DEPS Studio :

Un environnement intégré de modélisation et de résolution de problèmes de conception de systèmes

Pierre-Alain Yvars[1], Laurent Zimmer[2]

[1] SupMéca, QUARTZ, Saint-Ouen, France
pierre-alain.yvars@supmeca.fr
[2] Dassault Aviation, Saint-Cloud, France
laurent.zimmer@dassault-aviation.com

Résumé

Nous présentons DEPS Studio un environnement intégré de modélisation et de résolution conçu pour spécifier et résoudre des problèmes de conception de systèmes. Cet environnement permet de décrire, mettre au point et résoudre des problèmes modélisés avec le langage DEPS. Celui-ci combine des traits de représentation des connaissances par objets et des traits de spécification de problème de la programmation par contraintes. Les problèmes de conception adressés par DEPS peuvent-être à dominante technique, matérielle, logicielle (systèmes embarqués) ou bien mixtes (systèmes cyber physiques). Après quelques rappels sur le langage de modélisation, nous présentons les différents éléments qui composent l'environnement intégré. Nous détaillons particulièrement l'organisation de la chaîne de compilation qui va de l'édition du modèle jusqu'à la génération du modèle de calcul ainsi que les principales caractéristiques du solveur de programmation par contraintes qui a été développé pour être intégré à cet environnement. Nous terminons par un point sur les développements en cours qui seront intégrés dans la prochaine version.

Mots-clefs : spécification, conception de système, synthèse de système, représentation des connaissances, programmation par contraintes

Abstract

We present DEPS Studio an integrated modeling environment developed for system design problem solving. This environment enables to describe, debug and solve problems which have been modeled with the DEPS language. The design problems addressed by DEPS can be engineering design, hardware or software design or a mix (Cyber Physical System design). DEPS (Design Problem Specification) combines the modeling capabilities of object oriented knowledge representation and the problem solving capabilities of constraint programming. After a survey of the main distinctive features of the language, we put the emphasis on the compiling process from the model edition to the computational model generation. Then we describe the main characteristics of the constraint solver we developed. Finally, we give some information about the current work that will be integrated in the next version of the tool.

Keywords: specification, system design, system synthesis, knowledge modeling, constraint programming

1 Introduction

Acronyme de « Design Problem Specification », le langage DEPS a été initialement développé pour modéliser des problèmes de conception de produits et plus récemment des problèmes de conception de systèmes [1]. L'outil DEPS Studio exploite donc ce langage pour spécifier et résoudre des problèmes de dimensionnement, configuration, déploiement, allocation, vérification ou synthèse de système. Les problèmes de conception adressés par DEPS peuvent-être à dominante technique, matérielle, logicielle (systèmes embarqués) ou bien mixtes (systèmes cyber physiques).
Le point commun de tous ces problèmes est que leur résolution revient à compléter une représentation sous-définie (au sens de sa structure) du système étudié pour obtenir le ou les seuls systèmes qui satisfont l'ensemble des propriétés qui les caractérisent. Ces dernières pouvant provenir soit d'exigences du cahier des charges soit de contraintes physiques ou technologiques.
Ce que nous résumerons par notre slogan :

Résoudre un problème de conception = Compléter un modèle sous-défini

Nous avons affaire à une problématique que nous qualifierons de synthèse par rapport à la problématique plus classique d'analyse. Dans la problématique d'analyse on exploite un modèle de définition (donc bien défini ou complet) du produit ou du système avec des outils d'évaluation qu'on appelle communément des simulateurs. Dans la problématique de synthèse on cherche à obtenir un modèle de définition avec un outil de synthèse.
Ainsi, DEPS Studio est un outil de synthèse en conception de produits ou de systèmes.

2 Le langage DEPS

2.1 Paradigme

DEPS [1, 2] est un langage de modélisation dédié ou DSML (Domain Specific Modeling Language) C'est une combinaison entre un langage de modélisation et un langage de programmation par contraintes. Aux premiers ont été empruntés les traits de structuration et d'abstraction qui permettent de représenter les composants et l'architecture (éventuellement partielle) du système étudié. Aux seconds ont été empruntés les concepts logico-mathématiques nécessaires à la résolution des problèmes de l'ingénieur. Cette combinaison de paradigmes déclaratifs donne un langage lui-même déclaratif qui permet de représenter à la fois l'organisation des systèmes étudiés ainsi que les propriétés qui les régissent [3].

2.2 Langage orienté objet

Le langage DEPS est un langage objet orienté classes. En DEPS une classe est appelée un modèle et une instance de classe un élément. Les classes peuvent être spécialisées par un mécanisme d'héritage.
Un modèle est constitué de constantes, de variables, d'éléments et de propriétés. Les propriétés sont des relations entre constantes et variables. Un modèle peut comporter des arguments qui sont soit des constantes soit des éléments. Passer des constantes en argument d'un modèle permet de le paramétrer. Passer des éléments en argument d'un modèle permet de définir une relation d'agrégation entre ceux-ci. Cela permet aussi de définir des propriétés entre leurs variables. Dans ce dernier cas un modèle peut représenter une relation générale entre des éléments. A noter que dans DEPS le polymorphisme des modèles tient compte du nombre et du type des arguments.

2.3 Programmation par contraintes et ontologie

DEPS emprunte les aspects déclaratifs du modèle de représentation des problèmes de satisfaction de contraintes (CSP) [4], le modèle <V, D, C> (Variables, Domaines, Contraintes) mais les adapte pour représenter spécifiquement les problèmes de l'ingénieur.
En DEPS, on distingue donc les constantes et les variables. Les deux ont un domaine de définition mais les premières ont une valeur fixée dans leur domaine tandis que les secondes ont un domaine de définition non réduit à une valeur qui caractérise le caractère sous-défini du modèle auquel elles appartiennent.
Les domaines sont soit discrets à valeurs entières ou réelles soit continus à valeurs réelles.
Les constantes et les variables sont elles aussi à valeurs entières ou réelles mais dès lors qu'elles représentent des grandeurs physiques ou technologiques elles peuvent être typées par leurs grandeurs que l'on appelle des quantités.
Une quantité comporte :
- un type de quantité de base ; par exemple une longueur,
- une borne min (respect. max) qui représente la valeur minimale (respect. maximale) autorisée,
- une dimension qui représente la dimension au sens de l'analyse dimensionnelle de la quantité,
- une unité de la quantité ; par exemple le mètre m pour une longueur.

Les quantités, les dimensions et les unités constituent une ontologie du domaine de l'ingénieur.
Les propriétés d'un modèle sont l'équivalent des contraintes dans un CSP. Dans les sciences de l'ingénieur elles sont souvent définies en intension par des équations ou des inéquations algébriques qui relient des constantes et des variables. C'est le cas dans la version actuelle du langage mais toutes les autres formes de définition qu'on retrouve en programmation par contraintes regroupées sous l'étiquette de « contraintes globales » seront permises ultérieurement.

3 DEPS Studio

3.1 L'Environnement

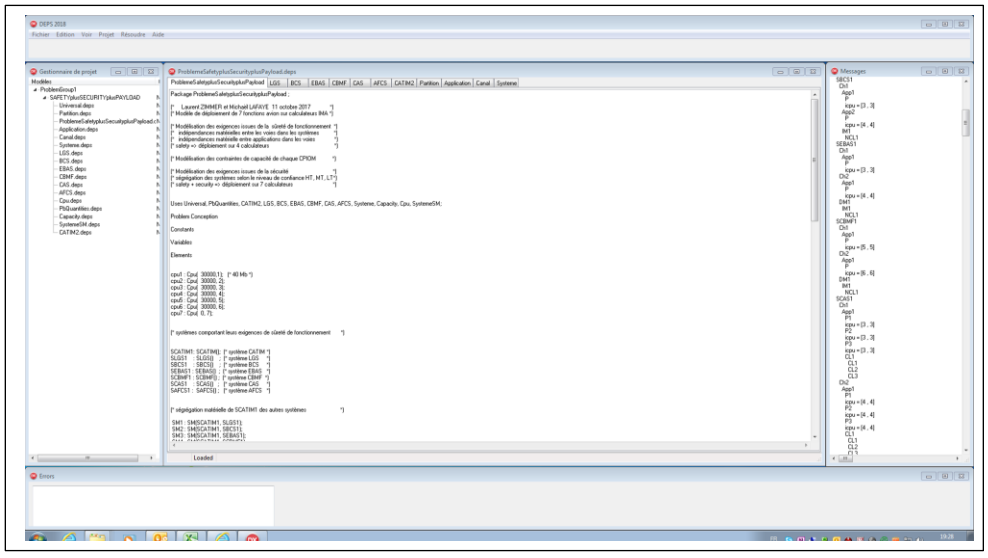

Figure 1 : L'environnement DEPS Studio

L'environnement de modélisation et de résolution intégré associé au langage DEPS comprend des fonctions d'édition de modèle et de gestion de projet, un compilateur et un solveur. L'expérience montre que la spécification d'un problème de conception de système n'est jamais bonne du premier coup et que beaucoup d'erreurs de modélisation ne sont détectables que par le calcul. Nous avons donc décidé de développer et d'intégrer notre propre solveur dans l'environnement de développement pour que la résolution contribue efficacement à la mise au point du modèle du problème. On se place ainsi dans une approche de développement rapide de modèles (analogue à une approche RAD) qui contrairement à une approche par transformation de modèle, diminue le temps d'exécution de la boucle de mise au point du modèle. Elle permet aussi de remonter les erreurs au bon niveau d'abstraction qui est celui de la modélisation.

3.2 Edition de modèles et gestion de projet

Un problème à résoudre est organisé en projet. Un projet est constitué de plusieurs packages. Chaque package est sauvegardé dans un fichier. Les packages contiennent des modèles, des éléments d'autres modèles et à la déclaration des constantes, des variables d'autres modèles.
Un des packages doit contenir un modèle particulier sans argument déclaré comme problème. Ce modèle représente le problème à résoudre exprimé sous forme de constantes, de variables, de relations et d'éléments.
L'environnement dispose :

- d'un éditeur multi-modèle pour charger, modifier et sauvegarder des packages,
- d'un gestionnaire de projet pour charger, modifier et sauvegarder le projet de modélisation d'un problème constitué de l'ensemble de ses packages.

3.3 Le compilateur

Le compilateur que nous avons développé transforme directement la modèle « source » DEPS d'un problème de conception en un réseau de contraintes « objet » associé au modèle <V, D, C>. C'est donc un compilateur « natif » qui n'est pas une surcouche d'un langage de programmation par contraintes. La compilation est anticipée ; tout le réseau est donc généré avant la résolution.
Le typage statique du langage DEPS est exploité par le compilateur pour détecter les erreurs de type des éléments et les erreurs de grandeur des constantes et des variables avant la résolution.
La compilation se fait en deux passes :

- la première passe de compilation contrôle les packages utilisés par le projet, analyse lexicalement et syntaxiquement leur contenu et crée la hiérarchie des modèles du projet ;
- La deuxième passe crée l'ensemble des éléments qui définissent le problème en partant de la création de l'unique élément instance du modèle déclaré comme problème.

Des erreurs sont traitées et remontées à l'utilisateur à toute les étapes de la compilation : contrôle des packages, analyse lexicale, analyse syntaxique, création de la hiérarchie des modèles, création des éléments sous-définis (avec des variables dont le domaine n'est pas réduit à une valeur).
Si l'étape de compilation réussit alors l'étape de résolution aura la tâche d'affecter des valeurs aux variables satisfaisant l'ensemble des propriétés/contraintes du problème.

3.4 Le solveur

Résoudre un problème de conception nécessite de disposer de capacités de résolution permettant de prendre en compte :

- Des problèmes sous contraints
- Des équations et inéquations algébriques non linéaires sur des domaines mixtes
- D'autres types de relations telles que des tables de valeurs, des relations logiques …

Pour ce faire, nous avons développé un solveur à base de contraintes dédié au calcul sur des modèles DEPS structurés. Les méthodes de calcul que nous utilisons sont tirées des travaux sur la résolution des CSP. La structure des modèles DEPS est conservée tout au long de la chaine de compilation jusque dans les modèles de calcul.
Le solveur implémente une méthode de propagation de type HC4 révisé [5] sur des équations et inéquations. Initialement prévue pour des domaines continus, nous avons étendu la méthode à quatre types de domaines : les intervalles ouverts de réels, les intervalles d'entiers, les ensembles énumérés de valeurs flottantes et les ensembles énumérés de valeurs entières signées. Les contractions sont réalisées directement sur les domaines typés sans repasser dans les intervalles de réels. L'algorithme de recherche de solution est une méthode de branch and prune. Les stratégies round-robin et first-fail sont disponibles.
Dans le cas d'un problème sur-contraint, un échec peut apparaitre dès la première propagation ou bien au final après avoir exploré les parties restantes de l'arbre de recherche. Dans ce cas, l'échec s'interprète comme la preuve qu'il n'y a pas de solution au problème posé et non pas comme une défaillance de l'algorithme de résolution.
L'architecture orientée-objet du solveur a été pensée de manière à pouvoir être étendue à d'autres méthodes de propagation et/ou de résolution (box-consistance, méthodes locales, …).

4 Conclusion et perspectives

Nous avons présenté dans ce papier DEPS Studio l'environnement de modélisation et de résolution de problèmes de conception de systèmes exprimés en DEPS. Dans sa version actuelle, cet environnement intègre un nouveau compilateur natif produisant directement un modèle de résolution et un solveur dédié dont les caractéristiques répondent aux problèmes rencontrés en ingénierie des systèmes. L'association qui en résulte est actuellement en cours d'évaluation sur des problèmes de synthèse d'architecture système.
Les développements en cours portent sur la prise en charge d'évolutions importantes du langage DEPS avec notamment l'introduction de collecteurs d'éléments ainsi que la définition de propriétés définies en extension. Ces développements seront disponibles dans une prochaine version de DEPS Studio.
L'association DEPSLink [2] supporte le développement et fait la promotion du langage DEPS.

Bibliographie

[1] P.A. Yvars, L. Zimmer, *"DEPS Un langage pour la spécification de problèmes de conception de Systèmes"*, proc of the 10th International Conference on Modeling, Optimization & SIMulation (MOSIM 2014), France.
[2] www.depslink.com
[3] L.Zimmer, M. Lafaye, P.A. Yvars, "Modélisation d'exigences pour la synthèse d'architecture avionique : Application à la sûreté de fonctionnement", 16eme journées AFADL, Approche Formelle dans l'Assistance au Développement de Logiciel, Montpellier, 2017.
[4] E. Tsang, Foundations of Constraint Satisfaction.London and San Diego:Academic Press, 1993.
[5] Benhamou F., Goualard F., Granvilliers L., Puget J.F., Revising Hull and Box consistency,16th International Conference on Logic Programming, 1993.