A Generative Constraint Formalism for Configuration Problems*

Markus Stumptner  Alois Haselböck
Institut für Informationssysteme, Technische Universität Wien
Paniglgasse 16, A-1040 Vienna, Austria

Abstract. Traditionally, constraint satisfaction systems have been considered an especially well-suited representation to configuration problems. However, a conventional constraint system with a predefined set of variables does not capture the flexibility inherent in composing systems out of a multitude of components of varying types. We propose an extended constraint satisfaction scheme that allows the incremental extension of a constraint network in accordance with the component-oriented view of configuration. Components can be individually represented and connected, while resource constraints express non-local requirements on the interaction of components. Constraints may be generative in that they lead to introduction of new variables, and are generic in that they may be defined to hold for all components of a given type.

1 Introduction

Design and configuration are traditional testbeds and application areas for AI techniques. Basically, configuration of technical systems is the task of selecting an appropriate set of components from a pre-defined “kit” (note that this set of available components must not be explicitly enumerated, but is usually described in a generic manner), assembling the selected components and thus deriving the structure of the resulting system, and possibly adjusting certain component parameters. Despite the success of rule-based configuration expert systems such as R1/XCON [McD82], recent years have seen a trend towards more soundly based knowledge representation schemes in various areas of AI. For configuration problems, constraint satisfaction (CS) techniques were embraced in a number of prototype systems [MA86, CGS+89] and form the foundation of the configuration approach described in [MF89], which focuses on the assembly of systems by connecting components.

Example 1. (from [MF89]) Assume a computer system has to be configured to meet an order specification of a customer. One of the required features is a printing function. The key component implementing such a printing function is a printer. After the selection of an appropriate printer type, additional components like interface cable or driver software must be chosen appropriately. This

* This work was supported by Siemens AG Austria under project grant CSS (GR 21/96106/4).
example demonstrates both function-component relationships and the idea of key components which, if chosen for configuration, result in the selection of a set of subordinate components.

However, the traditional constraint approach is limited in expressiveness and does not provide hooks for representing important features of configuration problems, namely the organization in terms of components and the dynamicity of the problem, i.e., the fact that the number of components may not be predetermined. In contrast to rule-based systems, where new components are explicitly introduced into working memory by actions specified in rule RHS's, the traditional constraint approach, while cleaner and more declarative, is effectively static. Mittal and Falkenhainer proposed dynamic CSPs (DCSPs) in [MF90] to avoid the latter problem, pointing out improved clarity and optimization possibilities.

Example 2. (from [MF90]) The variable package with the domain \{luxury, deluxe, standard\} denotes the three offered car models in a car configuration domain. The luxury package provides an \textit{air\_conditioner}. In the notion of DCSP, this can be specified by the following constraint:

\[
\text{package} = \text{luxury} \Rightarrow \text{air\_conditioner}.
\]

This constraint establishes the activation (i.e., inclusion of the corresponding component) of the \textit{air\_conditioner} variable in certain situations (if the \textit{package} is \textit{luxury}). In addition, compatibility constraints restrict the configuration space. The following example states that, if both an \textit{air\_conditioner} of type \textit{acl} and an automatic sunroof opener are chosen, a battery of capacity \textit{medium} must be included in the configuration:

\[
\text{air\_conditioner} = \text{acl} \&\& \text{opener} = \text{auto} \rightarrow \text{battery} = \text{medium}.
\]

However, the DCSP approach still suffers from strong limitations, since the component structure demanded in [MF89] remains unclear, while multiple occurrences of components with equal behavior are not supported at all. E.g., in describing a twelve-cylinder engine, every constraint on cylinder behavior would have to be written down twelve times. Bowen and Bahler [BB91] use a language based on the semantics of Free Logic to address this problem.

A different trend led to the development of strongly dynamics-oriented representation schemes for configuration. These were based on the realization that a central activity during configuration is the accumulation of entities that together provide for some needed functionality. The main assumption of the so-called resource paradigm is that functionality-based selection from a component library should be the central operation in a configuration system [HJ91].

Example 3. Consider another example from a computer configuration domain. The customer requests a PC with 8 MB RAM. The company has memory units of 2 MB capacity a piece in stock. A motherboard is initially equipped with 2 MB, but it has vacant slots for up to 32 MB. Thus, after the selection of the motherboard alone the working-memory resource shows a deficit. In the resource-based approach, configuration is fundamentally considered a task of resource balancing.