

# Configuration à Base de Contraintes

## Rapport de Recherche LSIS/2002/010

### Septembre 2002

Hélène Fargier et Laurent Henocque

IRIT

118 Route de Narbonne

31062 Toulouse Cedex

fargier@irit.fr

et

LSIS - UMR CNRS 6168

Domaine Universitaire de Saint-Jérôme

13397 Marseille cedex 20

laurent.henocque@lsis.org

#### Abstract

Configurer consiste à simuler la réalisation d'un produit complexe à partir de composants choisis dans un catalogue de types. L'industrie exploite depuis longtemps des configurateurs, que les progrès de la programmation par contraintes permettent d'envisager sous plusieurs angles. La dimension générative des problèmes de configuration peut être prise en compte au sein d'outils de programmation logique (CC, CLP, Modèles stables), par l'extension du strict cadre CSP (DCSP dynamiques, CSP génératifs, CSP structuraux), ou par des approches orientées objet autonomes ou reposant sur les logiques de description et terminologiques. Cet article tente de présenter un aperçu synthétique des problèmes posés par la configuration, et des approches retenues.

## 1 Introduction

Configurer consiste à simuler la réalisation d'un produit complexe à partir de composants choisis dans un catalogue de types. Simuler la construction consiste à réaliser la structure du produit final en s'appuyant sur les relations connues entre les différents types de composants (choix du nombre et du type de processeurs dans un PC par exemple), et en faisant des choix pour les différents attributs variables (quantité de mémoire, vitesse, etc...). La configuration correspond à un besoin industriel concret qui a fait l'objet de nombreuses implantations dans le passé, avec les technologies disponibles. En particulier, ce fut une des premières applications d'envergure des moteurs de système experts en chaînage avant. Issu de R1[24] le système XCON[3] utilisé pour la configuration de calculateurs chez Digital Equipment comportait en 1989 31000 composants, et de l'ordre de 17000 règles, pour un taux de changement des données de 10 pour cent par an. On connaît ou on envisage des applications de la configuration à de nombreux domaines, dont par exemple la relation client (chez Baan CRM), la vente sur internet (projet CAWICOMS[8]), le logiciel[44, 11], les ordinateurs[29], la configuration de véhicules (chez Renault, Daimler Chrysler), la configuration d'armoires électroniques (chez Edf, Demex) et d'alimentations de moteurs électriques[16] et de nombreuses autres.

La difficulté de maintenir avec une programmation de type déterministe des applications de configuration dont les catalogues de composants possèdent une grande variabilité annuelle conduit depuis quelques années à de nouvelles approches basées sur la programmation par contraintes, la programmation en logique et la programmation objet, utilisées éventuellement de manière conjointe. Le présent article est une tentative d'état de l'art en configuration à base de contraintes. Dans un premier temps (section 2), nous détaillerons les différentes problématiques posées par les applications en configuration. Puis nous présenterons les réponses qu'ont pu apporter ces nouvelles approches : la programmation logique en section 3, la programmation par contraintes en section 4 et la modélisation par objets en section 5.

## 2 Problématiques de la configuration

La configuration met en oeuvre différents types de tâches : la génération du modèle, la configuration proprement dite et un certain nombre de tâches en aval, comme la génération de données techniques, le suivi et la maintenance des configurations. Pour une analyse plus poussée des besoins que posent les problèmes de configuration, voir [42, 1].

### 2.1 Génération et maintenance du modèle

Tout d'abord, il faut être capable de construire, maintenir et tester un modèle du produit configurable (le "produit générique", dont les produits configurés sont des instances) et de son processus de configuration. En ce qui concerne la modélisation proprement dite, il faut tenir compte de plusieurs facteurs. Tout d'abord, les produits configurables sont généralement très structurés, en sous-produits, sous-fonctions, sous-composants, eux-mêmes configurables, et types, comme illustré par la figure 1. D'autre part, les valeurs que peuvent prendre les paramètres de la configuration ne sont pas indépendantes les unes des autres - c'est bien le problème - et il faut pouvoir modéliser, de manière aussi déclarative que possible, ces "contraintes". Enfin, on peut vouloir représenter des connaissances sur le processus de configuration lui-même (que configurer d'abord ? comment orienter les choix de l'utilisateur ? etc.).

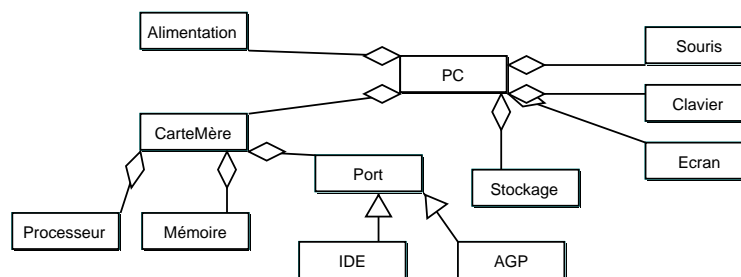


Figure 1: Un modèle simplifié de PC

La phase de validation du modèle est tout aussi importante: les produits configurables nécessitant l'emploi d'un configurateur étant souvent complexes, les modèles les représentant le sont souvent aussi. Parmi les tests que l'on peut mener sur un modèle, on peut citer: l'identification de données "mortes" (composants, sous-produits, fonctions, contraintes n'intervenant dans aucune configuration valide), l'identification d'incohérences, l'identification de produits types. Valider un modèle peut devenir encore plus difficile quand celui-ci dépend explicitement de la sémantique opérationnelle de l'outil logique sous-jacent.

Enfin, au cours de son cycle de vie, le modèle va être modifié. Ceci amènera souvent une augmentation considérable et rapide du volume des données et connaissances liées au produit configurable. Certaines données vont être également modifiées ou supprimées. Afin d'assurer ou de tenter d'assurer que le modèle reste cohérent, on va le tester sur des sessions de configuration type, positives (devant mener à un produit effectif) ou négatives (devant conduire à une incohérence).

### 2.2 La phase de configuration proprement dite

La fonctionnalité essentielle de la configuration est de permettre à l'utilisateur de définir un objet, instance du modèle générique (donc réalisable : à la fin de la configuration, toutes les contraintes doivent être vérifiées) et satisfaisant ses propres besoins.

### 2.2.1 Configuration interactive

Le modèle de session d'utilisation de configurateurs qui émerge est un modèle interactif, notamment du fait du potentiel ouvert par la configuration pour l'intermédiation intelligente sur internet [8]. On considère pouvoir utiliser un configurateur pour intégrer des requêtes de l'utilisateur à la fois non exhaustives, et présentées en ordre dispersé et pour produire de façon incrémentale des solutions satisfaisantes. Plusieurs possibilités surgissent alors dès lors que ces choix sont incompatibles avec le modèle, dont entre autres:

- proposer une explication de l'incohérence, sous la forme d'un ensemble minimal de contraintes qui la causent,
- proposer une ou plusieurs réparations de la requête, sous la forme d'ensembles de contraintes maximaux consistants, ou d'ensembles de contraintes devant être supprimées de la requête,
- retirer des domaines des variables visibles avant la précédente interaction les valeurs incompatibles avec l'existence de solution.

### 2.2.2 Complétion de la configuration et optimisation

Au terme de la phase interactive d'acquisition des besoins et des préférences de l'utilisateur, le problème de configuration est posé sous la forme d'un modèle de produit générique, et d'objectifs à atteindre. Une solution du problème est un produit satisfaisant à la fois les contraintes générales de réalisation et les contraintes et objectifs particuliers de l'utilisateur. Il faut donc, dans cette seconde phase, savoir générer une solution, si possible une solution optimale, ou, si le problème ne possède pas de solution, expliquer pourquoi.

## 2.3 En aval de la configuration

Le processus de configuration proprement dit est souvent suivi d'une phase de génération de données, qui permet d'évaluer l'objet construit - généralement en termes de prix et de délais, mais aussi en termes de nomenclature ou de gamme. Il s'agit alors d'intégrer configuration et production en liant le configurateur au système de gestion des données techniques. Dans de nombreux cas, le produit utile de la configuration est le "bill of materials" (BOM): la fiche listant les éléments à combiner, et les consignes de montage, à destination des ateliers. Enfin, on peut citer les problèmes liés à la gestion des objets configurés (suivi, modification, voire reconfiguration).

## 3 Programmation Logique

Une difficulté des problèmes de configuration tient au fait que même si les solutions cherchées sont finies, on ne peut connaître dans le cas général le nombre de composants mis en jeu dans la solution. En pratique, il est même possible de définir un produit générique dont les seules instances réalisables soient infinies. Le langage de modélisation utilisé pour décrire des problèmes de configuration doit donc posséder une puissance expressive comparable à la logique des prédicats. Le problème posé devient dès lors indécidable dans le cas général, car les contraintes de cardinalité minimum portant sur les relations (par exemple "un ordinateur possède *au moins un* disque dur") ont la même sémantique que les quantifications existentielles.

On entend par contrainte une relation définie existant entre des variables ou inconnues, qui représente une information partielle sur les valeurs pouvant être prises par ces variables. La satisfaisabilité ou l'implication d'ensembles de contraintes peuvent être testées par des algorithmes adaptés. En pratique, les contraintes prises en compte par les systèmes existants consistent en des expressions prises dans des algèbres pour lesquelles on connaît des algorithmes de décision efficaces. Deux grandes approches formelles existent pour compléter le cadre strict de la programmation par contraintes au sein d'un langage permettant de décrire comment des nouvelles contraintes peuvent être créées et combinées: CLP [15] (Constraint Logic Programming) et CCP [31] (Concurrent Constraint Programming), implémenté par la famille des langages *cc* ([10]).

### 3.1 CCP - Concurrent Constraint Programming

*cc* possède une sémantique dénotationnelle bien fondée, car un programme peut être compris comme une formule du premier ordre. La nature compositionnelle de ce langage permet la définition de langages de modélisation spécifiques d'un besoin descriptif particulier. Une application en est le langage CDL (Component Description Language [11]) dédié à la modélisation de machines électromécaniques pour la simulation, l'analyse de productivité et l'ordonnancement. Notamment CDL a été appliqué à la configuration de logiciel pour des machines de reprographie. Dans ce cadre, CDL permet sous une forme lisible par les utilisateurs la description des fonctions de la machine, ainsi que les contraintes devant être satisfaites lors de leur déroulement. Le cadre général de mise en oeuvre de cette solution est appelé "model based computing" [11] car le modèle logique est pris en charge de façon dynamique par la solution logicielle finale, qui devient auto-adaptative.

### 3.2 CLP Constraint Logic Programming

La famille de langages CLP [15] étend le langage Prolog [5] par des contraintes. Le système ConBaCon [16] basé sur le langage CLP CHIP [34] a été utilisé avec succès pour la configuration d'alimentations de très gros moteurs électriques. Ce système met l'accent sur le calcul de reconfigurations les plus pertinentes possibles après modification de la demande initiale, en utilisant des contraintes molles. Le langage Prolog offre naturellement la possibilité de créer dynamiquement de nouveaux objets et les variables associées. Il est ainsi possible de rendre compte du caractère génératif de la configuration. Cependant, la limitation aux clauses de Horn et la sémantique opérationnelle de Prolog (reposant sur le chaînage arrière) introduit les mêmes difficultés de maintenance du modèle de configuration que celle des langages à base de règles (en chaînage avant).

### 3.3 Modèles Stables

Dans le cadre général des approches basées sur la programmation logique et CLP, différents travaux ont été conduits à partir de l'utilisation des modèles stables [12]. Une extension des programmes logiques aux règles contraintes pondérées (weight constraint rules) [37] est utilisée pour représenter des problèmes de configuration. Les pondérations permettent de correctement décrire les contraintes de cardinalité nombreuses dans ces problèmes. L'utilisation de la sémantique des modèles stables est adaptée à la relative polarité des problèmes de configuration, où l'existence d'un composant est souvent rendue nécessaire (justifiée) par l'existence d'un autre muni de propriétés remarquables. Ainsi, un littéral n'est présent dans l'interprétation canonique que si c'est *justifié* par le reste du modèle. Une application de cette approche a été réalisée pour la configuration du système Debian Linux [39]. Toutefois, même si le langage autorise l'utilisation libre de disjonctions et de négations en partie droite de règle comme au sein des littéraux pondérés, il ne s'agit pas d'une utilisation libre de la logique. Un littéral négatif en partie gauche de règle ne vaut pas la même chose qu'un littéral positif en partie droite, et la sémantique des modèles stables doit être prise en compte par le modèle, ce qui en complique la formulation.

## 4 Satisfaction de Contraintes (CSP)

Les progrès réalisés dans l'utilisation d'approches énumératives pour les problèmes SAT/CSP suscitent de nombreux espoirs. Ils permettent de répondre efficacement à plusieurs des besoins de la phase de configuration: le filtrage des domaines en configuration interactive et la génération de solution. En effet, même si d'un point de vue théorique les problèmes de configuration sont par nature indécidables, les cas pratiques de leur mise en oeuvre peuvent être traduits en problèmes propositionnels, "seulement" NP complets ou NP difficiles. Puisque le nombre de composants utilisables dans une solution est limité<sup>1</sup>, on peut ramener la configuration à un problème de recherche de modèles finis, dont la traduction en CSP voire en logique propositionnelle est facile.

Dans ce contexte, un certain nombre de requêtes (énumération, filtrage) peuvent être facilement satisfaites. Mais il faut souvent étendre, enrichir le modèle CSP simple, afin de tenir compte d'autres caractéristiques de la configura-

<sup>1</sup>Une borne max peut être calculée en fonction des données du problème

tion, comme la structuration des domaines, qui appelle une modélisation objet et/ou typée, le calcul d'explications, et la génération dynamique de composants.

#### 4.1 Prise en compte de la dimension générative

Diverses extensions du formalisme CSP ont été proposées pour rendre compte de la dimension générative de la configuration. En effet, une fois traduit en CSP, en considérant que le nombre d'objets pouvant intervenir dans une solution est borné, il reste encore à rendre compte de ceux qui sont ignorés: il faut gérer les variables et contraintes correspondant aux composants non utilisés dans la solution. Différentes approches ont été proposées:

- L'extension des CSP appelée "Dynamic" CSP [26] aborde ce problème en considérant un ensemble de contraintes d'activité, permettant de justifier la présence d'une affectation dans le modèle. Brièvement, dans un DCSP - ou CSP conditionnel-, une variable est active ssi une affectation lui est donnée, une contrainte est active si toutes ses variables sont actives, et une contrainte d'activité déclenche l'activation d'une variable par celle d'une contrainte. Alors que [26] propose une sémantique reposant sur les modèles minimaux, [36] en donne une définition à base de point fixe, et prouve la NP-complétude du problème de décision associé. DCSP, bien que significativement plus expressif, est donc équivalent à CSP et SAT.
- Les CSP génératifs [38, 9] sont une évolution des DCSP permettant de s'affranchir de la donnée initiale de tous les composants pouvant apparaître dans une solution. De surcroît, ils permettent la description de relations entre les composants. Chaque composant intervenant dans un GCSP est à la fois une variable, dont le domaine est l'ensemble de ses types possibles, et un élément possible du domaine des connections. La génération de nouveaux composants est explicite.
- Les CSP composites [30] intègrent ces deux aspects par l'introduction de méta variables dont les domaines représentent des ensembles entiers de sous problèmes. L'affectation d'une méta variable à une valeur transforme le problème courant par l'ajout des variables et des contraintes du sous problème associé. Comparativement aux CSP hiérarchiques et dynamiques, cette approche permet la mise en oeuvre d'algorithmes ne manipulant à tout moment que l'ensemble des variables et des contraintes pertinentes. En contrepartie bien sûr, les possibilités de filtrage par consistance d'arc s'en trouvent réduites. Par ailleurs, cette solution conduit une modélisation du problème qui compile des propriétés qui auraient pu se trouver décrites par des contraintes.
- Les CSP Structuraux, appliqués à la configuration dans [27] diffèrent largement des précédents. Un CSP structurel combine des contraintes au sens habituel, des contraintes extensibles (pouvant dynamiquement intégrer de nouvelles variables), des contraintes "objet" qui ne filtrent pas les domaines mais décrivent un contexte d'aggrégation, et surtout des contraintes structurelles qui filtrent des sous graphes et non des éléments de domaine. Les SCSP sont indécidables.

#### 4.2 CSP, explications de l'incohérence et restauration de la cohérence

Dans une session interactive, produire des explications de l'incompatibilité d'une valeur ou, si la session aboutit à un échec, produire des relaxations maximales cohérentes des spécifications de l'utilisateur, pose des problèmes computationnellement difficiles - ce qui est, on peut le prévoir, peu compatible avec les objectifs d'interactivité requis par une application déployée sur internet. De même, l'utilisation systématique d'un filtrage exhaustif de type "cohérence globale" des choix présentés à un utilisateur à chaque étape d'interaction est en général irréalisable dans ce contexte, en raison de sa complexité intrinsèque (en général NP complète).

Parmi les approches étudiées pour surmonter cette difficulté figure la compilation de l'ensemble des solutions sous la forme d'un automate d'états finis [2] - nous y reviendrons dans la section suivante. Une approche moins coûteuse en espace, mais potentiellement exponentielle en temps serait d'attacher un poids à chaque choix de l'utilisateur et, lorsque le problème s'avère inconsistant, de le considérer comme un CSP valué dont on recherchera une solution optimale - ce que propose [41], dans le cadre général des CSP valués.

En ce qui concerne les explications, une approche efficace [17] consiste à attendre qu'un état d'inconsistance soit détecté par l'algorithme de maintien de la consistance du système (par exemple un algorithme de propagation de

contraintes) : on possède alors un conflit (l'ensemble de contraintes et/ou de choix de l'utilisateur dont le système a détecté l'incohérence), dont on extrait itérativement un sous ensemble minimal incohérent *au sens de l'algorithme de maintien de la cohérence utilisé*. Ce dernier étant sain, on peut garantir que l'explication produite est bien un ensemble incohérent de choix, mais, s'il n'est pas complet, on ne peut pas garantir sa minimalité dans l'absolu. En revanche, il suffit que l'une des contraintes ou des choix de l'explication soit retiré(e) pour que le système ne soit plus en échec *à cause de ce conflit*. Ce principe est utilisé dans l'outil ILOG-Configurator [20].

Enfin, on peut citer des approches de production d'explications par mémorisation partielle de justification [19], à l'œuvre dans le système PALM (associé au langage de programmation par contrainte CHOCO), mais qui n'ont pour l'instant pas encore été appliqués à un problème de configuration particulier.

### 4.3 Compilation

Une des réponses au problème de la configuration interactive est la compilation de l'ensemble des produits configurés (toutes les instances du produit générique) sous une forme facile à manipuler et aussi compacte que possible. Dans [2] comme dans [35], l'idée est de caculer un automate, ou un BDD, dont les chemins représentent exactement les produits configurés, vus comme les solutions d'un CSP ou d'une formule logique. C'est une approche récente de la configuration, empruntée à la validation de modèle (Model Checking) et à la compilation de connaissance, qui permet de répondre efficacement non seulement à la maintenance de la consistance globale et à la production de relaxations maximales cohérentes, mais aussi de traiter des tâches comme la vérification de configuration, la génération ou l'optimisation de configuration, la validation de modèle de produit générique (de manière tout à fait comparable à ce que l'on fait en model checking) comme proposé dans [35]. Un intérêt de cette approche est de facilement permettre la validation de contraintes globales lors de modifications de la base de données. Ce système est en particulier actuellement utilisé commercialement pour tester la validité (constructibilité) d'une commande chez Daimler Chrysler.

On retrouve aussi cette idée de compilation dans les travaux de Weigel et Faltings [43], eux aussi motivés par les problèmes de configuration. L'idée est encore de construire une structure de données, ici l'arbre de synthèse ("Synthesis Tree"), qui permette d'assurer un traitement efficace des requêtes (e.g., la maintenance dynamique de la consistance globale des domaines).

Dans tous ces travaux, la forme compilée permet de répondre aux requêtes en temps polynômial, et souvent linéaire. Cela dit, la complexité spatiale théorique de la structure compilée peut, au pire cas, être exponentielle par rapport à celle des données; mais tous aussi partent du principe qu'en pratique, cette complexité est plus faible: ceci est dû aux caractéristiques particulières des problèmes de configuration, à savoir leur structuration en sous composants et en composants optionnels, et surtout à la présence de nombreuses valeurs interchangeables.

### 4.4 Génération de solution, optimisation

Les progrès réalisés dans l'utilisation d'approches énumératives pour les problèmes SAT/CSP, on l'a vu, ont suscité de nombreuses approches fondées sur une modélisation de type SAT ou CSP, ce qui permet de récupérer les algorithmes puissants et les boîtes à outils proposés dans ces domaines.

Cela dit, certains problèmes peuvent même être traités de façon naturelle par des techniques classiques en recherche opérationnelle, comme la génération de colonnes. Par exemple, le problème du remplissage optimal de colis multiples (Bin Packing/Vellino), peut être résolu en deux phases en calculant les solutions du CSP décrivant les contraintes portant sur un colis, et en générant une colonne par "type" de colis solution. Un programme linéaire simple permet alors de calculer une solution de coût minimal garantissant l'envoi de toutes les pièces.

Les problèmes de calcul de configurations optimales donnent bien sûr lieu aux mêmes extensions que dans le cas des CSP classiques. En particulier, l'optimisation globale multi objectifs conduit à l'identification des solutions dites non Pareto dominées, qui doivent encore être triées au moyen d'une expertise appropriée. De telles approches sont connues en recherche opérationnelle [6], et le besoin d'optimisation multicritères existe en configuration [29]. Une autre approche de l'optimisation, non globale cette fois, consiste à exploiter les heuristiques (formulées sous forme de préférences entre décisions) pour produire une liste de solutions préférées.

Enfin, il faut noter qu'il est tout à fait possible de faire de l'optimisation dans un cadre CSP, via l'utilisation de coûts ou de degrés de priorités associés aux contraintes, comme le propose le cadre de CSP valués [32] : il s'agit alors

de minimiser la somme des poids des contraintes violées, ou leur produits - "MAX-CSP", "CSP probabilistes" - ou de minimiser la plus forte des priorités de contraintes violées - CSP dits "possibilistes". Un raffinement de cette dernière règle a été proposé pour la configuration par [18] : il s'agit d'implémenter efficacement la règle dite du "Discrimin" [4] [7], qui induit un ordre partiel quasitransitif sur les objets à comparer.

## 5 Modélisations Orientées Objet

L'évolution récente des langages de modélisation orientée objet a conduit au standard UML (Unified Modeling Language)<sup>2</sup>, peu connu comme un langage formel particulièrement adapté à la description de contraintes de configuration, bien qu'on puisse noter la part croissante de modèles présentés en UML dans les publications relatives à la configuration. UML combine la description des classes et de leurs relations avec l'expression de contraintes caractérisant les seuls assemblages valides (exprimées avec le langage formel OCL - Object Constraint Language -). UML permet donc la description du modèle - ou produit générique - et des contraintes qui s'y appliquent. Calculer une configuration équivaut à calculer une instance solution d'un modèle soumis à ses contraintes. Le modèle de la figure 1 permet l'expression en OCL de contraintes très complexes, très exactement du type de celles devant être acceptées par un système de configuration. Par exemple "le nombre d'unités de stockage du pc demandant une interface *ide* est inférieur au nombre de ports de type *ide* de la carte mère":

```
Context PC:  
inv:stockage.type->count("ide") <  
carteMère.ports->select(isKindOf(IDE))->size()
```

Il faut noter que l'utilisation d'un langage de modélisation orienté objet pour décrire un problème de configuration n'exige pas que le système de recherche de modèle sous-jacent s'appuie directement sur le modèle. Ainsi, le langage PCML [40] interface un système reposant sur l'utilisation des modèles stables.

### 5.1 CSP hiérarchiques

Une caractéristique des problèmes de configuration est l'existence de hiérarchies de types de composants. La sémantique habituelle de l'héritage est additive: ce qui est vrai d'un type est vrai de tous ses sous types. Les CSP hiérarchiques [21] permettent d'associer aux variables des domaines hiérarchiquement structurés, et adressent donc directement ce problème.

### 5.2 Concepts et Frames

L'approche orientée objet est également déclinée par le langage FPC (Frames, Parts, and Constraints) [22]. Chaque "frame" y représente une classe du modèle, dont les "slots" sont soit descriptifs (des attributs au sens habituel), soit "partonomiques" (par opposition à "taxonomiques") s'ils décrivent la structure, de la même manière que les "ports" décrits plus loin.

### 5.3 Logiques terminologiques et de description

Une approche consiste à exploiter les fonctionnalités de systèmes existants pour la représentation de connaissance reposant sur des hiérarchies de concepts, dans le cadre des logiques terminologiques, étendues par de la programmation par contraintes. [28] applique les langages de concepts à la configuration. On y retrouve les notions d'attributs, de composants, d'héritage et de contraintes. Egalement, le logiciel de configuration EngCon [14], qui repose sur le système KonWerk/PlaKon [13]. Dans ces systèmes, la connaissance est représentée avec un langage à base de frames, proche du concept d'objet, d'ailleurs muni d'une forme dégradée d'héritage multiple appelé "mixin". L'article [33] montre que les constructions ad hoc de KonWerk ont une traduction dans le formalisme des

---

<sup>2</sup><http://www.rational.com/uml>

logiques terminologiques, ce qui en fournit une justification formelle à posteriori. Le système CLASSIC [25] est utilisé avec succès à grande échelle depuis 1990 et a permis de réaliser de très nombreux configureurs.

#### 5.4 Contraintes et modélisation orientée objets

Une approche résolument orientée objet est décrite dans [23] et mise en oeuvre dans l'outil commercial ILOG Configurator [20]. Chaque instance d'un type décrit dans le modèle associe une variable de type dans la lignée des CSP hiérarchiques, des variables attributs correspondant à des variables de CSP classiques à domaines finis (entiers, chaînes de caractères) ou continus (flottants), des variables ensemblistes dites "ports" représentant les relations vers des objets instanciés et des tableaux de variables de cardinalités représentant de relations vers des objets qui sont seulement comptés par leurs types (une réponse aux problèmes de symétrie et aux besoins de mémoire). Les variables ensemblistes de ports et les contraintes qui s'y attachent permettent la génération automatique d'instances pour satisfaire les contraintes de cardinalité minimum. La recherche de solution exploite directement cette structure.

## 6 Conclusion

On l'a vu en section 1, les domaines d'application industriels de la configuration sont nombreux, et il n'est pas besoin de revenir sur l'intérêt pratique des solutions que peuvent apporter les recherches dans ce domaine. D'autre part, l'utilisation des modèles proposés par la configuration à base de contraintes peut dépasser les problèmes de configuration de produit, pour par exemple apporter de nouvelles solutions techniques à des domaines comme :

- la modélisation de connaissance et d'intelligence d'agents: on conçoit que la connaissance d'un agent puisse être modélisée par un langage destiné à la configuration, et que sa compréhension de l'environnement consiste en la construction d'une configuration compatible avec les données provenant de ses capteurs.
- le traitement du langage naturel, aux niveaux lexicaux, syntaxique, sémantique et dialogue. L'utilisation de techniques de configuration peut permettre de dépasser l'utilisation des réseaux sémantiques comme modèle de la connaissance, et de traiter le problème d'analyse syntaxique comme un problème de configuration en propre (la construction d'une forme profonde à partir des données).

Au delà des applications, la configuration est aussi un domaine riche par les problématiques scientifiques qu'il pose. Par exemple, pour ne citer que quelques questions proches de l'IA, on peut étudier sa complexité (existence de classes polynomiales?), les méthodes permettant d'éviter l'exploration d'espaces de recherche redondants (du fait de nombreux isomorphismes mais aussi pour des raisons logiques: coupures, backtracks intelligents), les heuristiques adaptées au calcul de configurations optimales (mono et multi critères), la parallélisation efficace des algorithmes (partage de lemmes par exemple), cette liste étant ouverte.

Enfin, en rapprochant la dynamique actuelle des recherches menées autour de la configuration du succès pratique de la programmation par contraintes, on peut s'interroger sur l'issue du conflit entre les méthodes. Une qualité majeure de la programmation par contraintes est d'offrir un langage à la fois parfaitement déclaratif et facile à comprendre, où le problème est décrit sans tenir compte de la manière dont il sera utilisé (preuve de consistance, recherche de modèle, recherche de solution optimale etc...). Il offre de surcroît un support simple à des heuristiques et algorithmes de coupure d'espaces de recherche inutiles. Si l'on doit projeter ces qualités sur l'ensemble des approches techniques pour la configuration, les meilleurs candidats se trouvent peut être dans le camp des systèmes reposant sur une modélisation orientée objet, à base de frames, ou terminologique, et pour lesquels la procédure de recherche de solution s'appuie directement sur le modèle sans demander de traduction dans un autre formalisme.

## Références

1. Michel Aldanondo, Hélène Fargier, and Mathieu Véron. *Configuration, configureurs et gestion de production*, pages 179–209. Hermes Science, Traité IC2 Productique, 2001.

2. Jérôme Amilhastre, H el ene Fargier, and Pierre Marquis. Consistency restoration and explanations in dynamic CSP - application to configuration. *Artificial Intelligence*, 135(1-2):199–234, 2002.
3. Virginia Barker, Dennis O’Connor and Judith Bachant, and Elliot Soloway. Expert systems for configuration at digital: Xcon and beyond. *Communications of the ACM*, 32:298–318, 1989.
4. Gerard Brewka. Preferred subtheories: An extended logical framework for default reasoning. In *Proceedings of the 11th IJCAI*, pages 1043–1048, Detroit, MI, 1989.
5. Alain Colmerauer. An introduction to prolog-iii. *Communications of ACM*, Vol 33(7), pages 69–90, 1990.
6. Indraneel Das and John E. Dennis. Normal-boundary intersection: A new method for generating pareto optimal points in nonlinear multicriteria optimization problems. *SIAM Journal on Optimization*, 8(3):631–657, 1998.
7. Didier Dubois, H el ene Fargier, and Henri Prade. Refinements of the maximin approach to decision-making in fuzzy environmen. *Fuzzy Sets and Systems*, 81:103–122, 1996.
8. Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. Semantic configuration web services in the cawicoms project. In *Proceedings of the Configuration Workshop, 15th European Conference on Artificial Intelligence*, pages 82–88, Lyon, France, 2002. <http://www.cawicoms.org/>.
9. Gerhard Fleischanderl, Gerhard Friedrich, Alois Haselb ock, Herwig Schreiner, and Markus Stumptner. Configuring large-scale systems with generative constraint satisfaction. *IEEE Intelligent Systems, Special Issue on Configuration*, 13(4), July 1998.
10. Markus Fromherz, Vineet Gupta, and Vijay A. Saraswat. cc – a generic framework for domain specific languages. In *POPL Workshop on Domain Specific Languages, Paris*, pages 89–96, January 1997.
11. Markus P. J. Fromherz, Vijay A. Saraswat, and Daniel G. Bobrow. Model-based computing: Developing flexible machine control software. *Artificial Intelligence*, 114(1-2):157–202, October 1999.
12. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.
13. Andreas G unter and Christian K uhn. Knowledge-based configuration - survey and future directions. In *5th Biannual German Conference on Knowledge Based Systems, W urzburg, Germany, Lecture Notes in Artificial Intelligence LNAI 1570*, pages 47–66, March 1999.
14. Oliver Hollmann, Thomas Wagner, and Andreas Guenter. Engcon - a flexible domain-independent configuration engine. In *Proceedings of 14th European Conference on Artificial Intelligence (ECAI’2000) - Configuration Workshop, Berlin, Germany, 2000*. <http://www.hitec-hh.de/ueberuns/home/aguenter/literatur/ecai2000.pdf>.
15. Joxan Jaffar and Jean Louis Lassez. Constraint logic programming. In *ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.
16. Ulrich John and Ulrich Geske. Reconfiguration of technical products using conbacon. In *Proceedings of AAAI’99-Workshop on Configuration*, pages 48–53, Orlando, Florida, July 1999.
17. Ulrich Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI’01 Workshop on Modelling and Solving problems with constraints (CONS-1)*, pages 81–88, Seattle, WA, USA, August 2001.
18. Ulrich Junker. Preference-based search and multi-criteria optimization. In *Proceedings of AAAI’02*, pages 34–40, 2002.
19. Narendra Jussien and Vincent Barichard. The palm system: explanation-based constraint programming. In *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, pages 118–133, Singapore, September 2000.
20. Olivier Lhomme. Ilog-configurator: Programmation par objet et par contraintes pour la r esolution de probl emes de configuration. In *Actes de JNPC’00*, Marseille, France, June 2000. <http://www.ilog.fr/products/jconfigurator/>.
21. Alan K. Mackworth, Jan Mulder, and William S. Havens. Hierarchical arc consistency: Exploiting structured domains in constraint satisfaction problems. *Computational Intelligence*, 1(3):118–126, 1985.
22. Diego Magro, Pietro Torasso, and Luca Anselma. Problem decomposition in configuration. In *Proceedings of the Configuration Workshop, 15th European Conference on Artificial Intelligence*, pages 50–55, Lyon, France, 2002.
23. Daniel Mailharro. A classification and constraint-based framework for configuration. *AI in Engineering, Design and Manufacturing*, (12), pages 383–397, 1998.

24. John P. McDermott. R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19:39–88, 1982.
25. Deborah L. McGuinness and Jon R. Wright. An industrial-strength description-logics-based configurator platform. *IEEE Intelligent Systems* 13(4), pages 69–77, 1998.
26. Sanjay Mittal and Brian Falkenhainer. Dynamic constraint satisfaction problems. In *Proceedings of AAAI-90*, pages 25–32, Boston, MA, 1990.
27. Alexander Nareyek. Structural constraint satisfaction. In *Papers from the 1999 AAAI Workshop on Configuration, Technical Report, WS-99-0*, pages 76–82. AAAI Press, Menlo Park, California, 1999.
28. Harald Meyer auf'm Hofe. Construct: Combining concept languages with a model of configuration processes. In *Papers from the 1999 AAAI Workshop on Configuration, Technical Report, WS-99-0*, pages 17–22, 1999.
29. Kevin R. Plain. Optimal configuration of logically partitioned computer products. In *Proceedings of the Configuration Workshop, 15th European Conference on Artificial Intelligence*, pages 33–34, Lyon, France, 2002.
30. Daniel Sabin and Eugene C. Freuder. Composite constraint satisfaction. In *Artificial Intelligence and Manufacturing Research Planning Workshop*, pages 153–161, 1996.
31. Vijay A. Saraswat, Martin Rinard, and Prakash Panangaden. Semantic foundations of concurrent constraint programming. In D. B. Skillicorn and D. Talia, editors, *Programming Languages for Parallel Processing*, pages 283–302. IEEE Press, Los Alamitos, 1995.
32. Thomas Schiex, Hélène Fargier, and Gérard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In Chris Mellish, editor, *IJCAI'95: Proceedings International Joint Conference on Artificial Intelligence*, pages 631–637, Montreal, 1995.
33. Carsten Schröder, Ralf Möller, and Carsten Lutz. A partial logical reconstruction of plakon/konwerk. In F. Baader, H.-J. Bürckert, Andreas Günther, and Werner Nutt, editors, *Proceedings of the Workshop on Knowledge Representation and Configuration*, number D-96-04 in DFKI-Memo, pages 55–64, 1996.
34. Helmut Simonis. The chip system and its applications. *Proceedings of the international conference on Constraint Programming CP 95*, pages 643–646, 1995.
35. Carsten Sinz. Knowledge compilation for product configuration. In *Configuration Workshop, 15th European Conference on Artificial Intelligence (ECAI-2002)*, pages 23–26, Lyon, France, July 2002.
36. Timo Soinen, Esther Gelle, and Ilkka Niemela. A fixpoint definition of dynamic constraint satisfaction. In *Proceedings of CP'99*, pages 419–433, 1999.
37. Timo Soinen, Ilkka Niemela, Juha Tiihonen, and Reijo Sulonen. Representing configuration knowledge with weight constraint rules. In *Proceedings of the AAAI Spring Symp. on Answer Set Programming: Towards Efficient and Scalable Knowledge*, pages 195–201, March 2001.
38. Markus Stumptner and Alois Haselböck. A generative constraint formalism for configuration problems. In P. Torasso, editor, *Advances in Artificial Intelligence: Proceedings of the Third Congress of the Italian Association for Artificial Intelligence AI\*IA'93*, pages 302–313. Springer, Berlin, Heidelberg, 1993.
39. Tommi Syrjänen. A rule-based formal model of software configuration. In *Research Report A 55, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Helsinki, Finland, December 1999.*, 1999. <http://citeseer.nj.nec.com/472460.html>.
40. Juha Tiihonen, Timo Soinen, Ilkka Niemela, and Reijo Sulonen. Empirical testing of a weight constraint rule based configurator. In *Proceedings of the Configuration Workshop, 15th European Conference on Artificial Intelligence*, pages 17–22, Lyon, France, 2002.
41. Gerard Verfaillie and Lionel Lobjois. Problèmes incohérents: expliquer l'incohérence, restaurer la cohérence. In *Actes des 5ièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-Complets (JNPC-99)*, pages 111–120, 1999.
42. Mathieu Véron. *Modélisation et résolution du problème de configuration industrielle : utilisation des techniques de satisfaction de contraintes*. PhD thesis, Institut National Polytechnique, Tarbes, November 2001.
43. Rainer Weigel and Boi Faltings. Compiling constraint satisfaction problems. *Artificial Intelligence*, 115(2):257–287, 1999.
44. Katariina Ylinen, Tomi Männistö, and Timo Soinen. Configuring software products with traditional methods - case linux familiar. In *Proceedings of the Configuration Workshop, 15th European Conference on Artificial Intelligence*, pages 5–10, Lyon, France, 2002.