A logic-based description of configuration: the Constructive Problem Solving approach

Rüdiger Klein
Daimler-Benz AG – Research Dept.
Knowledge Based Systems Group (F3/SW)
Alt-Moabit 96a
D-10559 Berlin
Tel.: (++49–30) 399 82 211
email: klein@DBresearch-Berlin.de

1. Introduction

Configuration problems have been in the focus of AI for a long time. Different AI techniques have been applied for configuration problem solving: rule based approaches [McDermott82], object-oriented techniques [Plakon91], structural refinement [Tank90], various grammars [BMS94], resource based techniques [Heinrich89, SW92, HJ91+96], or constraint based ones [SH92, FW94] – just to mention a few. Though there are quite a lot of applications, the real success of these approaches has been quite limited up to now. The main reason for this seems to be that each of these approaches reflects only certain aspects of configuration (more or less in an ad-hoc manner). What is missing is a comprehensive and well-formalized AI paradigm of configuration which would provide the foundation to integrate the different aspects in a well-defined way. This AI paradigm of configuration should reflect the essence of knowledge based configuration:

- Configuration problem solving is essentially a generative process: a solution has to be generated which fulfills the different goal and consistency requirements. The central point here is to have a well-founded, logic based formulation of generative problem solving (in contrast to "traditional", proof-oriented formalizations).

- Configuration problems need an expressive knowledge representation allowing to describe the many different knowledge aspects in an integrated way: generic versus case-specific knowledge, taxonomies, part-of hierarchies, structural and arithmetic constraints, etc., and which allow us to use this knowledge in the configuration typical way: as taxonomic reasoning, hierarchical refinement, constraint specification and solving/propagation, resource-based reasoning, etc.

Recently, a number of approaches have been formulated allowing to integrate different formalisms in a well-defined way: the CLP scheme of Jaffar and Lassez [JL87] integrating logic and constraint programming, the Höhfeld-Smolka scheme [HS88], or Baader and Hanschke’s concrete domains [BH91] integrating taxonomic and constraint reasoning – just to mention a few. The point is, that all these approaches are deductive ones, i.e., they are proof oriented, not generation oriented (for an overview see for instance [Lloyd90]). On the other side,
constructive or generative aspects play a major role in a variety of problem classes: design, configuration, planning, scheduling, etc. Thus research is under way at many places on such topics as abduction [O'Rorke90+91; Lloyd90], model generation [MB88; DD92; FHKF92], model elimination [Stickel85; BB91+93], dynamic constraint techniques [MF89, FW94] etc., which could provide a suitable formal platform for these constructive problem classes.

Configuration problems provide for a good test case for a general formalization of generative problems because of their inherent clear structures: we can imagine configuration (maybe in a certain approximation) as a problem class with no indefinite goals, no unspecified constraints, with completely described objects, relations and constraints between them, etc. They do not suffer (so much) from uncertainty, vagueness, or incompleteness problems as other application classes do (for instance design and planning). The main problems encountered in configuration tasks can be summarized as follows:

- Given the expressiveness requirements and the needed generative type of problem solving – how to deal with the resulting complexity?

- Given the various types of inferencing (taxonomic, combinatorial, arithmetic constraints) – how to integrate them?

- Quite often configuration problems are underspecified – i.e., the real problem is not to find any solution, but to find an optimal or a good one (how that ever may be expressed). These optimality issues as well as the complexity problems require sophisticated control facilities.

The Constructive Problem Solving [Klein91, KBN94] has been developed as an expressive, well-formalized approach to configuration which provides the basis to solve these problems. This paper is organized as follows: In the next chapter we’ll describe a very abstract view on configuration and how this can be used to get a comprehensive formalization. In chapt. 3 the basic idea behind the Constructive Problem Solving formalization is outlined – the model generation approach. In chapt. 4 the main properties of the CPS calculus will be discussed in an informal way – in order to demonstrate the principles of model generation and to characterize the most essential requirements to the CPS inferences and their interplay. In chapter 5 we conclude with a discussion of the merits and problems encountered with CPS, followed by a brief description of open research issues.

2. Constructive Problem Solving: Configuration as Model Construction

The starting point for a comprehensive formalization of configuration problem solving is the determination of the typical knowledge structures and how they are related in problem solving.

A comprehensive analysis of the typical knowledge structures in configuration problems revealed, that configuration knowledge can be represented within the following main knowledge categories:

- First, we have a clear distinction between object-level knowledge, problem solving knowledge, and control knowledge. The object knowledge includes an explicit representation of all the objects and object classes (or types, or sorts), their relevant properties (attributes), relations, constraints, etc. The problem solving knowledge reflects the configuration-typical ways to deal with the object-level knowledge (for instance the ge-
nerative way of problem solving), and the control knowledge represents the various strategies, heuristics, etc.

- Due to the clear structures typical in configuration, the object-level knowledge can be divided into two well-distinguished categories: general domain knowledge, and case-specific knowledge. The case-specific knowledge consists of knowledge about the concrete configuration problem to be solved (a set of functional and/or structural goals or requirements) as well as of knowledge about the generated solution. In configuration problems, such a solution normally has to be formed out of descriptions of technical devices by which a concrete system is built up as well as their relationships. For instance, descriptions of the technical devices as they show up in a catalogue can be part of a solution. In contrast, the goal specification can contain more general statements (leaving certain degrees of freedom for the overall solution generation). Thus the general domain knowledge should contain expressions (called the definitional knowledge), which relate the more general notions in the domain to the more specific vocabulary.

- Beside the definitional knowledge the general domain description should contain various forms of consistency information (constraints). They facilitate the explicit representation of knowledge about inconsistent solutions (negative constraints) as well as knowledge about necessary preconditions of any solution, especially knowledge about completeness of solutions. The main forms of constraints relevant in configuration can be summarized as follows:
  - taxonomic constraints: expressing necessary sort conditions of certain objects;
  - structural constraints: objects must be related somehow; including relations between compound objects and their components (abstraction hierarchies);
  - cardinality constraints: certain relations can only exist in a certain number for a given object (or must exist in a certain number);
  - logical constraints: if certain objects fulfill certain conditions they have to fulfill other constraints, too;
  - arithmetic constraints: real valued attributes are related by arithmetic equations, disequations, and/or inequalities
  - A special form of constraints which play a prominent role in many configuration problems are resource constraints [HJ89, SH92, HJ91+96].

The identification of these configuration-typical knowledge structures has been shown to be an essential precondition for the formulation of the CPS calculus, which makes use exactly of these categories.

The basic principles of configuration knowledge given have been illustrated with an example from programmable logic control (PLC) devices [KBN94]: The definitional knowledge, for instance, has to represent the fact that cpu boards, communication boards, power supply units, etc. all are boards, or that crates can be main crates or additional crates. Constraints describing consistency information in the domain are, for instance, the need for every board to

1. This will not be dealt with here. We only want to point out, that the explicit representation of the object level knowledge as well as the formulation of the configuration-typical CPS inference rules provide a necessary pre-requisite for an adequate representation of the control knowledge.

2. To have a solution containing assertions like "any cpu" or "any crate" makes no sense, because nobody will be able to insert "any cpu" or "any crate" into the concrete configuration. What one can say is take a cpu of type "xyz-123" or a crate of type "abc".
be placed in any crate, the need for every main crate to contain a cpu board (completeness constraints), or that various board types as well as main and additional crates are incompat-
ible (negative constraints). A goal then contains, for instance, a set of boards needed in a specific configuration (each described by a set of attributes) and some relations they should possess.

3. The Basic CPS Idea

We will now outline the basic idea underlying our formalization of configuration: a configuration problem solver gets as input a goal specification of the system to be configured (a set $G$ of logical formulas) and produces as output a semantical model $C$ of this specification. Of course, this idea needs some refinement in order to be useful.

We suppose that we are given a logical language, with signature $\Sigma$, that allows us to talk about the systems we are interested in. In addition we assume that there is a sublanguage, called the basic language, whose signature $\Sigma_{\text{basic}}$ is a subsignature of $\Sigma$ – that gives us the vocabulary to name the technical devices by which a concrete system is built up as well as their relationships. One can imagine, for instance, that the names of the technical devices in a catalogue are part of $\Sigma_{\text{basic}}$. We assume that the basic language is rich enough to describe concrete systems. We distinguish between the two languages because the specification of a system to be configured will be given using the overall language – thus providing more expressiveness there (including abstractions and various degrees of freedom), whereas descriptions of models that satisfy the specification consist only of basic formulas.

The representation of the domain knowledge about technical systems will consist of the two parts: the definitional knowledge, that relates the abstract notions, i.e., the elements of $\Sigma \setminus \Sigma_{\text{basic}}$, to the basic language, and the consistency information. The definitional knowledge will be represented by a set $D$ of formulae in our language. The consistency information will be a set $I$ of integrity constraints.

Now we are ready to outline the main idea of Constructive Problem Solving [Klein94]: given a configuration domain (characterized by the definitional knowledge $D$ and the domain constraints $I$), and a concrete configuration problem (described by a goal $G$), a solution will be a finite model $C$ fulfilling the following conditions\(^3\):

\[
C \models D \exists G
\]

and

\[
C \models D \Gamma
\]

Thus Constructive Problem Solving (CPS) represents in a well-formalized way what is actually happening in configuration: to a given configuration problem $G$ a solution $C$ is generated which is a model, i.e., which has to fulfil this goal and the general constraints in the domain.

In this way CPS model construction provides a well-founded approach to the synthetic type of configuration problem solving.

4. An Illustration of CPS

After these abstract definitions and specifications in a brief and informal manner the key ideas of CPS shall be illustrated (using parts of the PLC example [KBN94]).

**Definitional knowledge:**

3. with $\exists G$ being the existential closure of $G$
The definitional knowledge $D$ relates more abstract notions (or concepts, or sorts) to more concrete ones (and finally to basic ones). For instance, it allows us to express that a board is either a cpu board, or a communication board, or a power supply unit (or some other here not mentioned):

$$\text{board} := \text{cpu} \lor \text{comm-board} \lor \ldots \lor \text{power-supply}.$$ 

This means that each cpu instance will be an instance of board, too – thus this kind of definitional knowledge results in taxonomic hierarchies. Vice versa, it also tells us that each board instance has to be either a cpu instance, or a comm-board instance, or ... a power supply instance (and nothing else).

In the same way abstract relations (like 'connected') could be related to more concrete ones (like 'left-connected' or 'right-connected') which can be realized in a concrete configuration. We can also express, for instance, that a certain concept has only a pre-defined number of instances:

$$\text{voltage-values} := \{4.5, 9, 24, 110, 220\}.$$  

Consistency information:

In configuration problems, consistency information could have many different forms. Three of them will be mentioned here:

1.) implications:
they have the general form

$$p_1 \land p_2 \land \ldots \land p_n \rightarrow q_1 \land q_2 \land \ldots \land q_m$$

with the meaning that if the preconditions $p_1, p_2, \ldots, p_n$ are fulfilled by the generated solution the conclusions $q_1, q_2, \ldots, q_m$ have to be fulfilled, too.

In our example, for instance, we want to represent the consistency condition that every board has to have a place in a crate:

$$\forall B \cdot \exists C \cdot \text{board} \rightarrow \text{crate} \land \text{placed}(B,C)$$

(with 'placed' being a binary relation). Another important form of consistency information to be expressed in many configuration domains are cardinality constraints: for instance we want to say that there is only a limited number of places in a certain crate. In CPS this means that the new goal placed$(B,C)$ can only be fulfilled in a certain model if these cardinality restrictions are fulfilled.

Many aspects of configuration knowledge can be represented in this form of logical implications: that all instances of a certain class (sort) have to have certain properties and fulfill certain constraints, or that a compound object has to consist of a set of parts and how these parts must be related to the whole and to each other, etc.

2.) negative information (denials):
can be represented by a special form of implications: their "conclusion" is the special expression 'false' (written '⊥'). The denial that a cpu board must not be placed in a small crate can be expressed in this way:

$$\forall B \forall C \cdot \text{cpu} \land C: \text{small-crate} \land \text{placed}(B,C) \rightarrow \bot$$

with the meaning that the fulfillment of the preconditions signals an inconsistent state.

4. Variables occurring in pre-conditions (and maybe conclusions) are all-quantified, variables only occurring in conclusions are existentially quantified. This last case results in the generation of new variables and maybe objects as part of the solution generation.
3.) resource constraints:
This is a form of constraints typical for many configuration problems. For instance, we want to
guarantee that in every crate the sum of the power consumed by the different boards is less
(or equal) than the power provided by the supply units in the same crate. Taking classical
constraint techniques, two points are worth mentioning here:

- the arithmetic constraints run over sets of expressions which at definition time are not explic-
  itly but only implicitly (or intensionally) specified. Only at run time it can be decided which
terms have to be included in the sum.

- a violation of this constraint does not necessarily result in a backtracking. The typical form of
  resource based reasoning is that resource constraint violation triggers a kind of repair opera-
  tion. In the example given here, for instance, an additional power supply unit could be gener-
  ated and placed in the underbalanced crate.

Formally these resource constraints can be expressed as constraints on sums over intension-
ally expressed terms:

\[ \text{sum}(C \mid p(C)) \geq \text{sum}(C' \mid q(C')) \]

where the \( C \) and \( C' \) are variables which will be instantiated on run time to all terms fulfilling the
logical expressions \( p \) and \( q \).

In our PLC example the mentioned power balance constraint can be formulated in CPS as
follows:

\[ \forall C\ C:\text{crate} \rightarrow \sum \{ P \mid S:\text{power-supply} \land \text{placed}(S,C) \land \text{power}(S,P) \} \geq \sum \{ V \mid B:\text{board} \land \text{placed}(B,C) \land \text{consumes}(B,V) \} \]

Goals and solutions:
After this short discussion of definitional knowledge and consistency issues we can show how
a concrete configuration problem can be represented and how it can be solved within CPS. A
concrete configuration problem will be represented as a set \( G \) of goals, for instance:

\[ G = \{ b1:\text{comm-board}, b2:\text{comm-board}, ..., c1:\text{cpu}, \ldots \} \]

with each maybe described in more detail by certain properties it should have. Generating a
solution for this problem means to construct a model which fulfills this goal as well as the
constraints characterizing this domain in general. For instance, this includes:

- to select basic concepts for the boards which agree with the goal specification (i.e.,
  which are subconcepts of comm-board);

- to generate new goals in order to fulfill the general consistency conditions (for instance
  the placement constraint which results in the generation of \( C:\text{crate} \) and \( \text{placed}(B,C) \)
goals);

- to guarantee the resource constraints (which may result in the generation of new goals).

A solution to the given example goal could be the model

\[ C = \{ b1:\text{comm-board12}, \text{cr0:crate3, placed}(b1,c1), b2:\text{comm-board23}, \text{placed}(b2,c1), \ldots, c1:\text{cpu33, placed}(c1,\text{cr0}), \ldots \} \]

This short discussion gives an impression how CPS incrementally generates a solution as a
model. This includes the generation of new objects and constraints.
5. Problem Solving In CPS

The various forms of knowledge relevant in configuration problems result in a corresponding manifold of problem solving issues. The main advantage of CPS is that it provides a common formal framework of how they are to be formulated and of how they must interact.

A central issue in configuration is the relation between functional and structural aspects. Many requirements (goals) are given as functional specification (though also structural ones are common). They have to be "mapped" somehow to structures which fulfill them. The main advantage of configuration is that this mapping is relatively simple (in contrast, for instance, to design in general). In many cases it simply results in some taxonomic constraints on components and in constraints on their attributes.

The taxonomic knowledge results in taxonomic reasoning: having a goal bl:board means to select an appropriate basic subsort of board which fulfills all the other constraints existing or comming up for this object bl. Having a goal B:board also could mean to select one of the existing objects in the (partial) model to be unified with variable B (thus fulfilling this goal). Logical constraints generate new goals if their preconditions are fulfilled, denials signal inconsistent states.

Cardinality constraints on relations and constraints on finite sets may result in combinatorial and finite domain inferences: knowing that only n boards can be placed in a crate means to generate a sufficiently large number of crates and to find such an assignment for each placement goal of the m existing boards that the cardinality as well as all the other (for instance power balance) constraints are fulfilled. Here especially important is the aspect of symmetry: in many cases the problem space is symmetric w.r.t. interchanges of components – such combinatorial search in such subspaces will not contribute to find really new solutions.

Hierarchical structures are quite common in many technical applications. Thus hierarchical refinement is one of the central problem solving activities. As mentioned in the previous chapter, formally it resembles to dealing with general logical constraints. It should be controlled carefully: a lot of efforts can be wasted if done in an inappropriate way (for instance too early).

6. Discussion

The model generation approach of Constructive Problem Solving provides a well-formalized framework for the description of configuration problems. It can be realized on expressive representation languages including cardinality and resource constraints, and it reflects the generative type of configuration problem solving based on a standard declarative (Tarsky-style) semantics. For a given representation language a CPS calculus can be given including a set of inference rules which realize the intended semantics of model generation. In this way configuration problems can be solved incrementally, and they can be solved making use of standard logic and constraint solving techniques.

Knowing these advantages of CPS – which are the problems? There are mainly two which are closely related: optimality and complexity.

The optimality problem simply means that normally one is not interested to get any solution to a given problem, but "the best" or at least a "good" one. Thus from the many models which can normally be generated for a configuration problem, only a small amount will be considered as real solution candidates or solutions. Some of the optimality criteria can be represented (or
approximated) as local constraints or local decision criteria ("take the smallest/cheapest" etc.). But that's not generally possible, and sometimes it is even contra-productive. In those cases only an overall estimation of the total solution (or of larger parts of it) can help.

Complexity issues are a main issue in nearly every knowledge based application. That's true in proof or classification-oriented approaches, and it is even more relevant in generative problem solving (see for instance [EG92]). Theoretically the number of models which can be generated to solve a given configuration problem is so large that there is no chance to investigate only a small part of the search space (not to mention those parts of the problem space which do not contain any solution but which must be searched through).

Both problems are serious, and they are not specific to the way configuration problem solving is formalized in CPS: they are inherent to every adequate approach to configuration. Three research issues could be derived from these considerations:

- In order to avoid unnecessary efficiency problems, use the various available techniques in the most appropriate way: constraint solving on arithmetic constraints à la CLP(R), finite domain constraint propagation techniques to deal with combinatorial and cardinality aspects, feature unification for taxonomic reasoning, etc. The main point here is to find an adequate integration of these techniques...

- Control is of central importance in order to find any solution and to find a good or optimal solution. Not only the way in which decisions are made is essential, also the sequence in which they are made. For instance, the attractiveness of resource-based configuration comes (at least in part) from the fact, that this type of problem solving allows for intermediate inconsistent states (and provides appropriate repairs). – What needs further research is how control knowledge (strategies, heuristics, etc.) could be represented, which expressive means are needed, and how it is interrelated with the the object level knowledge and the various types of inferences.

- A possible consequence from these considerations could be that configuration in its full extend is still too hard for current AI technologies. But what could be realistic today is (besides configuration checking) an interactive configuration assistant, where not only the goal can be specified interactively and incrementally, but also (the more important) decisions can be made interactively by a human user. Of course, they can also be withdrawn and corrected interactively. This means that dependency maintenance technology has to be integrated into all the other (not even simple) problem solving techniques.

A system following this approach is just under development at our research lab. Later we hopefully can report some concrete experiences therefrom.

Acknowledgement

The author wants to thank his colleagues Frank Feldkamp, Michael Heinrich, Ernst-Werner Jüngst, Helmut Ritzer, and Klaus Schild from the Daimler-Benz Knowledge Based Research Group for their steady interest and lively discussions.

References

[BH91]
[BB91]  

[BB93]  

[BMS94]  

[DD92]  

[EG92]  

[FHKF92]  

[FW94]  

[Gero92]  

[Gero93]  

[Gero94]  

[Goebel92]  

[GR93]  

[Heinrich89]  

[HJ91]  

[HJ96]  
M. Heinrich, E.-W. Jüngst: "The resource-based paradigm: configuring technical systems from modular components", proc. of this workshop.

[HS88]  

[JL87]  

[Klein91]
[Stickel85]

[SW91]

[Tank90]

[TN94]