Some Thoughts on Configuration Processes

David C. Brown

Computer Science Department, Worcester Polytechnic Institute, Worcester, MA 01609, USA.
dcb@cs.wpi.edu, http://www.wpi.edu/~dcb/

1. Definition

The most commonly used definition of the Configuration task was given by Mittal & Frayman [1989, p. 1396]:

> "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."

For example, for problem of building a software system from modules, the *components* are a modules; the *ports* are the variables which 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 & Frayman [1989] point out that three important aspects of configuration are that:

- one cannot design new components during the configuration task;
- that 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
- the solution specifies both the components in the configuration as well as how they are related.

2. Inadequacy of Definition

There are some problems with this definition. Even though it is not appropriate to completely discuss the issues here, we will give some indication of the problems.

Mittal & 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 connect. For example, the components may influence each other with fields, or they may touch but not in any fixed position.

There is also an issue with "ports". It is hard to imagine where the ports are for some mechanical problems (e.g., gear pairs). This term is also very tied to the idea of configurations whose parts are linked because something directly flows between them. It isn't clear that must be true for all configurations.

Other important issues include at what level of abstraction the components are "predefined", and at what level they must finish up, as well as whether *all* or just some of the components need to be used in the configuration.

3. Design versus Configuration

Design is a complex task that means different things to different people. Most "AI in Design" 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 Configuration is an ingredient of the complete design task. The distinction which is often made is that a design task produces (i.e., generates, or synthesizes) components and values for attributes, whereas a configuration task does not. Such distinctions are controversial. For example, if one allows abstract components, there appears to be a need to specify them completely before a configuration can be produced. Some researchers refer to the process of configuring and then fully specifying as "Configuration Design".

Some knowledge-based design systems claim to *explicitly* address configuration -- for example, MICON [Birmingham 1992]. Others do not, despite having a strong flavor of it [Brown & Chandrasekaran 1989] [Steinberg 1989].

4. Ingredients of the Configuration Task

The configuration task can be "logically" divided into several subtasks. Components have to be selected. They each need to be able to play a part in satisfying the requirements and must fit into the (current partial) configuration. Once selected, they have to be placed into that configuration. We will refer to that subtask as Association. Another logical subtask is Evaluation.

Thus:

Configuration = Selection + Association + Evaluation

where:

Selection = Choosing components

Association = Establishing logical relationships between components

Evaluation = Compatibility Testing + Goal Satisfaction Testing

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. This is explored in the next subsection.

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:

Selection = Choosing Components + Compatibility Testing

It may be possible 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].

In some cases:

Configuration = Selection + Association + Arrangement + Evaluation

where:

Arrangement = Establishing specific relationships

Examples of "logical relationships", used in Association, might be "next to", or "connected to". These do not specify the exact placement of one component relative to the other. Specific relationships, used in Arrangement, will precisely locate one component with respect to another or with respect to some reference location.

If there is any doubt that these 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 an arrangement (Figure 1). Moving a pulley towards another pulley produces another arrangement (Figure 2).

Many tasks which we casually refer to as "configuration" also include Arrangement. It is hard to imagine Arrangement being done without at least an implicit Selection + Association. It may be appropriate to consider tasks which we casually refer to as "arrangement" as Configuration tasks with the Arrangement portion dominant. The Layout task can be thought of as Arrangement in 2D. Thus:

Layout = Arrangement-in-2D

Other authors have presented different analyses. For example, for a more fine-grained analysis, see [Runkel et al 1992] and its references.

5. Approaches to Configuration

5.1. Introduction:

The issues that concern reuse in systems that use any sort of pre-existing unit (e.g., a software module) are *Indexing*, *Mapping*, and *Instantiating*. Indexing is concerned with how to organize the modules so that the right ones can be selected. Mapping is concerned with how to convert the task model (i.e., the user's requirements) to a configuration. Instantiating is concerned with how to produce a complete description of a working system from the final configuration. A variety of techniques that address these issues are presented below.

When considering Mapping, we often describe it in a state-space fashion, where states are *transformed* to other states by Operators. States are incomplete, or partial, configurations. This is a rather "bottom-up" approach to configuration, adding one component at a time. The mapping may be viewed in a "top-down" fashion, where abstract descriptions of the configuration or parts of the configuration are "refined" into more concrete descriptions. This is known as *Refinement.*

5.2. Modules/Components

For configuration to be a generally useful approach to building systems (of any kind), the modules/components need to be reusable, the number not too large, and we have to ensure that we don't have to create new modules/components for each application that is built.

For software, a module is "reusable if it can be employed for several domains and tasks" [Klinker et al 1990], whereas a module is "usable" if a person who has limited programming skills can use it to build a program for a task. An excellent survey of reuse in software engineering is given by Krueger [1989].

It is difficult to produce generic, abstract models of tasks around which libraries of modules could be built. In some situations, it is likely that the many modules will be quite specific, and strongly tied to the particular situations for which they were developed. This is a negative factor if systems to be configured are going to be used in a wide variety of situations.

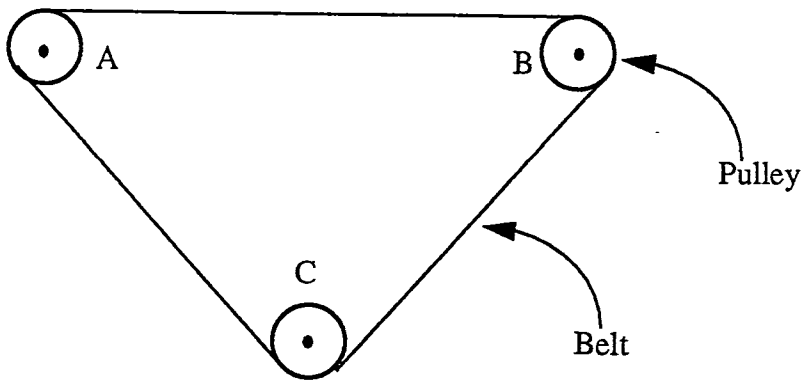

Figure 1: Configuration 1, Arrangement 1

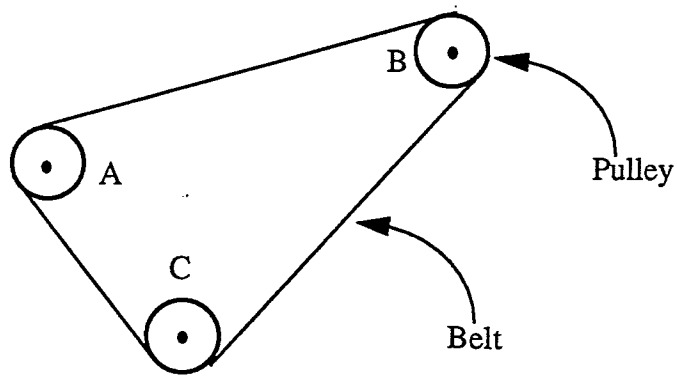

Figure 2: Configuration 1, Arrangement 2

In general, the more knowledge about a particular domain is encoded in a module, the more context-dependent it is, and the less likely it is to be useful in some other domain [Brown 1992]. In the DIDS system, for example [Runkel et al 1992], high reuse is obtained by having domain-neutral modules (mechanisms) which use externally defined domain information.

The size of the modules/components used, i.e., *grain size*, is an important issue. The larger the module the more likely it is to have strong requirements for which other modules it requires to have with it, and the less flexible its use will be. However, large modules can be considered to be preconfigured smaller modules. Therefore, less configuration needs to be done. Small modules will probably provide more flexibility, but will require more configuration.

Another issue is the level of abstraction of the module. Obviously, an abstract module cannot be "executed", as it will require "instantiating" -- i.e., being made specific. This observation also applies to Templates as well (see below).

5.3. Experience and Knowledge

A key issue in the implementation of Configuration tasks is how much experience and knowledge is available. Without these, given only a set of descriptions of the individual modules, the problem is one of searching for combinations of modules that satisfy the requirements. With no extra knowledge there is no way to guide the search.

Knowledge about the modules allows us to build structured descriptions about the configuration system's library of available modules, so that search is reduced. The structure allows us to relate groups of modules, so that deciding that one isn't suitable for inclusion in the configuration also decides that the others, which are related, also aren't suitable (i.e., pruning).

Research on building configuration systems has shown that they should include explicit, separate knowledge about modules/components, as opposed to having module knowledge implicit in the control strategy (e.g., hidden in search control rules).

Experience allows us to build previously discovered sub-configurations into the system. As these do not need to be configured, search is further reduced. It can also allow us to add heuristic knowledge, which may allow us to prefer one module over another in a particular situation (thus lowering the chance of having to backtrack and remake this decision), or could allow us to do things in an order which led to success previously.

5.4. Search

Almost any basic AI search technique could be used. Generate and Test could be used to "generate" configurations and test them for satisfaction of the requirements. Means-ends analysis could be used to try to reduce the difference between having no configuration, and having one that satisfies the requirements. As modules are added to the configuration, new differences emerge, and can be reduced. Heuristic searches, such as A*, could be used, which search by using an evaluation function applied to partial configurations, and use the best at every stage. Unfortunately, it isn't always possible to evaluate the quality of a partial configuration.

5.5. Constraints

It has been pointed out that design problems and configuration problems can be formulated as constraint reasoning problems. How easily this can be done depends on the exact nature of the problem.

If we know the pattern of connections, and have constraints on this pattern, but do not know the components, then this is equivalent to a crossword puzzle. This is a standard example used to explain Con-

straint Satisfaction techniques. A *Constraint Satisfaction Problem* (CSP) occurs when one tries to find a set of values for some variables which are related by constraints. CSP methods can be guided by knowledge and heuristics.

However, usually, we do not know the pattern of connections, and as Frayman and Mittal point out [1987], selection of components introduces new variables and new constraints [Mittal & Falkenhainer 1990]. Thus, configuration is a dynamic sequence of CSPs, a *Dynamic CSP.*

Constraints can also be used for checking compatibilities between choices. This would allow the compatibility of the ports of two components to be tested.

In general, whenever a decision (choice) is made, this will impose restrictions. These restrictions can be represented by constraints. Constraints can be "posted" to (i.e., attached to) the things they restrict, so that they can affect its subsequent use, such as its refinement. In some cases constraints can be "propagated" through a configuration, so that it flows across relationships between components to affect other components.

Constraints can be used to record decisions made which do not correspond to objects in the system. They can describe something which must be true of subsequent choices. This can be used to restrict the set of currently considered components, without deciding which. Frayman and Mittal refer to this as *partial choice* [1987].

Partial choice, along with constraint posting, can be used to implement a *least commitment* strategy, where choices are delayed, so that more information can be accumulated and premature decisions (which may be wrong) can be avoided.

5.6. Hierarchies

Hierarchies -- abstraction hierarchies, generalization hierarchies, or taxonomies -- record groupings between types of things that share common properties. The relationship between an object lower in the hierarchy with one higher is that it is less abstract, that it is a type of the higher one, and that it has all the properties of the higher one.

Abstraction is useful in problem-solving 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.

Refinement can easily be implemented by moving down the hierarchy, making decisions about which more specific lower thing to choose. These choices can produce constraints. These constraints can lead to more choices: hence, *constraint-guided.* Constraints attached to a node in an abstraction hierarchy will restrict all the subnodes of that node. This hierarchy can be viewed as an OR tree, with alternative refinements at every node.

The first form of abstraction hierarchy useful for configuration is the *Component hierarchy.* Specific components can be grouped into types, and those types into subtypes.

The second form of abstraction hierarchy useful for configuration is the *Functional hierarchy.* This provides a way of storing functions organized by type and abstractness.

In some systems, for example [Lee et al 1992], the two types of hierarchies are linked into one, with the functional hierarchy leading to abstract component types, and eventually down to specific components. This allows refinement to occur from function all the way to specific component, using the same technique.

Other hierarchies can be used to record parts and subparts, with an implicit "part-of" relation. This hier-

archy shows the "decomposition" of a thing. It is often used to show alternative subparts. This leads to an AND/OR tree, where the choice (OR) is between alternative decompositions (ANDs).

Part-subpart hierarchies can be used for both functions and for components. A particular decomposition, if selected, provides a preformed configuration.

This type of hierarchy can also be used to record the configuration, as each component selected is part-of the configuration. In some systems, the resulting configuration is always a portion of the tree built into the system. Thus the configuration can be recorded merely by marking that tree.

5.7. Templates

We will use the term *template* to refer to any preformed piece of configuration (i.e., from past experience). For convenience, we will call the things in the template "items". A template may associate functional or structural items, or a blend. If the system uses hierarchies, these items will also appear as nodes in the hierarchies. As with modules, templates can vary by size, by level of abstraction, and by how domain specific they are.

A template can also indicate the preferred the order of refinement of the items in it. Consequently, it can also take on some of the aspects of a *plan* -- i.e., a sequence of activities.

Systems that use templates vary in the way in which they use indexing to organize them. Templates are usually associated with nodes in a hierarchy.

A template may just be an alternative representation for a decomposition. However, the template may also include items which are not strictly part of the decomposition, but which are required by those items.

The relationships between the items in a template depend on their level of abstraction and whether they are of function or component type. They can include I/O relationships, connection, spatial relationships, constraints that relate them, and, for software, control sequencing information (e.g., While loops).

Structural templates (such as those used in MICON [Birmingham et al 1992]) can include abstract or specific components related to indicate the structure of that piece of configuration. For example, abstract electrical components might be wired together in the template. A functional template would probably have much weaker spatial relationships, but could have quite specific I/O relationships.

At any point during the configuration process there may be alternative Templates available. Thus a selection process is required. Preferences could be used to help select one over another [Frayman & Mittal 1987]. Constraints may also be used to prune the set of alternatives. Incorrect selection may lead to backtracking.

A more knowledge-intensive selection technique, such as that found in [Mittal & Araya 1989] or [Brown & Chandrasekaran 1989], might include an evaluation of the suitability of each template, and then selection from among the most suitable.

In some systems, the actual configuration task is controlled by a fixed sequence of tasks [McDermott 1982] [Birmingham & Siewiorek 1984]. Sometimes these tasks correspond to the top-level functional decomposition of the system to be configured. While strictly these are Plans, they can be considered as Task Templates. Skeletal plans are those which are slightly abstract and that require refinement.

5.8. Key Components

Mittal & Frayman [1989] point out that even if a particular function has been selected, there may still be a variety of ways of putting together components (or even subfunctions) to achieve that functional-

ity. To reduce the search space, they introduce the idea of Key Components. Thus if a "mapping from each function F_i to components C_i that are *key components* in providing F_i" is available, then the search for a configuration that provides F_i can be reduced.

The key component could correspond to a component which is (almost) always required. A more heuristic interpretation is that a key component is an item in a template on which many other decisions depend. This suggests that its correct choice should take priority.

7. 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 presented an analysis of different approaches to implementing the configuration task, such as hierarchies and templates.

8. References

W. P. Birmingham & D. P. Siewiorek, MICON: A Knowledge Based Single Board Computer Designer, *21st Design Automation Conference*, Vol.1, 1984, pp. 565-571.

W. Birmingham, A. Gupta & D. Siewiorek, *Automating the Design of Computer Systems: The MICON Project*, Jones & Bartlett Publishers, 1992.

D. C. Brown, Design, *Encyclopedia of Artificial Intelligence*, 2nd edn., (Ed) S.C.Shapiro, Wiley-Interscience, May 1991.

D. C. Brown, The Reusability of DSPL Systems, Proc. Workshop on Reusable Design Systems, *Second Int. Conf. on AI in Design*, June 1992.

D. C. Brown & B. Chandrasekaran, *Design Problem Solving: Knowledge Structures and Control Strategies*, Research Notes in Artificial Intelligence Series, Pitman Publishing, Ltd., London, England, May 1989.

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

G. Klinker, C. Bhola, G. Dallemagne, D. Marques & J. McDermott, Usable and Reusable Programming Constructs, *Proceedings of 5th Knowledge Acquisition Workshop*, AAAI, 1990. (Also in: *Knowledge-Acquisition*, Vol.3, No.2, pp. 117-135).

C. W. Kreuger, *Models of Reuse in Software Engineering*, CMU-CS-89-188, 1989.

C-L. Lee, G. Iyengar & S. Kota, Automated Configuration Design of Hydraulic Systems, *AI in Design '92*, (Ed.) J.S.Gero, Kluwer Academic, 1992, pp.61-82.

J. McDermott, R1: a rule-based configurer of computer systems, *Artificial Intelligence*, Vol. 19, 1982, pp. 39-88.

S. Mittal & A. Araya, A Knowledge-Based Framework for Design, *IJCAI*, Vol.1, 1989, pp. 856-864.

S. Mittal & F. Frayman, Towards a generic model of configuration tasks, *IJCAI*, Vol. 2, 1989, pp. 1395-1401.

S. Mittal & B. Falkenhainer, Dynamic Constraint Satisfaction Problems, *Proc. 8th Nat. Conf. on AI, AAAI-90*, 1990, pp. 25-32.

J. Mostow, A Transformation Approach to Knowledge Compilation, In: *Automating Software Design*, (Eds) Lowry & McCartney, MIT Press, 1991.

J. Runkel, W. Birmingham, T. Darr, B. Maxim & I. Tommelein, Domain Independent Design System: Environment for Rapid Development of Configuration Design Systems, In: *Artificial Intelligence in Design '92*, (Ed.) J.S.Gero, Kluwer Academic Publishers, 1992, pp. 21-40.

L. Steinberg, Design as Refinement Plus Constraint Propagation: The VEXED Experience, *IJCAI*, 1989, pp. 830-834.

This paper is a revised version of part of a longer report, originally written in 1992, for Digital Equipment Corporation.