (1989, p. 1936).

For example, for the problem of building a software system from modules, the *components* are modules; the *ports* are the variables that need values or provide values; the *constraints* are descriptions of the number and types of values needed, or constraints about the compatibility of one module with another; and the *description* of the desired configuration is the user's description of what the software system is supposed to do.

Mittal and Frayman (1989) point out that three important aspects of configuration are:

1. one cannot design new components during the configuration task;
2. each component is restricted in advance to only be able to "connect" to other certain components in fixed ways (i.e., they can't be modified to get arbitrary connectivity); and that
3. the solution specifies not only the components in the configuration but also how they are related.

3. ADEQUACY OF THE DEFINITION

There are some problems with this definition. Even though we can not completely discuss the issues here, we will try to give some indication of what they are.

Mittal and Frayman use the word "connect" throughout, probably influenced by the computer configuration domain in which they were working. However, not every configuration has components that physically connect. For example, the components may influence each other with fields, or they may touch but not in any fixed position. Configurations are determined by relationships, of which connect and touch are examples.

There is also an issue with "ports." For example, it is hard to imagine where the ports are for some mechanical problems (such as gear pairs). This term is also tied to the idea of configurations whose parts are linked because something directly flows between them. It is not clear that must be true for all configurations.

This problem with ports and connection can be handled by concentrating on the idea of components with relationships between them. A port can be defined as "where" on a component a relationship acts, and by what kind of relationship it can take part in. It need not be a precise, fixed, single place, but might be an area, a portion of the component, or

Reprint requests to: David C. Brown, AI in Design Group, Computer Science Department, Worcester Polytechnic Institute, Worcester, MA 01609, U.S.A. Tel: (508) 831-5618; Fax: (508) 831-5776; E-mail: dcb@cs.wpi.edu, http://www.wpi.edu/~dcb/

all of it. It might be thought of as a variable with a specific range. It might be defined abstractly.

In general, the relationships might describe ways in which one component might influence another. So, gears might have torque-transferring ports, with a torque-transfer relation between them. Other components can be influenced by fields. While others might be influenced by data flow.

Other important issues raised by the Mittal and Frayman definition include at what level of abstraction the components are "predefined," and whether *all* or just some of the components need to be used in the configuration.

The issue of level of abstraction is related to 1 in 1–3 above. If the components allow additional refinement in any way—such as color, dimension, or material—there is the potential for producing something "new." This is most clear for dimensional refinement. An abstraction for the shape of an object's surface might refine to a square, or to a variety of distinctly different rectangles.

For more complex shapes the situation is worse, especially as it might affect the object's relationships (e.g., touching, or connecting). Hence, there is a possible interaction with point 2 above, as refinement might modify an object's allowed connections. This shows that allowing the complete refinement of abstract components takes the problem to the *edge* of the class of problems we can safely refer to as configuration.

4. DESIGN OR CONFIGURATION?

Refining abstract components by specifying the values for their attributes is usually thought of as a *Design* task (e.g., decide the dimensions for the components of a piston engine). It may be difficult, and quite nonroutine.

Design is a complex task that means different things to different people. Most "AI in Design" (Brown & Birmingham, 1997) researchers and "Design Theory & Methodology" researchers consider design to have several logical phases (Brown, 1991). These roughly correspond to the types of things that are being decided in that phase. These types of decisions include the functionality, the type of device, the general types of components, the configuration of types of components, the actual components, and the values of the attributes of those components.

Thus, the configuration task is an essential *ingredient* of the complete design task. The convenient distinction that is often made is that a design task produces (i.e., generates, or synthesizes) values for attributes, whereas a configuration task does not. Such distinctions are controversial.

For example, if one allows abstract components, there may be a need to specify some or all of them completely before a configuration can be produced—particularly if the allowed relationships depend on those values. Note though that configurations of abstract components can be produced, and that it may be useful to do so as part of a configuration process.

Some well-known, knowledge-based design systems claim to *explicitly* address the configuration task—for example, MICON (Birmingham et al., 1992). Others do not, despite having a strong flavor of it (e.g., Brown & Chandrasekaran, 1989; Steinberg, 1989). For example, my AIR-CYL system essentially configured by selecting between predetermined configurations.

There are two special cases that occur when the relationships between components are given. The first is when the components have parameters that need values, but no additional refinement is needed for the relationships between components. This corresponds to parametric design, and no configuration is being done. The second case also has the characteristic of parametric design, but is the special situation (demonstrated in ten Teije et al., 1996), where the components are given as *types*; those components are considered as parameters of the configuration, and the values decided are instances of those types. By deciding values, the configuration is being refined.

5. LOGICAL INGREDIENTS OF THE CONFIGURATION TASK

From now on we will try to refer to the process as "configuring" and the result as a "configuration." The task of configuring can be *logically* divided into several subtasks—we are not arguing that they are sequential or physically separate, as this depends on the techniques used to produce the configuration.

The Selecting subtask controls which components are selected. To be selected they each need to be able to play a part in satisfying the requirements, and they need to "fit into" the (current partial) configuration. Once selected, they have to be placed into that configuration. We will refer to that subtask as Associating. Another logical subtask is Evaluating.

Thus, logically:

Configuring = Selecting + Asociating + Evaluating

where:

Selecting = Choosing components;

Associating = Establishing relationships between components; and

Evaluating = Compatibility Testing + Goal Satisfaction Testing.

Note that as Selecting may be imperfect, Associating may fail. And, as Associating may be imperfect, Evaluating may return result a result of "poor."

The actual process used (i.e., the implementation) for a configuration system depends on how much about each subtask is known in advance, on how much knowledge is used in each subtask, and on the mix and order of these subtasks.

The actual process used for a configuration system also depends on whether knowledge from later subtasks can be moved forward into earlier subtasks to prevent failures. For example:

Selecting$_1$ = Choosing Components + Compatibility Testing.

It may be possible, for example, to ensure that only compatible components are selected. This sort of "knowledge compilation" process, where one piece of knowledge is compiled into another, has even been applied to the generate and test method, so that components generated do not need to be tested, as the generator (with the test compiled into it) only generates correct things (Mostow, 1991).

6. RELATING AND ARRANGING

Associating, the establishing of relationships between a partial configuration and a potentially compatible selected component, can be done at different levels of abstraction. For example, at the most precise level, geometric information can be given that describes exactly how and where component A touches component B. An abstract relationship might just specify *that* A touches B, for example.

We refer to Associating that uses abstract relationships, that is, those that permit additional refinement, as Relating. Associating using precise relationships is referred to as Arranging.

In some cases:

Configuring = Selecting + Relating + Arranging + Evaluating,

where

Relating = Establishing abstract relationships and

Arranging = Establishing specific relationships.

Examples of abstract (or "logical") relationships might be "next to," "touching," or "connected to." These do not specify the exact placement of one component relative to the other. Specific relationships, used in Arranging, will precisely locate one component with respect to another or with respect to some reference location. We refer to a configuration that has been produced by Arranging as an "arrangement." This seems to be compatible with colloquial usage of the term.

If there is any doubt that these concepts are different, imagine three pulleys placed in roughly a triangle, with a rubber belt that fits over the outside of all three so that the belt is pulled tight. What has been described here is a configuration.

Precise description of the positions of all three pulleys will constitute a particular arrangement (Fig. 1). Moving a pulley toward another pulley, hence changing the specific relationships, produces another arrangement (Fig. 2). Many tasks that we casually refer to as "configuration" (i.e., Configuring) also include Arranging. It is hard to imagine arranging being done without at least an implicit Selecting + Relating. It may be appropriate to consider tasks that we casually refer to as "arrangement" as configuring tasks with the Arranging portion dominant. The common task of producing a "layout" can be thought of as Arranging in 2D. Thus:

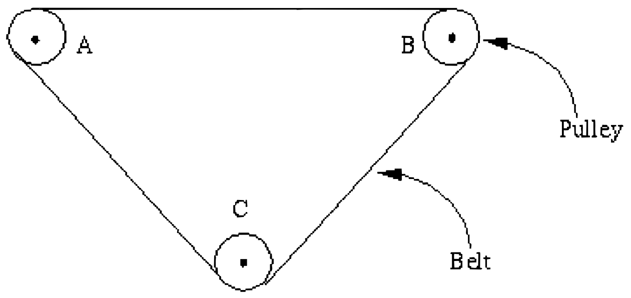

Fig. 1. Configuration1, arrangement1.

Laying-out = Arranging in 2D.

To allow *all* the possibilities present, we need to also include Arranging in 1 dimension. This points out the strong connection between configuring and planning, as, in planning, actions are configured into a plan in 1D, that is, time. Hence:

Laying-out = Arranging in 2D or **Arranging in 1D.**

By allowing relationships in "time," as opposed to "space," we can see that in most of the paper above there is an assumption that the relationships that describe the configuration are in terms of space. However, this need not be the case. It is easy to imagine relationships in time, weight, or color; for example, as these easily map in an analogical manner into space. Others may be harder to imagine, and are not commonly involved in what we think of as a configuration.

Note too that Associating, and hence both Relating and Arranging, might be in three *or more* dimensions. For example, some linguistic theories consider a "place" to be defined in time and space. Also consider the relationships

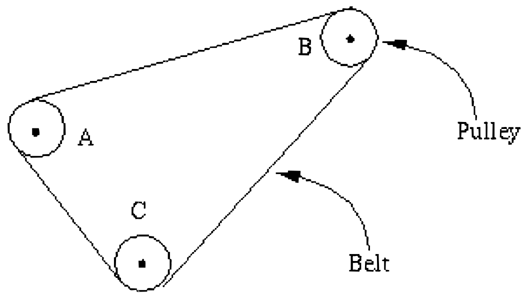

Fig. 2. Configuration1, arrangement2.

involved in software configuration. They too can be mapped into space, and perhaps also time.

While the analysis in this paper appears to be useful, other authors have presented different analyses. For example, for a more fine-grained analyses, see Runkel et al. (1992) and its references. Another useful discussion can be found in Wielinga and Schreiber (1997).

Note that our analysis does raise some problems. For example, (a) In Figure 2, is arrangement2 when viewed from the other side (i.e., looking *out* from the page) a different arrangement? (b) Is it possible to specify an abstract relationship such that the configuration changes?

Another issue, suggested by discomfort with part of Wielinga and Schreiber's analysis (1997), is which subtasks can be eliminated so that a Configuring task still remains. They suggest that even if the Selecting and Associating subtasks are not required and only Evaluating (what they call "verification") is done, then it is still a Configuring task. This, however, appears to miss the *essence* of Configuring, that is, the Associating subtask.

7. TECHNIQUES

A variety of techniques can be used together to support Configuring. Each technique supports different "ingredients" of the configuring process to a greater or lesser degree. Some of the key techniques are presented below.

Component choice plays a big role in how useful a component is in general. A larger and more complex component is more likely to have strong requirements for which other components also need to be included, and have less flexible use. Small components will probably provide more flexibility, but will require more configuring. Large components can be considered to be preconfigured sets of smaller components.

Experience and Knowledge affect the directness of the search for a configuration. Knowledge allows us to build structured descriptions of the available components, so that search is reduced. Experience allows us to build previously discovered subconfigurations or heuristic into the system, providing preferences that reduce errors.

Constraints are introduced by decisions. Selection of components introduces new variables and new constraints (Mittal & Falkenhainer, 1990). Thus, in general, configuration is a sequence of *Constraint Satisfaction Problems* (CSPs), a *Dynamic* CSP. Constraints can be used to record decisions made which do not directly correspond to objects in the system. Such a *partial choice* (Frayman & Mittal, 1987) can describe something that must be true of subsequent choices. In a *least commitment* manner this restricts the set of appropriate components, without deciding them.

Hierarchies record abstractions that are used to implement a least-commitment strategy, to allow a top-down strategy, to allow refinement guided by constraints, and to avoid the combinatorics produced by considering excessive detail too early. A *Component* hierarchy groups specific components into types and subtypes. A *Functional* hierarchy provides a way of storing functions organized by type and abstractness. *Part-subpart* hierarchies can be used for functions, for components, or both (Lee et al., 1992). A particular decomposition, if selected, provides a preformed configuration due to the part-of relationships imposed.

Templates refer to any preformed piece of configuration (i.e., from past experience). A template may associate functional and/or structural items, or record a decomposition. Templates include components and relationships between them at some level of abstraction. If alternative templates are available then selection criteria may be needed, or both alternatives can be explored.

Key Components correspond to those that are (almost) always required, or those on which many other choices depend, suggesting that their correct choice should take priority (Mittal & Frayman, 1989).

8. SUMMARY

We have discussed the definition of the configuration task, including some of its inadequacies; have described the relationship between design and configuration; have outlined one view of the problem-solving ingredients of configuration; and have related these ingredients to some of the different approaches to implementing the configuration task.

ACKNOWLEDGMENTS

This paper is a revised portion of "Some Thoughts on Configuration Processes," that appeared in *Proc. AAAI 1996 Fall Symposium on Configuration*. That paper was extracted from a longer 1992 report, sponsored by and written for Digital Equipment Corporation. The author would also like to acknowledge the feedback from other members of the WPI AI in Design Group, and from Dr. Mark Klein.

REFERENCES

Birmingham, W., Gupta, A., & Siewiorek, D. (1992). *Automating the design of computer systems: The MICON project*, Jones & Bartlett Publishers, Boston, MA.

Brown, D.C. (1991). Design. In *Encyclopedia of Artificial Intelligence*, 2nd ed., (Shapiro, S.C., Ed.), Wiley-Interscience, New York.

Brown, D.C., & Birmingham, W. (Eds.). (1997). Special double issue of *IEEE Expert on AI in Design*, *12(2)*, 3.

Brown, D.C., & Chandrasekaran, B. (1989). *Design problem solving: Knowledge structures and control strategies*. Research Notes in Artificial Intelligence Series, Pitman Publishing, Ltd., London, England.

Frayman, F., & Mittal, S. (1987). COSSACK: A Constraints-Based Expert System for Configuration. In *KBES In Engineering: Planning and Design*, (Sriram, D., & Adey, B., Eds.), pp. 143–166. Computational Mechanics Publications, Southampton, U.K.

Lee, C.-L., Iyengar, G., & Kota, S. (1992). Automated configuration design of hydraulic systems. In *AI in Design '92*. (Gero, J.S., Ed.), pp. 61–82. Kluwer Academic, Dordrecht, The Netherlands.

Mittal, S., & Falkenhainer, B. (1990). Dynamic constraint satisfaction problems. *Proc. 8th National Conf. Artificial Intelligence, AAAI-90*, 25–32.

Mittal, S., & Frayman, F. (1989). Towards a generic model of configuration tasks. *International Joint Conf. Artificial Intelligence, IJCAI-89*, 1395–1401.

Mostow, J. (1991). A transformation approach to knowledge compilation. In *Automating Software Design*, (M.R. Lowry & R.D. McCartney, Eds.), MIT Press, pp. 231–259. Cambridge, Massachusetts.

Runkel, J., Birmingham, W., Darr, T., Maxim, B., & Tommelein, I. (1992). Domain independent design system: Environment for rapid development of configuration design systems. In *Artificial Intelligence in Design '92*, (Gero, J.S., Ed.), pp. 21–40. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Steinberg, L. (1989). Design as refinement plus constraint propagation: The VEXED experience. *Int. Joint Conf. Artifical Intelligence, IJCAI-89*, 830–834.

ten Teije, A., van Harmelen, F., Schreiber, G., & Wielinga, B. (1996). Construction of problem-solving methods as parametric design. *KAW'96: Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Calgary, Alberta, Canada. (http://ksi.cpsc.ucalgary.ca/KAW/KAW96/KAW96Proc.html)

Wielinga, B., & Schreiber, G. (1997). Configuration-design problem solving. *IEEE Expert*, *12*(*2*), 49–56.