DEPS Studio : Un environnement intégré de modélisation et de résolution pour la synthèse de système

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

[1]ISAE-Supméca, laboratoire QUARTZ EA7393, Saint Ouen, pierre-alain.yvars@supmeca.fr
[2]Dassault Aviation - Direction Générale Technique, Saint Cloud, laurent.zimmer@dassault-aviation.com

1 Introduction

Le langage DEPS (Design Problem Specification) a été développé pour modéliser à l'origine les problèmes de conception de produits et plus récemment les problèmes de conception de système [1]. DEPS Studio est l'IDE de ce langage. Il est conçu pour spécifier et résoudre des problèmes de conception, de configuration, de déploiement, d'allocation, de vérification ou de synthèse de systèmes. Les systèmes considérés peuvent être des systèmes physiques, des systèmes à forte composante logicielle ou des systèmes mixtes (embarqués, mécatroniques, cyber-physiques). Le point commun de tous ces problèmes est que leur résolution revient à compléter une représentation sous-définie (ou partielle) du système étudié afin de n'obtenir que les systèmes satisfaisant toutes les propriétés qui les caractérisent. Ces propriétés proviennent soit des exigences du cahier des charges, soit des contraintes physiques ou technologiques. Nous résumons cela par le manifeste suivant : "Résoudre un problème de conception = Compléter un modèle sous-défini". Nous cherchons à obtenir un modèle de solution avec un outil de synthèse. DEPS Studio est l'outil de synthèse que nous proposons et dont nous présentons les caractéristiques principales dans ce papier.

2 Aperçu du langage DEPS

2.1 Paradigme

DEPS [1] est un langage de modélisation dédié ou DSML (Domain Specific Modeling Language). Son objectif est de fournir un formalisme permettant de représenter des problèmes de conception de systèmes techniques. Le lecteur intéressé pourra trouver dans [1] et dans [3] un état de l'art détaillé justifiant la nécessité d'un tel formalisme. Sa grammaire non contextuelle est décrite sous la forme de Backus-Naur. DEPS est né d'une initiative de recherche académique et industrielle et est actuellement supporté par l'association DEPS Link [2]. Il combine certaines caractéristiques de modélisation structurelle propres aux langages orientés objet avec des caractéristiques de spécification de problèmes issues de la programmation par contraintes. Aux premiers ont été empruntées les caractéristiques 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 aboutit à un langage déclaratif qui permet de modéliser l'organisation des systèmes étudiés et les propriétés qui les régissent.

2.2 Types de données

Les types de données DEPS de base utilisés pour le calcul sont : les valeurs entières, les réels, les intervalles entiers, les intervalles réels, les domaines énumérés entiers et les domaines énumérés réels.

2.3 Type de quantité et quantité

Dans DEPS, les constantes et les variables sont associées à des quantités et des types de quantités physiques ou technologiques appelés respectivement *Quantity* et *QuantityKind* qui sont nécessaires dans l'ingénierie des systèmes. Un*QuantityKind* comporte : un type de base (entier ou réel), une valeur minimale, une valeur maximale ainsi qu'une dimension au sens de l'analyse dimensionnelle de la quantité (par exemple L pour une longueur ou U pour une quantité sans dimension). Une *Quantity* est définie à partir d'un *QuantityKind*. Le *QuantityKind* porte la dimension, la *Quantity* porte l'unité. Un *QuantityKind* et une *Quantity* peuvent avoir le même nom. Plus précisément, une *Quantity* comporte : un *Kind*, type de quantité de base (Par exemple, Real,

Integer, Length) et une borne Min (resp. Max) qui représente la valeur minimale (resp. maximale) qui peut être prise par toute constante ou variable ayant la quantité définie comme type, une unité de la quantité.

2.4 Problème et modèles

La caractéristique fondamentale du langage est le modèle. Tout modèle est défini à l'aide du mot-clé *Model* suivi de son nom et de sa liste (éventuellement vide) d'arguments. Il contient dans l'ordre : une zone de déclaration-définition des constantes du modèle (mot-clé *Constants*), une zone de déclaration des variables du modèle (mot-clé *Variables*), une zone de déclaration-définition des éléments du modèle (mot-clé *Elements*) et une zone de définition des propriétés du modèle (mot-clé *Properties*). La définition d'un modèle DEPS se termine par le mot-clé *End*. Le problème à résoudre est exprimé à l'aide du mot-clé *Problem*. Le problème est un modèle sans arguments. Le problème est la racine de l'arbre des éléments qui sont des instances de modèles. Une constante est une quantité numérique dont la valeur ne varie pas pendant la durée de vie d'une copie du modèle dans lequel elle est déclarée et définie. Une variable est une inconnue du modèle. Elle est caractérisée par sa quantité éventuellement limitée à un sous-ensemble de valeurs possibles. Les variables portent le caractère sous-défini des modèles DEPS.

2.5 Fonctionnalités orientées objet

Les modèles DEPS peuvent hériter les uns des autres (mot-clé *extends*). Il s'agit d'un héritage public et simple : les constantes, les variables, les éléments et les propriétés sont ainsi hérités des modèles ancêtres. Un élément est une instance de modèle. Il peut être passé en argument à un Modèle pour représenter une agrégation et doit alors être déclaré dans la zone de champ *Elements* du Modèle. Il peut également être créé dans un modèle pour modéliser une composition et doit alors être déclaré et créé dans la zone des éléments du modèle. Tous les éléments d'un problème sont organisés en utilisant des relations d'agrégation et de composition formant une structure arborescente. L'accès aux éléments de cette structure est autorisé par l'utilisation d'une notation pointée. Une constante, une variable ou un élément à différents niveaux de cette structure arborescente peut être désigné et manipulé en utilisant un chemin. Chaque Modèle possède une signature. Ce mécanisme lève toute ambiguïté lorsque des modèles portent le même nom mais ont des arguments différents.

2.6 Les propriétés

Une propriété est une relation nécessairement respectée par toute instance du modèle qui la contient. Dans la version actuelle de DEPS, les propriétés sont des relations algébriques (ou contraintes) :égalité et/ou inégalité entre expressions, définition de la valeur d'une expression déclarée relative à des constantes et variables du modèle ou d'un ou plusieurs éléments du modèle. Tous les opérateurs de la norme IEEE754 pour l'arithmétique à virgule flottante sont disponibles. Une expression déclarée ou nommée (mot-clé *expr*) pointe vers une expression algébrique et permet de la référencer et de l'utiliser dans de nombreuses propriétés. Les équations et les inéquations linéaires ou non linéaires, sont naturellement des propriétés. Elles peuvent aussi être des relations dédiées au domaine de la conception. Ce sera par exemple le cas pour la contrainte catalogue qui permet à l'utilisateur de créer des relations définies en extension par une table de n-uplets.

3 DEPS Studio IDE

3.1 Généralités

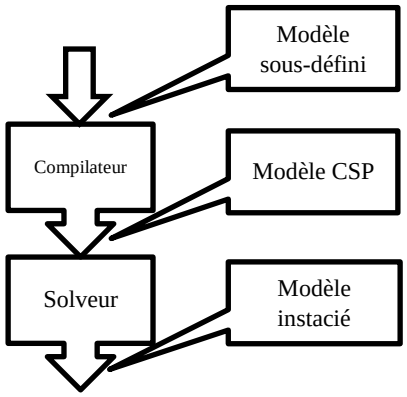

Figure 1: DEPS Studio - chaîne de synthèse

L'environnement intégré de modélisation et de résolution DEPS Studio [3] associé au langage DEPS comprend un éditeur de modèle, un gestionnaire de projet, un compilateur et un solveur (cf. Figure 1). L'expérience montre que la spécification d'un problème de conception de système n'est jamais correcte du premier coup et que de nombreuses 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 afin que la recherche de solutions contribue efficacement à l'ajustement du processus de modélisation du problème. Il s'agit d'une approche de développement rapide de modèles (analogue à une approche RAD) qui, contrairement à une approche de transformation de modèles, réduit le temps d'exécution de la boucle de développement de modèles. Elle permet également de remonter les erreurs au bon niveau d'abstraction. DEPS Studio a été développé en Delphi. Les couches de base du calcul d'intervalle ont été écrites en C++ et utilisent la bibliothèque open source de calcul par intervalle gaol [4]. L'ensemble des développements représente environ 100 000 lignes de code.

3.2 Edition du modèle et gestion du projet

Un problème à résoudre est organisé en un projet. Un projet est constitué de plusieurs paquets (*package*). Chaque paquet est enregistré dans un fichier ".deps". Les paquets contiennent des modèles, des types de quantités, des quantités et des tableaux. L'un des paquets doit contenir un modèle particulier sans arguments et déclaré comme étant le problème. Ce modèle représente le problème global à résoudre exprimé sous forme de constantes, variables, éléments et propriétés.
L'environnement dispose :
- d'un éditeur multi-modèles pour charger, modifier et sauvegarder les paquets,
- d'un gestionnaire de projet pour charger, modifier et sauvegarder le projet de modélisation d'un problème composé de tous ses paquets.

Un projet est défini comme un ensemble de *packages*. Chaque *package* suit la structure suivante :

Package <packageName> ;
Uses <ListOfPackageName> ;
<List of DEPSFeature>
With
<DEPS Feature> : <QuantityKind>
| *<Quantity>*
| *<Table>*
| *<Model>*
| *<Problem>*

3.3 Le compilateur

Le compilateur que nous avons développé transforme directement le modèle "source" DEPS d'un problème de conception en un réseau de propriétés "objet" associées au modèle <V, D, C> d'un problème de satisfaction de contraintes (CSP) [5]. Il s'agit donc d'un compilateur "natif" qui n'est pas une surcouche d'un langage de programmation par contraintes. La compilation est anticipée ; l'ensemble du réseau est donc généré avant la résolution. Le typage statique du langage DEPS est exploité par le compilateur pour détecter les erreurs de type sur les constantes, variables, éléments et propriétés avant la résolution. La compilation se fait en deux passes :
- La première passe de compilation vérifie les paquets 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 à partir de la création de l'élément d'instance unique du modèle déclaré comme problème. Les erreurs sont traitées et signalées à l'utilisateur à toutes les étapes de la compilation : vérification du paquetage, analyse lexicale, analyse syntaxique, création de la hiérarchie du modèle, création des éléments sous-définis.

Si l'étape de compilation réussit, l'étape de résolution aura pour tâche d'attribuer des valeurs aux variables satisfaisant toutes les propriétés/contraintes du problème. L'intérêt d'une approche intégrée et maitrisée noous permet de faire évoluer en parallèle les traits du langage DEPS, son compilateur et le sover.

3.4 Le solveur

La résolution d'un problème de conception nécessite la capacité de prendre en compte les problèmes sous-contraints, les équations et inégalités algébriques non linéaires sur des domaines mixtes ainsi que d'autres types de relations telles que des tables de valeurs. Les problèmes de conception que nous adressons nécessitent à la fois de pouvoir travailler sur des inconnues à domaine de valeurs de type intervalles continus, intervalles d'entiers ou bien des énumérés (entiers ou réels) et de poser des propriétés logique et/ou algébriques sur ces inconnues. Un solveur SAT ou SMT ou bien un solveur purement discret ou purement continu ne seront donc pas suffisants. Nous avons ainsi développé un solveur mixte à base de contraintes dédié au calcul sur des modèles DEPS structurés. Les méthodes de calcul que nous utilisons sont issues des techniques de résolution de CSP [5]. La structure des modèles DEPS est préservée tout au long de la chaîne de compilation jusqu'aux modèles de calcul. Le solveur implémente une méthode de propagation HC4 révisée [6] sur les équations et les inégalités. Initialement conçue pour les domaines continus, nous avons étendu la méthode à quatre types de domaines : intervalles réels ouverts, intervalles entiers, ensembles énumérés de valeurs flottantes et ensembles énumérés de valeurs entières signées. Pour des raisons de performance, les réductions sont effectuées directement sur les domaines typés sans revenir aux intervalles réels. L'algorithme de recherche de la racine est une méthode de branchement et d'élagage. Pour l'instant, seules les stratégies classiques round-robin et first-fail sont implémentées. Dans le cas d'un problème sur-contraint, un échec peut survenir lors de la première propagation ou après avoir exploré l'ensemble de l'arbre de recherche. Suivant le paradigme CSP, l'échec est interprété comme la preuve qu'il n'existe pas de solution au problème et non comme un échec de l'algorithme de résolution. L'architecture orientée objet du solveur a été conçue de manière à pouvoir être étendue à d'autres méthodes de propagation et/ou de résolution existantes (box-consistance, méthodes locales, ...).

4 Conclusion

DEPS Studio a été utilisé sur plusieurs applications industrielles. Il a été notamment appliqué à des problèmes de dimensionnement de système électriques [7] ainsi qu'à des problèmes de déploiement de fonctions sur des architectures informatiques embarquées [8-10]. Les résultats obtenus sont positifs en modélisation comme en résolution. Les futures versions de DEPS Studio tireront parti des prochaines évolutions du langage DEPS.

Réferences

[1] P.A. Yvars, L. Zimmer, 2014. *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] P.A. Yvars, L. Zimmer, *Integration of Constraint Programming and Model-Based Approach for System Synthesis*, IEEE International Systems Conference, SYSCON 2021, Vancouver, Canada

[4] Goualard F., 2015. *Gaol 4.2.0 Not Just Another Interval Arithmetic Library*, https://frederic.goualard.net/software/gaol-4.2.pdf

[5] E. Tsang, 1993. *Foundations of Constraint Satisfaction*. London and San Diego:Academic Press.

[6] Benhamou F., Goualard F., Granvilliers L., Puget J.F., 1993. Revising Hull and Box consistency,*16th International Conference on Logic Programming.*.

[7] S. Diampovesa, A. Hubert, P.A. Yvars, *Designing Physical Systems through a Model-Based Synthesis Approach. Example of a Li-ion Battery for Electrical Vehicles*, Computers In Industry Journal, vol 129, 2021

[8] L.Zimmer, M. Lafaye, P.A. Yvars, *Modélisation d'exigences pour la synthèse d'architecture avionique : Application à la sûreté de fonctionnement*, 16ème journées AFADL, Approche Formelle dans l'Assistance au Développement de Logiciel, Montpellier, 2017.

[9] L. Zimmer, P.A. Yvars, M. Lafaye, *Models of requirements for avionics architecture synthesis: safety, capacity and security*, Complex System Design and Management conference – CSD&M 2020, December 2020, Paris, France.

[10] L. Zimmer, P.A. Yvars, *Synthesis of software architecture for the control of embedded electrical generation and distribution system for aircraft under safety constraints: The case of simple failures*, 14th International Conference of Industrial Engineering, CIGI-QUALITA 2021, Grenoble, France.