ConStruct: Combining Concept Languages with a Model of Configuration Processes

Harald Meyer auf'm Hofe
German Research Center for Artificial Intelligence (DFKI)
Postfach 2080
D-67608 Kaiserslautern, Germany
Harald.Meyer@dfki.de

Abstract

This paper revises the idea of representing configuration processes by design decisions and design goals where the latter represent decisions to be made. An approach on knowledge based configuration is presented that is based on this idea covering the issues well-founded knowledge representation, off-line search for a configuration, interactive modification of configurations by dependency-directed backtracking, and support in constructing and maintaining knowledge bases.

Introduction

Well-founded formalizations of configuration typically represent configuration tasks as the problem of constructing a model of the current specification due to a theory that represents general knowledge on applicable configurations (Buchheit, Klein, & Nutt 1994; Schröder, Möller, & Lutz 1996). These approaches exhibit mainly three drawbacks:

1. Yet, logical frameworks for configuration concentrated on structure-oriented configuration (object classes, attributes, parts) neglecting the management of so-called resources (Heinrich & Jüngst 1991). Resources, which may be considered as constraints between a certain attribute of all parts of an object, are essential to represent several relations between an object and its parts in a compact way.
2. Knowledge on legal configurations in general and the currently required properties on a configuration characterize a configuration task *not* completely. Additional requirements result from embedding the configuration process in a design or business process and concern for instance user interaction, reaction, maintenance of existing configurations etc. (Meyer auf'm Hofe 1996).
3. More complex configuration tasks typically imply optimization problems — for instance the optimization of costs or utility. Generally, the models for a specification are not equally good. In the *constraint satisfaction* community, this kind of problems is well known (Freuder & Wallace 1992; Schiex, Fargier, & Verfaillie 1995).

This paper sketches knowledge representation aspects and organization of the configuration process according to the *ConStruct* approach. Particularly, problem 1 and 2 will be addressed.

The paper is organized into three sections where section one presents the description of a well-founded object-oriented concept language that enables the specification of resource constraints. Section two describes a model of the configuration process that is derived from the REDUX formalism (Petrie 1991) of concurrent design. Finally, a concluding section is about further work for instance on the integration of soft constraints.

The *ConStruct* Formalism for Configuration Problems

The representation language: The *ConStruct* formalization of configuration problems is based on concept descriptions of object classes. Concept definitions describe the legal configurations in the knowledge base as well as the query for a product that commensurate with several requirements. The examples of Figure 1 follow a *logic through the backdoor* strategy and present concept definitions in a JAVA-like syntax. A concept is defined by the specification of superconcepts, attributes, parts, and constraints. Attributes have a primitive type (like `int` or a finite domain of symbols) that can be handled by constraint solvers whereas parts denote relations to objects which refer to another concept definition. Parts may have a variable cardinality. Attribute values may have to comply with certain constraints which are formulated in an appropriate constraint language. In Figure 1, concept `motorcar` presents the definition of a resource — a constraint between attributes of all objects which are affected by a certain has-part relation (relation `wheels`). In contrast to traditional logical frameworks like *feature logic*, syntax checking should ensure that attributes and parts are only shared within the inheritance hierarchy. This is usual practice in modern object-oriented programming languages.

Declarative semantics: Concepts are, as usual in logics, viewed as a possibly infinite set of standard models. The *ConStruct* formalization uses *feature terms* as sketched in Figure 2 where values are assigned to attribute names and complex objects respectively lists of complex objects are assigned to names of parts. Since *feature terms* are widely accepted as codings for configurations, we will skip their foundation by rules for logical interpretation and conclude to clarify the relation of concept definitions and standard models.

Let Δ be the set of all standard models, which may be

```
class motorcar {
  motor    : Motor;
  wheels   : Wheels[4];
  speed    : {economic, standard, fast};
  weightKG : int;

  speedUpFct : int;

  275 > (max) wheels[].width;
  (=) wheels[].width;

  (speed, speedUpFct) :
    { (economic, 14),
      (standard, 10),
      (fast,      8)  };

  motor.powerPS*speedUpFct < weightKG+200;
  weightKG-200 < motor.powerPS*speedUpFct;
  ...
}

class funcar extends motorcar {
  speed : {standard, fast};
  ...
}

class familycar extends motorcar {
  speed : {economic, standard};
  ...
}
```

Figure 1: An example for concept definitions including attributes, parts, and resources.

composed of feature names which occur in the knowledge base. Then, a concept definition c represents the set $\Delta_c \subseteq \Delta$ according to the following constraints[1]:

1. If C denotes the superconcepts of c then $\Delta_c \subseteq \bigcap_{c' \in C} \Delta_{c'}$.
2. All elements of Δ_c have to define the features that occur in the definition of c.
3. The feature values in an element of Δ_c have to be consistent with the constraints which have been specified on c.
4. Let $\circ$ be an associative and commutative function in the constraint language and the resource constraint $a = (\circ)p[].a'$ be a part of concept definition c where a and a' are attributes and p is a feature representing a part. Then, all elements of Δ_c have to contain $(a = v_a, p = [(a' = v_1, \ldots), \ldots, (a' = v_n, \ldots)])$ in such a way that $v_a = v_1 \circ \ldots \circ v_n$.
5. If c has a non-empty set of subconcepts C then $\Delta_c = \bigcup_{c' \in C} \Delta_{c'}$.
6. If c does not have subconcepts, then the elements of Δ_c define *only* attributes, which occur in the definition of c or its superconcepts.

[1]Of course, a well founded definition of c would refer to the interpretations of the models in Δ_c.

query:

```
extends motorcar { speed=economic; }
```

a model:

```
(wheels     = [(width=205, ...), (width=205, ...),
               (width=205, ...), (width=205, ...)],
 motor      = (powerPS=75, ...),
 speed      = economic,
 speedUpFct= 14,
 weightKG   = 1070,

 ...)
```

Figure 2: A query and a complying model referring to the concepts of Figure 1).

The items 1 to 3 present usual demands on the interpretation of concept definitions. Item 4 describes the meaning of expressions describing resource constraints. Item 5 presents the so-called *cover axiom* that is well known in the area of configuration (Schröder, Möller, & Lutz 1996). The cover axiom guarantees that all instances of an abstract concept implement all properties of a most specific concept definition. Hence, a `motorcar` as defined in Figure 1 is either a `funcar` or a `familycar`.

The *cover axiom* enables the inference via constructive disjunction — a disjunction of constraints in the definition of subconcepts is used to improve the search for an instance of the superconcept. The *ConStruct* formalization of configuration additionally uses the strong *closed world assumption (CWA)* of item 6. All instances of a concept define only features which are either inherited from a superconcept or defined in a subconcept.

Most significant unifiers of concepts: This assumption enables very quick proofs that some concepts are disjointed computing *most general unifiers (MGU)* in order to detect conflicts between requirements on a configuration as early as possible. Figure 3 lists the most important situations in computing the MGU of two concepts.

a) All features of *concept 2* are inherited from *superconcept* and are, thus, also referred by *concept 1*. Hence, there may exist models $\Delta_{concept1} \cap \Delta_{concept2}$ depending on the constraints which are introduced by *concept 1* and *concept 2*. Especially in this case, efficient constraint processing will be valuable in order to proof whether the constraints on both concepts are satisfiable in conjunction or not.

b) In this situation, both concepts introduce new features which are not considered by the other concept. According to the strong CWA in the *ConStruct* formalization, we cannot think of models which belong to both concepts.

c) As in case **b**, both concepts introduce new features. In contrast to case **b**, some concept definitions (*subconcept 1* and *subconcept 2*) are known which inherit from both. One can conclude from the *cover axiom* and CWA, that $\Delta_{concept1} \cap \Delta_{concept2} = \Delta_{subconcept1} \cup \Delta_{subconcept2}$.

Hence, the searched configuration is either an instance of *subconcept 1* or *subconcept 2*.

On queries for a particular configuration: The proposed concept language comprises all basic elements which have been considered as relevant to define legal configurations in the knowledge base. Nevertheless, one still needs the opportunity to specify queries to this knowledge base for legal configurations. The query language shall be able to concern all possibly relevant requirements on a configuration: On the one hand a conjunction of concepts, that the configuration is required to be a model of, and on the other hand a conjunction of constraints on attributes and parts. In contrast to concept definitions in the knowledge base, a query should *not* affect the semantics of any concept definition. Figure 2 illustrates a simple solution to this problem: A query has the syntax of a class definition without a class name. Query expressions are defined according to the items 1 to 4 of the presented semantics without considering cover axiom and CWA.

The *ConStruct* Model for Finding Configurations

Enumerating legal configurations: The enumeration of all possible configurations can be defined by a state transition relation. A state is composed of *design decisions* D and *open design goals* G_{open}, where the first set represents commitments defining a (partial) configuration and the latter set represents what further commitments are needed in order to obtain a legal configuration. Additionally, a state comprises a set of currently valid constraints C on a configuration. Let the tuple $\langle C, \{g\} \cup G_{open}, D\rangle$ define the current state. Then all applicable transition steps generate states $\langle C' \cup C, G' \cup G_{open}, D' \cup D\rangle$ according to one of the following rules:

- Let $g = \mathit{configure}(o : c)$ where o describes an object which is either the root object or a part of the root object and c is a concept definition. Moreover, let $c.C$ be the set of constraints, $c.A$ the set of attributes, and $c.P$ be the set of parts as introduced by concept definition c. If $C \cup c.C$ has a solution then the following states may result from a transition step
  - $D' = \{\mathit{configure}(o : c)\}$.
  - Let G_S include a $\mathit{configure}(o : c_S)$ goal for any superconcept c_S of c without a corresponding $\mathit{configure}(o : c_S)$ decision in D and a $\mathit{specialize}(o, S')$ goal referring to the subconcepts S' of c. Then, $G' = G_S \cup \bigcup_{a \in c.A}\{\mathit{define}(o.a)\} \cup \bigcup_{\langle p,c'\rangle \in c.P}\{\mathit{decompose}(o.p : c', c)\}$.
  - $C' = c.C$.

  Valid configure goals mean that o is required to comply with the definition of c. Decisions of type $\mathit{configure}(o : c)$ state that all following transitions will lead to configurations where object o complies with concept definition c.
- Let $g = \mathit{specialize}(o, S')$, where S' is a set of concept definitions. If C has a solution then a transition step is possible for any $S'' \subseteq S$ with $mgu(S'') \neq \{\}$ resulting in:
  - $D' = \{\mathit{specialize}(o, S'')\}$ with $S'' \subseteq S$.
  - If $mgu(S'') = S''$ then $G' = \bigcup_{c' \in S''}\{\mathit{configure}(o : c')\}$. If S'' has a non-empty but more complex MGU (cf. case **c** in Figure 3) then G' must additionally include a goal $\mathit{specialize}(o, mgu(S'))$.
  - $C' = \{\}$.
- Let v be a value that is in the domain of attribute a. If $g = \mathit{define}(o.a)$ and $C \cup \{(o.a = v)\}$ has a solution then
  - $D' = \{\mathit{define}(o.a, v)\}$.
  - $G' = \{\}$.
  - $C' = \{(o.a = v)\}$.
- Let v be a cardinality of the parts p of object o, that complies with the constraints in C, and $c.R$ the set of resource constraints being defined for concept c.

  If $g = \mathit{decompose}(o.p : c', c)$ then all states of the following kind are results of possible transition steps:
  - $D' = \{\mathit{decompose}(o, p : c', c, v)\}$.
  - $G' = \bigcup_{0 \leq i \leq v}\{\mathit{configure}(o.p[i] : c')\}$.
  - Let $o.p.card$ denote an attribute of object o that determines the cardinality of parts in $o.p$, then $C' = \{(o.p.card = v)\} \cup \bigcup_{(a=(\circ)p[].a') \in c.R}\{(o.a = o.p[0].a' \circ \ldots \circ o.a = o.p[v-1].a')\}$.

The initial state $\langle C, G, \{\}\rangle$ represents the query to the knowledge base where G is the set of conf goals on all required concepts and C is the set of additionally desired constraints. A transition step found a valid configuration if the resulting state has the format $\langle C, \{\}, D\rangle$. D represents a unique feature list which can be constructed from the decisions of type def and $\mathit{decompose}$.

The four kinds of design goals and decisions match perfectly with the common sense in structure-oriented configuration, since resources are integrated as a new kind of constraint affecting a variable set of attributes. In general, the set of legal configurations is non-recursive. However, constraints and resource constraints provide practical opportunities to enforce a finite number of configurations. If, for instance, any part has a cost and there is an upper bound on the overall costs of an object then, obviously, only a finite number of configurations is legal. Configuration problems of this kind are recursive.

Interactive and reactive configuration: The presented state transition is a (very simplified) theory on finding legal configurations in off-line applications. In principle, this model is also appropriate to organize a process of configuration that is guided interactively by an expert. The basic idea is that the expert starts with the initial state and refines it by selections among concluding states with the opportunity to return to a previous state if the remaining opportunities do not comply with the expert's purpose. Frameworks on concurrent engineering like the REDUX model (Petrie 1991)

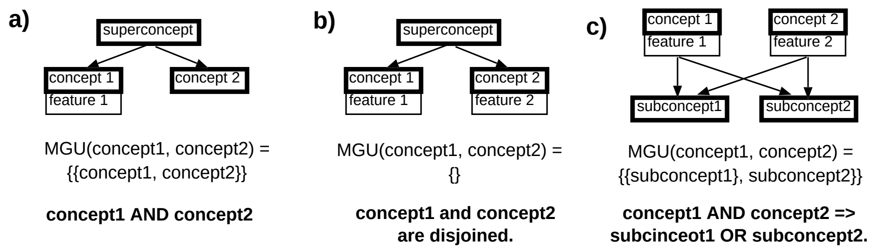
a)
superconcept
concept 1
feature 1
concept 2
MGU(concept1, concept2) = {{concept1, concept2}}
concept1 AND concept2
b)
superconcept
concept 1
feature 1
concept 2
feature 2
MGU(concept1, concept2) = {}
concept1 and concept2 are disjoined.
c)
concept 1
feature 1
concept 2
feature 2
subconcept1
subconcept2
MGU(concept1, concept2) = {{subconcept1}, subconcept2}}
concept1 AND concept2 => subcinceot1 OR subconcept2.

Figure 3: Situations of computing the MGU of two concepts.

interact with experts for mainly two purposes: On the one hand, the coordination of more than one expert designing the same object should be supported. On the other hand, the experts should be able to use their individual expertise as a complement to the knowledge that has been formalized in the knowledge base. Especially the latter point is often of extreme importance since knowledge bases of non-trivial content can never be expected to represent the intended information exhaustively.

However, this purpose requires (at least) two extensions to the simple off-line search for a configuration. On the one hand, the expert should be able to select (temporarily) inconsistent states. As a consequence, a so called set of conflicts is added to the state description. Non-empty set of conflicts indicate for instance, that the currently valid set of constraints is not consistent etc. On the other hand, support of user interaction requires the system to conduct *dependency directed backtracking (DDB)* because the expert shall be enabled to retract any of his or her decisions in such a way that the following state is as similar to the current state as possible. Particularly the latter extension is relevant to many other application scenarios of knowledge-based configurators. DDB enables for instance maintenance of existing configurations or the ability to react on changes in the applicability of parts which are triggered by external events (empty stock, external requirements, etc.).

The basic idea for achieving DDB in configurators is to describe the transition relation by (Boolean) constraints. Consider the following examples: On each state $\langle G_{open}, D\rangle$ let $G_{admissible}$ be the set of goals which are either open — the elements of G_{open} — or have generated a decision in D. Then, obviously, the following constraints hold true:

- If a decision d is valid ($d \in D$) then the corresponding goal g is admissible ($g \in G_{admissible}$). In case of configure goals and decisions this relation is even stronger, because configure goals do not allow alternative decisions: $\mathit{configure}(o : c) \in G_{admissible} \Leftrightarrow \mathit{configure}(o : c) \in D$.
- For each configure, specialize, and define decision d, $d \in D$ is equivalent to the fact that all goals, which are introduced by this decision, are admissible. In case of decompose decisions, this constraint is a little bit weaker.

The idea is now to put such constraints into a constraint solver and post additionally preferences (soft constraints) on the current decisions and goals to remain valid respectively admissible. The resulting problem may either be considered as a *maxSAT* problem that comprises also compulsory clauses or as a *MaxCSP* with soft and hard constraints. A lot of heuristic and exhaustive algorithms are known to compute solutions to these problems. In case of linear inheritance among concepts (each concept has at most one superconcept) the resulting constraint graph will likely to be (nearly) a tree (this statement is not yet proved). On such problems, solutions can be found in time linear to the number of variables (Dechter, Dechter, & Pearl 1990).

A solution to this problem represents a new state that is valid according to the transition relation and induces minimal changes to the current state.

Obviously, the idea of viewing DDB as a constraint problem exhibits strong similarities to constraints on the existence of variables in dynamic constraint satisfaction (Mittal & Falkenhainer 1990), which are used in some constraint-based configuration systems. However, the proposed model of the configuration process has the advantage that design goals and decisions have an obvious relation to the concept language that has been used in the knowledge base. Hence, presentations of intermediate states in the configuration process to the expert should be easier to understand.

How to Support Generation and Maintenance of Knowledge Bases

An extended version of the proposed representation of configuration states by goals and decisions may also be helpful in constructing and maintaining the knowledge base. The following questions illustrate the main problems on constructing a knowledge base:

- Do the currently defined attributes suffice to represent all relevant data on possible configurations and specifications?
- Does the knowledge base allow or prefer unwanted configurations?
- Does the knowledge base neglect desired configurations?

These questions are typically answered investigating sample problems. The developer will have to change the knowledge base according to undesired states in the configuration process on sample problems. In this situation, an extension of the *ConStruct* model with new kinds of decisions, that reflect the current state of the knowledge base, will be helpful. As a consequence, a system for maintaining knowlegde bases gets able to change the configuration state on the current sample problem in such a way that the new state complies with the changes in the knowledge base but retain as many previously stated decisions as possible.

As described in section , the knowledge base consists of the elements concepts, attributes, parts, constraints, and inheritance relations. Hence, a knowledge base can be described by a set of decisions of the following kind:

- *declare_concept(name)* says, that a concept of the given name is in use.
- *declare_attribute(name, concept name)* and *declare_part(name, concept name)* state that concept *concept name* has an attribute respectively a part named *name*.
- *declare_constraint(predicate, attributes, concept name)* means that *predicate* is valid between the listed attributes on each instance of concept *concept name*.
- *declare_inheritance(superconcept, subconcept)* tells *subconcept* to inherit the properties of *superconcept* and notifies *superconcept*, that its semantics changed according to CWA and *cover axiom*.

A change in the knowledge base may, thus, be considered as adding an unknown decision of this kind or removing a valid one.

On verifying a knowledge base by solving sample problems, states of the configuration system now comprise decisions on the currently investigated knowledge base *and* the currently used sample problems. Obviously, the set of admissible goals — and, as a consequence, the set of valid decisions on the sample problem —- depends to the currently investigated knowledge base. For instance, one can only *define* the value of an attribute if this attribute has been *declared* by a concept definition with a valid *configure* decision and so on. Again, Boolean constraints may be used to represent conditions on consistent states.

A system for constructing and maintaining knowledge bases enables the expert to interleave transactions on the knowledge base with verifying transactions on a sample problem. As in normal configuration processes, working on sample problems means choosing an available alternative to a valid goal and retracting valid decisions. Transactions on the knowledge base intend to overcome with unwanted effects on solving the sample problem:

- A valid goal asks to define an attribute that is unknown in the current situation. Possible transactions: Remove the attribute from the concept definition if it is irrelevant or unknown in general. Alternative: Push the attribute into more specific concept definitions that are currently not valid. Occasionally, this requires to define a new concept (because attributes shall not be declared twice).
- The configuration system offers unwanted alternatives for defining an attribute value. Possible transaction: Introduce a new constraint that prunes these alternatives. Occasionally, the expert has to define new attributes in order to inform the constraint.
- Many sample problems reveal that a very special concept is requested frequently. The expert got fed up with selecting the same sequence of specialization decisions again and again. Consequently, he or she decides to *add* a new inheritance relation between the special case and a very general concept. The logical semantics of the knowledge base does not change but the frequently requested special case will be offered earlier as alternative specialization.

These few examples motivate a deeper investigation of integrating transactions on the knowledge base into the process of solving sample problems. However, this idea is currently far from being subject of concrete implementations.

Further Work

This paper sketched a theory on configuration that combines ideas from concept logic, object-orientation, and concurrent engineering. An implementation of *ConStruct* is currently under development at DFKI. However, some theoretical questions are still open.

The *ConStruct* approach as presented in this paper neglects the problem of preference among legal configurations. This problem can be solved extending concept definitions and constraints by evaluation structures as they are known in *constraint satisfaction* (Schiex, Fargier, & Verfaillie 1995). However, a lot of work has to be done regarding the semantics of such evaluation structures since the proposed combination of *cover axiom* and CWA opens for instance several ways to deal with conditional probabilities elegantly and efficiently.

The implementation of the *ConStruct* model raises additional questions. The concurrent management of decisions, goals, *and* constraints on admissible attribute values implies the demand to process the dependencies between these elements consistently. For instance, detection of inconsistencies in the constraint set on attribute values should induce *no-good* markers into the constraint representation of the DDB problem. Vice versa, an integration of approaches on meta-reasoning on constraint problems (Meyer auf'm Hofe 1998) and reasoning on decisions and goals in *ConStruct* would certainly prove to be useful to achieve efficient implementation of configurators.

Finally, interactive configuration requires to spend further effort on methodologies to present open goals, admissible decisions, and strategies for repairing conflicts in a comprehensible way.

References

Baader, F.; Bürckert, H.-J.; Günter, A.; and Nutt, W., eds. 1996. *WRKP–96: Proceedings of the Workshop on Knowledge Representation and Configuration*, number D–96–04 in DFKI Document.

Buchheit, M.; Klein, R.; and Nutt, W. 1994. Configuration as model construction: The constructive problem solvin approach. Technical Report TM-95-01, DFKI.

Dechter, R.; Dechter, A.; and Pearl, J. 1990. Optimization in constraint networks. In Oliver, R. M., and Smith, J. Q., eds., *Influence Diagrams, Belief Nets and Decision Analysis*, 411–425. John Wiley & Sons.

Freuder, E. C., and Wallace, R. J. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58:21–70.

Heinrich, M., and Jüngst, E.-W. 1991. A resource-based paradigm for the configuring of technical systems from modular components. In *CAIA–91: Proc. of the 7th IEEE Conference on AI Applications*.

Mellish, C., ed. 1995. *IJCAI-95. Proceedings, 14th International Joint Conference on Artificial Intelligence*. San Francisco: Morgan Kaufmann.

Meyer auf'm Hofe, H. 1996. What is still to do in order to solve configuration problems in practice? In *(Baader* et al. *1996)*, 25–32.

Meyer auf'm Hofe, H. 1998. Finding regions for local repair in partial constraint satisfaction. In *KI–98: Advances in Artificial Intelligence*, number 1504 in LNAI, 57–68. Springer Verlag.

Mittal, S., and Falkenhainer, B. 1990. Dynamic constraint satisfaction problems. In *AAAI-90: Proceedings of the 10th National Conference on Artificial Intelligence*, 25–32.

Petrie, C. 1991. *Planning and Replanning with Reason Maintenance*. Ph.D. Dissertation, MCC AI Lab, Austin, TX.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued constraint satisfaction problems: Hard and easy problems. In *(Mellish 1995)*, 631–637.

Schröder, C.; Möller, R.; and Lutz, C. 1996. A partial logical reconstruction of PLAKON/ KONWERK. In *(Baader* et al. *1996)*, 55–64.