Université Fédérale

Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut Supérieur de l'Aéronautique et de l'Espace

Présentée et soutenue par :
Patrick LESERF

le mardi 2 mai 2017

Titre :

Optimisation de l'architecture de systèmes embarqués par une approche basée modèle

---

École doctorale et discipline ou spécialité :

ED MITT : Réseaux, télécom, système et architecture

Unité de recherche :

Équipe d'accueil ISAE-ONERA MOIS

Directeur(s) de Thèse :

M. Pierre DE SAQUI-SANNES (directeur de thèse)
M. Jérôme HUGUES (co-directeur de thèse)

Jury :

M. Jean-Michel BRUEL, Professeur Université Toulouse II - Président
M. Ludovic APVRILLE, Maître de Conférences Télécom ParisTech - Rapporteur
Mme Maryline CHETTO, Professeur Université Nantes
M. Pierre DE SAQUI-SANNES, Professeur ISAE-SUPAERO - Directeur de thèse
M. Jérôme HUGUES, Professeur associé ISAE-SUPAERO - Co-directeur de thèse
M. Chokri MRAIDHA, Ingénieur de Recherche CEA - Rapporteur

Remerciements

Le travail présenté dans ce mémoire a été réalisé au centre de recherche ESTACA'LAB, au sein du pôle Systèmes et Energie Embarqués pour les Transports (S2ET) à Laval (Mayenne).

Je remercie Pascal BIDAN, directeur de l'ESTACA de m'avoir permis de réaliser ces travaux, en adaptant ma charge d'enseignement et me permettant de me déplacer régulièrement à l'ISAE-SUPAERO de Toulouse ainsi qu'aux conférences.

Je remercie la direction de l'ISAE-SUPAERO ainsi que Alain HAÏT, directeur du département DISC pour m'avoir accueilli à chacun de mes déplacements à Toulouse.

Ce travail a été dirigé par Pierre DE SAQUI SANNES, Jérôme HUGUES ainsi que Khaled CHAABAN (pour la première année). J'exprime à tous les trois ma gratitude pour leurs conseils avisés et leur investissement dans l'encadrement de ces travaux.

Je remercie l'ensemble des membres du pôle S2ET de l'ESTACA sous la direction de Chérif LAROUCI, permanents et doctorants, que je n'ai pas manqué de solliciter à plusieurs reprises, et particulièrement Bertrand BARBEDETTE, Fouad KHENFRI, Sébastien SAUDRAIS, Judicael AUBRY, Nassim RIZOUG ainsi que Mahmoud QAFSAOUI (enseignant), Frédéric AMEDEMEGNAH (stagiaire), Hussein Mhanna (All4TEC).

Le dernier mot s'adresse à ma famille, ma femme Nathalie, mes enfants Louis et Emilie pour leur soutien à mes côtés.

Patrick LESERF

Table des matières

Introduction

L'International Council on Systems Engineering ou INCOSE (INCOSE, 2007) définit un système comme un ensemble intégré d'éléments chargés d'accomplir un objectif défini. Les systèmes actuels utilisés dans les transports et notamment les systèmes embarqués (caméra dans un véhicule, système de pilotage d'un drone) ont évolué fortement ces dernières années, en terme de complexité, de connectivité et de performance. Pour rester compétitif, ces systèmes complexes doivent intégrer toujours plus de fonctions, à un coût acceptable. Il s'agit là d'objectifs « externes » au système lui-même. Un second défi concerne les constituants du système lui-même et son architecture, laquelle intègre des éléments mécaniques, optiques, électroniques (HW), logiciels (SW). Pour développer un système complexe, l'ingénierie système ou *IS* propose une approche multidisciplinaire permettant de développer des solutions « équilibrées » en réponses à de multiples exigences. L'ingénierie système s'est développée dans l'aéronautique, l'automobile, et la défense et a pour but :

1) La définition du système, et la documentation des exigences dans les phases amont du développement.
2) La conception du système et la vérification du système par rapport à ses exigences.

Plusieurs processus, méthodes et outils existent pour décrire et mettre en œuvre les activités d'ingénierie système. Ces processus définis par la norme ISO 15 288 (ISO-15288, 2008) ou le modèle CMMI pour Capability Maturity Model Integration (Dufay, 2010) définissent les activités en terme d'objectif à atteindre tandis que les méthodes d'IS décrivent les techniques de réalisation de ces activités. Ces méthodes s'appuient généralement sur une approche documentaire.

Mais pour les systèmes complexes, de nouvelles approches sont nécessaires. Il faut en effet s'appuyer sur des représentations du problème et des solutions, à différents niveaux d'abstraction, pour conceptualiser, concevoir, estimer et valider les choix. Cela est rendu possible grâce à l'ingénierie système basée sur les modèles ou méthodes *MBSE* (Wymore, 1993) comme la méthode orientée objet OOSEM, (Friedenthal, 2007) ou ARCADIA développée par Thalès depuis 2007 (Voirin, et al., 2015). Un premier résultat des méthodes *MBSE* est un modèle cohérent du système à développer, dans lequel les liens sont plus faciles à établir entre les composants du système. Ce modèle permet de générer la documentation, ainsi que tout ou partie d'un composant logiciel. Les méthodes *MBSE* trouvent leurs origines dans d'autres disciplines comme la mécanique, l'électronique et l'informatique où l'utilisation des modèles est devenue courante. Les méthodes *MBSE* sont également liées à un langage de modélisation. Dans le domaine du développement logiciel, des langages comme UML (Rumbaugh, 2004) se sont développés depuis les années 1990 et offrent les bases d'autres langages dédiés à l'ingénierie système comme SysML (SysML, 2015) qui sera utilisé dans ce mémoire. AADL (Feiler, 2006) ou MARTE (MARTE, 2011) permettent également de modéliser et d'analyser les systèmes temps réel embarqués et disposent d'un outillage associé comme Ocarina (Hugues, 2008). Ces langages que nous appellerons « Langages MBSE » ont en commun de

pouvoir décrire une architecture hiérarchique de composants. Dans ces langages, un composant dispose d'attributs et d'une relation d'allocation permettant d'affecter un composant « source » à un ou plusieurs composants « destination ».

Problématique

Une méthode *MBSE* englobe l'ensemble des activités permettant de spécifier, concevoir et de vérifier un système. Avec cette démarche, l'ingénieur système apporte au client une solution performante avec un coût maitrisé (une solution de compromis), tout en satisfaisant un ensemble de contraintes en lien avec les parties prenantes. Parmi les activités *MBSE*, la recherche des solutions de compromis est une activité primordiale. En effet cette activité se traduit par un certain nombre de choix d'architectures, par exemple un choix de composants physiques ou un niveau de redondance. Un choix (ou une alternative) représente l'ensemble des possibilités (ou des variantes) du système. Ces choix sont déterminants pour le système livré, car réalisés très tôt dans le cycle de vie du système.

Ces choix d'architecture sont de plus en plus nombreux, en raison de l'augmentation constante de la complexité des systèmes, impliquant davantage de rigueur et de formalisation. Il faut pouvoir intégrer l'aspect hétérogène des éléments du système, qui se traduit par des équipes de développement multidisciplinaires : concepteurs logiciels, mécaniciens, électroniciens, chacun disposant de ses propres outils et formalismes. Les éléments hétérogènes conçus par ces équipes peuvent alors être paramétrés par des variables « entières » ou des variables « réelles » selon les disciplines. Pour choisir ces paramètres, l'ingénieur système doit alors résoudre des problèmes combinatoires mais aussi des problèmes faisant intervenir des variables « continues », voir des problèmes mixtes.

Les éléments du système et donc les choix à effectuer pour ces éléments sont également dépendants les uns des autres. Par exemple le choix d'un processeur (HW) et de son architecture interne (mono-cœur ou multi-cœur) peut être lié à la définition des tâches logicielles (SW) lorsque l'on veut maximiser la performance et la robustesse du système. L'ingénieur système doit également pouvoir combiner ces différents choix entre eux.

L'ingénieur système se trouve donc face à la problématique suivante : **comment déterminer les choix d'architecture pour obtenir les solutions de compromis pour le système, sachant que les objectifs (coût, performance, fiabilité …) sont bien souvent contradictoires ?** Les approches documentaires traditionnelles ou les capacités de l'esprit humain sont suffisantes lorsque les possibilités à évaluer pour le système sont peu nombreuses.

Certes, l'avènement du *MBSE* et de SysML marque une étape majeure dans l'évolution de l'ingénierie système, aujourd'hui indispensable pour concevoir les systèmes complexes. Cependant, lorsqu'il s'agit d'obtenir les solutions de compromis, cette approche est encore limitée. L'ingénieur système doit modéliser une à une les différentes possibilités, et évaluer les différents objectifs. Bien souvent il combine les objectifs pour déterminer une « mesure d'efficacité » globale. Cette modélisation des variantes est manuelle et prend du temps.

D'une part il est facile d'oublier une possibilité, et d'autre part c'est impossible lorsqu'un trop grand nombre de variantes existent. L'approche MBSE et le langage SysML en particulier peuvent alors être enrichis pour permettre cette recherche des solutions de compromis.

Plutôt que de d'utiliser le MBSE, l'ingénieur système peut faire appel à un mathématicien ou à un spécialiste de l'optimisation, pour obtenir une représentation abstraite du problème. Dans ce cas, deux approches sont possibles : la programmation mathématique (Collette, 2013) et la programmation par contrainte (Rossi, 2006). Ces deux approches sont utilisables en ingénierie système, pour obtenir une architecture optimale. Dans le cas de la programmation mathématique, on parle de problème d'optimisation multi-objectifs et celui-ci s'écrit de la manière suivante :

Définition 1 : Problème d'optimisation multi-objectif (Collette, 2013) :

$$\text{minimiser } \boldsymbol{f}(\boldsymbol{x})$$

$$\text{avec } \boldsymbol{g}(\boldsymbol{x}) \leq 0$$

$$et\ \boldsymbol{h}(\boldsymbol{x}) = 0$$

$$o\grave{u}\ \boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}}, \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^{\boldsymbol{k}}, \boldsymbol{g}(\boldsymbol{x}) \in \mathbb{R}^{\boldsymbol{m}}, \boldsymbol{h}(\boldsymbol{x}) \in \mathbb{R}^{\boldsymbol{p}}$$

Le vecteur $\boldsymbol{f}(\boldsymbol{x})$ contient les **fonctions objectifs** $f_i(\boldsymbol{x})$ appelées également fonctions de coût ou critères d'optimisation. Les **variables de décision** x_i sont regroupées dans le vecteur $\boldsymbol{x}$. Les vecteurs $\boldsymbol{h}(\boldsymbol{x})$ et $\boldsymbol{g}(\boldsymbol{x})$ représentent respectivement le vecteur de contraintes d'égalité et le vecteur de contraintes d'inégalité. Dans la programmation par contrainte, on parle de problèmes de satisfaction de contraintes ou « CSP ». Cette technique est issue de la programmation logique et de l'intelligence artificielle. Initialement dédiée à la résolution de problèmes combinatoires elle a évolué ensuite vers la résolution de problèmes d'optimisation. Elle consiste à représenter un problème par des variables ainsi que par un ensemble de relations logiques, des contraintes.

Cette coopération entre l'ingénieur système et le spécialiste de l'optimisation se heurte à plusieurs problèmes. Le spécialiste de l'optimisation doit reformuler le problème. Il doit comprendre quelles sont les variables du problème, quelles sont les contraintes ainsi que les objectifs à minimiser ou à maximiser. Il échange avec l'ingénieur système, au risque de perdre d'importantes informations. Si l'on rajoute à cela la complexité du système, on arrive à un long processus d'optimisation.

Obtenir les variantes optimales d'un système complexe à partir d'un modèle constitue donc le verrou scientifique principal que nous souhaitons lever.

Contributions de la thèse

1. **Une méthode d'optimisation pour l'ingénierie système**
   Nous proposons d'enrichir la méthode OOSEM basée sur le langage SysML avec de nouvelles étapes d'optimisation. Ces étapes vont permettre au concepteur d'introduire des degrés de liberté dans un modèle initial, puis d'obtenir un ensemble de solutions de compromis en optimisant plusieurs objectifs comme le coût et la fiabilité du système.

2. **L'intégration de l'optimisation dans un langage MBSE et dans SysML**
   Nous proposons pour cela d'introduire les notions de point de décision, de fonctions objectifs, de contrainte d'optimisation et de contexte d'optimisation. Ces notions sont définies à l'aide de nouveaux stéréotypes, dans un métamodèle. Les relations entre ces nouveaux stéréotypes, ainsi qu'avec les éléments du langage MBSE (composants, attributs, relations) sont également présentées. Pour le langage SysML, les stéréotypes proposés sont ensuite définis par extension du métamodèle SysML. Une nouvelle utilisation des diagrammes BDD et paramétrique de SysML est également proposée pour l'optimisation ainsi qu'une formalisation permettant une vérification par les algorithmes.

3. **Un ensemble de transformations pour résoudre le problème d'optimisation.**
   Ces transformations automatisent la phase d'optimisation du modèle. Le résultat de ces transformations est une définition d'un problème de satisfaction de contraintes dédié à l'optimisation ou « CSMOP » (Constraint Satisfaction Multi-criteria Optimization Problem). Ce problème peut ensuite être résolu grâce à des solveurs disponibles et développés en dehors des travaux exposés dans ce mémoire.

4. **Une démonstration d'applicabilité**
   - Une implémentation sous forme de plug-in des transformations dans un outil de modélisation système open source existant (Papyrus) complétée par l'intégration de trois solveurs différents, selon le type de problème à traiter (continu, discret ou mixte).
   - Le développement de trois cas d'études correspondant à des systèmes complexes aux caractéristiques différentes : un système à redondance matérielle, un système optique, un système de pilotage d'un drone.

Plan de la thèse

Le chapitre 1 constitue l'état de l'art et contient trois sections. Nous rappelons tout d'abord les principes de l'ingénierie système (IS), du MBSE et des différentes méthodes et langages associés. Ces méthodes permettent en partie au concepteur d'obtenir une architecture de compromis. Dans une seconde section dédiée à l'optimisation nous détaillons la programmation mathématique et la programmation par contrainte, sous l'angle de l'ingénierie système,

ainsi que les solveurs associés. Enfin dans une troisième section nous étudions les différents travaux combinant l'optimisation, l'ingénierie système et le MBSE. Dans le chapitre 2 nous proposons les notions nécessaires à l'intégration de l'optimisation dans le MBSE sous forme de méta-modèle. Dans une première section, nous proposons d'enrichir un langage MBSE avec des « points de décision ». Dans une seconde section nous proposons d'associer à ces points de décision plusieurs types de variables propres au problème d'optimisation CSMOP pour le résoudre efficacement. Puis nous formulons un ensemble de contraintes à intégrer dans le modèle initial ainsi que les fonctions objectifs, et nous réunissons l'ensemble de ces nouvelles notions dans un contexte d'optimisation. Finalement, nous décrirons les transformations permettant de générer automatiquement le problème CSMOP à partir du modèle du système enrichi avec les points de décisions. Le chapitre 3 présente l'intégration dans le langage SysML des notions développées dans le chapitre 2. Une première partie définit ces notions par extension des classes de base UML (méta-classe) ou par spécialisation d'un stéréotype. Une nouvelle utilisation des diagrammes de blocs (BDD) et paramétrique de SysML est également proposée pour l'optimisation. Une seconde partie adapte les algorithmes de transformation proposés pour prendre en entrée le métamodèle UML/SysML ainsi que les nouveaux stéréotypes proposés. Le résultat de cette transformation est une définition d'un problème d'optimisation CSMOP, pouvant être résolu par un solveur tel que ceux présentés au chapitre 1. Enfin, pour permettre la mise en œuvre des différents solveurs, nous proposons la définition d'un métamodèle décrivant le problème CSMOP indépendamment d'un solveur particulier. La sémantique de ce métamodèle est décrite au format EBNF. Dans le chapitre 4, la méthode d'optimisation proposée est détaillée. Nous présentons son intégration dans la méthode OOSEM basée sur SysML, et pour laquelle plusieurs activités sont modifiées. L'implémentation de la méthode dans l'outillage Papyrus sous forme de plug-in complète cett**e** présentation. Puis dans une seconde section, les trois cas d'études développés tout au long de la thèse sont explorés (choix de composant pour une caméra, intégration de la caméra dans un véhicule, système de pilotage d'un drone).

Une conclusion présentant les perspectives de la thèse vient conclure ce mémoire dans un dernier chapitre.

Publications

- Le chapitre d'ouvrage "Architecture Optimization with SysML Modeling A Case Study Using Variability" écrit par Leserf, P., Saqui-Sannes, P. D., Hugues, J., & Chaaban, K. de l'ouvrage CCIS – Communications in Computer and Information Science, Springer-Verlag, Heidelberg, Vol. 580 Chap. 18, 2015

- L'article " Multi-Domain optimization with SysML modeling." écrit par Leserf, P., Saqui-Sannes, P. D., Hugues. 20th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) 2015, Luxembourg

- L'article « SysML Modeling For Embedded Systems Design Optimization: A Case Study" écrit par Leserf, P., Saqui-Sannes, P. D., Hugues, J., & Chaaban. In Proceeding of ModelsWard 2015, Angers, France.

- La présentation en worskshop « Trade-off analysis with SysML and Papyrus : a drone application" de P. Leserf. CPSE-Labs, SysML France, January 2016 - Toulouse

- La présentation en workshop « Trade-off analysis with SysML/Papyrus" de P. Leserf. Séminaire « Optimisation des architecture système » , Juin 2016, CNES Toulouse

1. Ingénierie Système, MBSE et Optimisation

Ce chapitre constitue l'état de l'art et contient trois sections. La première rappelle les principes de l'ingénierie système (IS), du MBSE et des méthodes permettant au concepteur d'obtenir une architecture de compromis. Dans une seconde section dédiée à l'optimisation nous détaillons la programmation mathématique et la programmation par contrainte, sous l'angle de l'ingénierie système, ainsi que les solveurs associés. Enfin dans une troisième section nous étudions les différents travaux combinant l'optimisation, l'ingénierie système et le MBSE.

1.1. L'ingénierie système, le MBSE et les méthodes associées

Dans cette section nous rappelons les principes de l'ingénierie système, ainsi que l'ingénierie système dirigée par les modèles (ou MBSE, *Model-Based Systems Engineering*, en anglais). Les différentes méthodes associées sont présentées, et en particulier comment elles permettent au concepteur d'obtenir une architecture de compromis ou optimale.

1.1.1. Ingénierie système

Dans cette sous-section, nous présentons différentes méthodes d'ingénierie système. Ces méthodes disposent d'activités communes, et nous proposons d'en améliorer deux d'entre elles dans cette thèse : la modélisation et l'analyse de compromis. Les figures de cette sous-section proviennent de (Friedenthal, 2014). Pour les notions de système et d'ingénierie système, nous nous référerons aux définitions de l'International Council on Systems Engineering (INCOSE, 2007) (Friedenthal, 2007). L'INCOSE est un organisme sans but lucratif, fondé en 1990 et dont la mission est de faire progresser l'ingénierie système dans l'industrie. Selon l'INCOSE, un système est une création humaine, constitué de blocs ou de composants qui concourent au même but ne pouvant être atteint par chacun des blocs. Un bloc peut être de type logiciel (SW), matériel (HW), mécanique, ou également une personne. Selon l'INCOSE, un système est défini par un ensemble d'éléments interagissant ensemble, et constituant un tout qui lui-même interagit avec un environnement.

L'ingénierie système (Friedenthal, 2014) est une approche multidisciplinaire permettant de développer des solutions « équilibrées » en réponses à de multiples exigences. Les méthodes de développement de tels systèmes doivent s'adapter à ces exigences de plus en plus nombreuses. L'ingénierie système s'est développée dans l'aéronautique et l'automobile, pour développer des systèmes incluant des éléments physiques (mécaniques, optiques, etc.), électroniques (HW) et logiciels (SW) interagissant avec l'environnement.

L'ingénierie système a pour but :

1) La définition du système, et la documentation des exigences dans les phases amont du développement.
2) La conception du système et la vérification du système par rapport à ses exigences.

Plusieurs processus, méthodes et outils existent pour décrire et mettre en œuvre les activités d'ingénierie système. Un processus définit les activités en terme d'objectif à atteindre sans préciser comment atteindre ces objectifs. Dans la Figure 1 « Processus, méthodes et outils » une méthode va définir comment atteindre ces objectifs, en se basant éventuellement sur un langage de modélisation pour définir une sémantique. Ce langage peut être utilisé par un outil. Le concept de méthodologie va englober toutes ces notions, liant un processus à une méthode ainsi qu'à un langage.

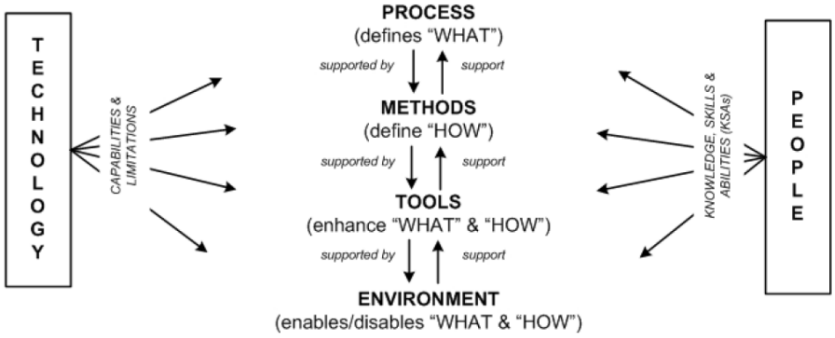


Figure 1 « Processus, méthodes et outils »

Parmi les processus d'ingénierie système, on peut citer la norme ISO 15 288 (ISO-15288, 2008) et le CMMI (Capability Maturity Model Integration) (Dufay, 2010). La norme ISO 15 288 (Figure 2 « Processus ISO 15288 ») date de 2003 et a été revue en 2008 et 2015 pour intégrer les systèmes logiciels. Elle étend les différents processus à tout le cycle de vie du système (exploitation, maintenance, démantèlement et recyclage du système). Cette norme propose également des processus d'entreprise, focalisés sur l'ingénierie système. Aujourd'hui cette norme apparaît comme la norme de référence, aussi bien pour l' AFIS (Association Française de l'Ingénierie Système) que par son homologue américain, l'INCOSE.

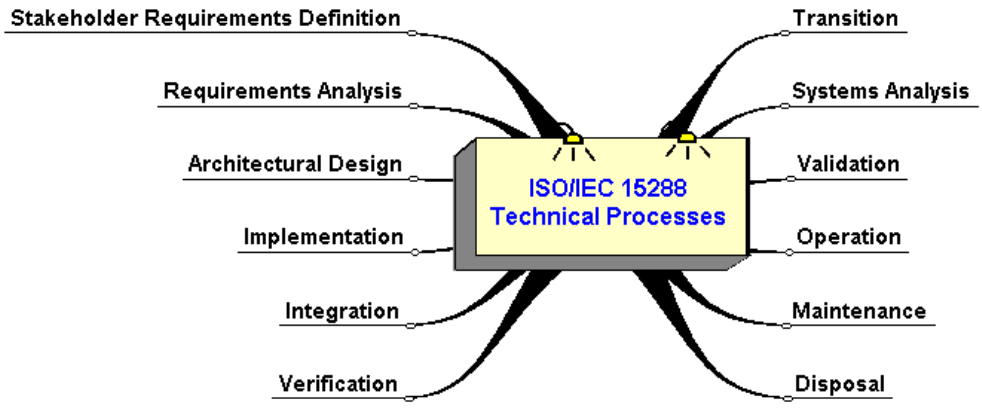


Figure 2 « Processus ISO 15288 »

Le CMMI est une approche d'ingénierie des systèmes permettant de transformer des besoins utilisateurs en un produit technique. C'est un modèle de développement et de maintenance des systèmes et des applications informatiques. Il a été conçu dès 1987, à partir des

meilleures pratiques du logiciel, par le SEI (Software Engineering Institute) et des représentants de l'industrie du logiciel. Le CMMI est un référentiel permettant d'évaluer une organisation par rapport à ses pratiques d'ingénierie système. Les cinq niveaux de maturité du CMMI sont présentés Figure 3 « Niveaux de maturité du CMMI ».

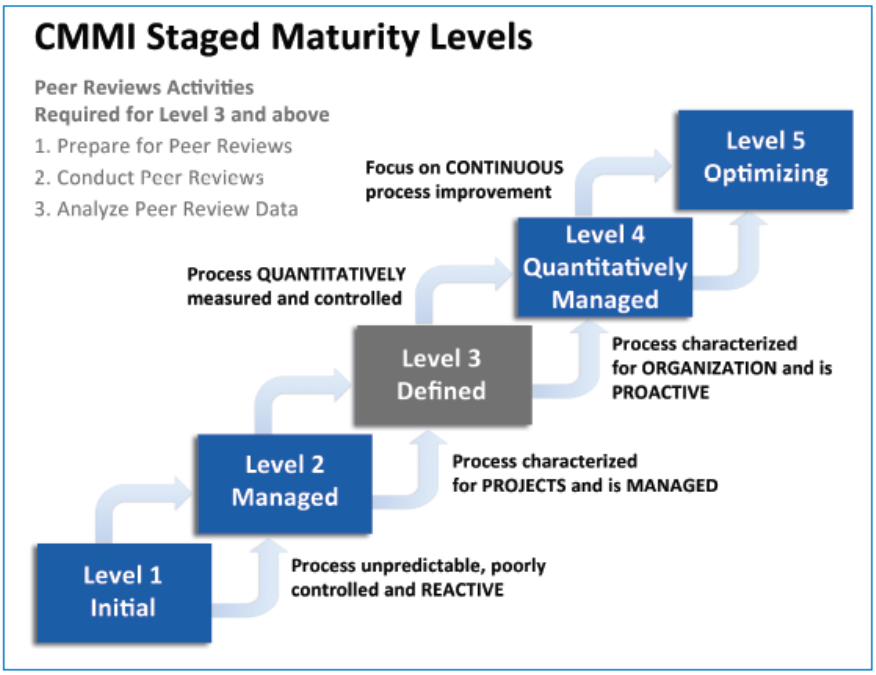


Figure 3 « Niveaux de maturité du CMMI »

Finalement, les activités d'un processus d'ingénierie système, qu'il soit de type CMMI ou ISO 15288, peuvent se résumer aux activités présentées dans la Figure 4 « Activités d'un processus d'ingénierie système » de (Friedenthal, 2014). Les besoins des parties prenantes sont analysés pour spécifier les exigences du système. Ces exigences sont alors allouées aux différents composants. Les composants du système sont alors conçus, implémentés et testés. Ces composants sont finalement intégrés dans le système final qui peut alors lui-même être testé. Dans un système complexe, plusieurs itérations seront nécessaires pour obtenir le système complet.

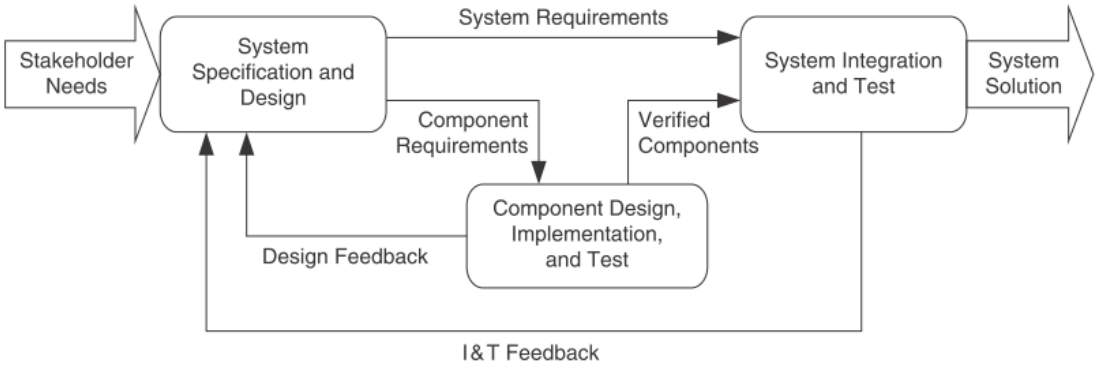


Figure 4 « Activités d'un processus d'ingénierie système »

La première étape de ce processus (System Specification and Design), permettant d'obtenir une solution équilibrée au regard des différents besoins des parties prenantes, regroupe elle-même les activités suivantes :

1. Eliciter (faire apparaître) et analyser les besoins des parties prenantes pour comprendre le problème à résoudre et les objectifs à atteindre (Stakeholder Needs). Il s'agit de déterminer les mesures d'efficacité destinées à évaluer le système par rapport à ces objectifs.
2. Spécifier les fonctionnalités du système, les interfaces, les performances et les autres caractéristiques entrant en jeu dans la mesure d'efficacité. Le résultat de cette étape est une spécification constituée d'exigences.
3. **Modéliser les différentes variantes de conception, en partitionnant le système en plusieurs composants pour satisfaire les exigences du système.**
4. **Réaliser des analyses de compromis pour évaluer et sélectionner une solution permettant de répondre de façon « optimale » aux exigences du système.**
5. Maintenir une traçabilité entre les objectifs du système, les exigences du système et des composants, et les éléments de vérification. Ceci dans le but de s'assurer que les besoins des parties prenantes sont satisfaits.

Dans cette thèse, nous traiterons **les points 3 et 4**, à savoir la modélisation du système et l'évaluation des solutions permettant de répondre de façon optimale aux exigences. Nous allons maintenant nous intéresser à la modélisation du système.

1.1.2. Ingénierie système dirigée par les modèles (MBSE)

L'ingénierie système a d'abord été mise en œuvre par la documentation textuelle. Il s'agit de l'approche documentaire. Dans cette approche, les spécifications textuelles et les documents de conception et de validation sont échangés entre les utilisateurs, concepteurs et testeurs du système. Les spécifications en particulier sont organisées dans une arborescence, depuis le système jusqu'au composants mécaniques, HW et SW. L'analyse fonctionnelle est utilisée pour décomposer les fonctions en sous-fonctions puis les allouer aux composants du système. Les diagrammes de flux et les schéma-blocs sont utilisés pour décrire respectivement les aspects fonctionnels et architecturaux. L'approche documentaire peut être rigoureuse mais elle dispose de limitations. La traçabilité entre les exigences et les éléments de conception est décrite dans de multiples documents, sous forme de relations parfois très nombreuses pour les systèmes complexes. La réutilisation des différents éléments de conception est également difficile à réaliser. Lorsque l'on veut disposer d'analyses très tôt dans le cycle de conception, de nouveaux documents doivent être produits et les données d'entrée de ces analyses sont bien souvent recopiées. Cette copie d'information est source d'erreur en cas de mise à jour de ces données.

L'ingénierie système dirigée par les modèles *MBSE* (Wymore, 1993), considère la modélisation du système comme une activité à part entière de l'ingénierie système. Cette activité va permettre d'analyser, spécifier, concevoir et vérifier le système. Un premier résultat du

MBSE est un modèle cohérent du système à développer, dans lequel les liens sont plus facile à établir entre les éléments du modèle. Ce modèle permet de générer la documentation du système, ainsi que tout ou partie d'un logiciel. Le MBSE améliore la qualité de la spécification et de la conception, favorise la réutilisation des parties du modèle développé dans ces étapes et l'amélioration des échanges au sein de l'équipe de développement.

L'approche MBSE trouve son origine dans d'autres disciplines comme la mécanique, l'électronique et l'informatique où l'utilisation des modèles est devenue courante. Dans le domaine du développement logiciel, des langages comme UML (Rumbaugh, 2004) se sont développés depuis les années 1990 et offrent les bases d'autres langages dédiés à l'ingénierie système comme SysML (Friedenthal, 2014) qui sera utilisé dans les prochains chapitres de ce mémoire.

Les objectifs de la modélisation système (MBSE)

Les objectifs du MBSE sont les suivants (Friedenthal, 2007) :

- Caractériser un système existant (reverse engineering)
- Spécifier et concevoir un nouveau ou une nouvelle version d'un système
    - Représenter les concepts d'un système
    - Spécifier et valider les exigences d'un système
    - Concevoir le système
    - Spécifier les exigences des composants du système
    - Maintenir la traçabilité des exigences
- Evaluer le système
    - **Réaliser des analyses de compromis**
    - **Analyser les performances, le coût ou d'autres paramètres du système**
    - Vérifier que la conception du système répond à ses exigences
    - Evaluer l'impact de changements d'exigences ou de conception
- Former les utilisateurs et concepteurs du système

Dans cette thèse, nous proposons d'améliorer les analyses de compromis ainsi que l'analyse des différents paramètres du système comme le coût, la performance ou la fiabilité.

Mettre en œuvre une approche MBSE demande également de choisir un langage de modélisation, nous allons présenter les langages SysML et MARTE.

1.1.3. Les langages de modélisation système : SysML/MARTE

L'origine de SysML et la notion de profil

Devenu une référence pour la modélisation systèmes, le langage SysML est issu du langage UML (Unified Modeling Langage) dont il partage un certain nombre d'éléments tout en apportant des compléments. UML, défini par l'OMG (Object Management Group), permet d'unifier à la fois les notations et les concepts orientés objets. UML a connu un important essor dans le domaine du développement logiciel, et il a ensuite été envisagé de l'utiliser en

ingénierie système. Cependant certaines faiblesses d'UML ont rendu cette tentative infructueuse, notamment :

- La difficulté de décrire des exigences et leurs liens vers les éléments du système
- Représenter les équations de la physique
- Définir des allocations au sens large, pas seulement SW vers HW

La nouvelle version UML 2 adoptée en 2001 par l'OMG (UML2, 2007) a permis d'effectuer un premier pas vers les besoins des ingénieurs systèmes, sans pour autant répondre à toutes leurs attentes. En particulier il restait encore trop de notions proche du développement logiciel comme la notion d'« objet » et de « classe ». A partir de 2006, L'OMG a donc proposé l'adoption d'un nouveau langage d'ingénierie système, SysML (SysML, 2015). Ce langage se base sur UML 2, tout en proposant de nouvelles extension, et constitue un profil UML. Un grand choix d'outils implémente SysML, à la fois sur modèle open source (Modelio, Papyrus, TTool) ou commercial (Artisan, Magic Draw, Rhapsody).

Un profil sert à adapter un langage de base à un domaine particulier ou à une problématique spécifique. SysML et MARTE sont deux profils du langage UML, le premier dédié à l'ingénierie système et le second aux systèmes embarqués. Les concepts du langage de base sont décrits par des classes, étendu par de nouvelles classes appelées stéréotypes dans le profil. On peut voir un **stéréotype** comme une étiquette que l'on rajoute à certains éléments du langage de base. On distingue deux types de profils (Sélic, 2014), les profils destinés à étendre les concepts existants dans UML (profil de langage) et les profils apportant des informations supplémentaires permettant d'analyser un modèle ou de générer du code (profil d'annotation). Dans la Figure 5 « exemples de stéréotype SysML » (SysML, 2015) une exigence fonctionnelle « functional requirement » est définie par la spécialisation d'une exigence SysML « requirement ». Ce nouveau stéréotype dispose également d'une relation avec une fonction de type « Behavior » correspondant à une classe de base (une « metaclass ») du langage UML. Le stéréotype « ConfigurationItem » est défini par extension des méta-classe UML « NamedElement » et « DirectedRelationship ». Un modèle utilisant ces stéréotypes pour représenter une exigence liée à une distance de freinage d'un véhicule est présenté également.

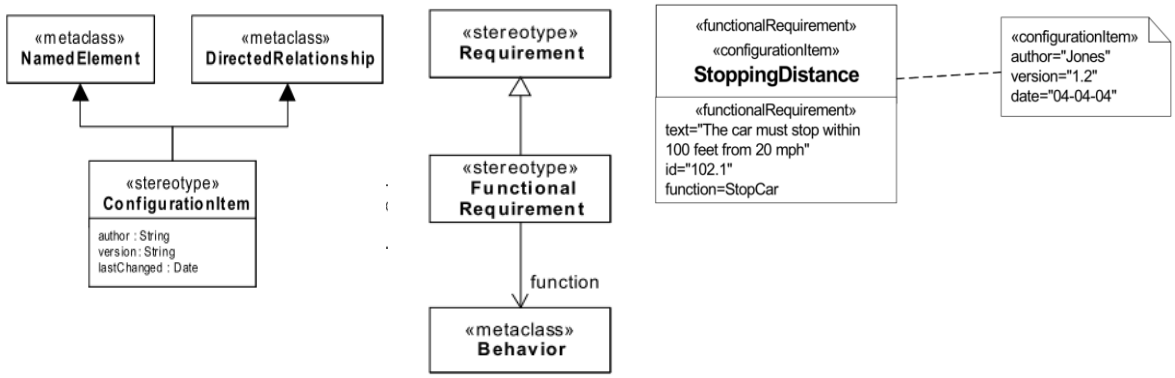


Figure 5 « exemples de stéréotype SysML »

Le langage SysML

Le langage SysML inclut une représentation graphique des éléments d'un système et permettant l'analyse (Bruel, 2013), la spécification, la conception, la vérification et la validation des systèmes complexes. De tels systèmes se composent de composants mécaniques, électroniques (HW), logiciels (SW), d'utilisateurs et de procédures. SysML peut également être utilisé conjointement avec d'autres langages plus spécifiques à un domaine, comme UML pour les logiciels et VHDL pour les composants électroniques. SysML permet de représenter les éléments suivants d'un système :

- La structure du système, incluant la hiérarchie et la classification des composants
- Le comportement du système, qui peut être fonctionnel, ou bien par échange de messages ou par des machines à états
- Des contraintes applicables aux propriétés du modèle
- Des relations d'allocations entre les comportements, les exigences et la structure du système
- Des exigences et leurs relations avec les autres éléments du modèle

Modèle d'un système SysML

La Figure 6 « Structure d'un modèle SysML » présente certaines composantes du modèle SysML. Ce modèle se compose d'un ensemble de diagrammes de type « ibd », « act », « req » et « par ». Ces diagrammes rassemblent différents éléments du modèle : « ibd » les éléments de structure du système (éléments physiques, composants mécaniques ou électroniques), « act » les éléments logiques du système (composant logiciel), « req » les exigences du système et « par » les équations de la physique s'appliquant au système. Entre ces différents éléments, les flèches représentent des liens de type « allocation », entre une exigence et un composant par exemple ou bien entre un composant logique et un composant physique.

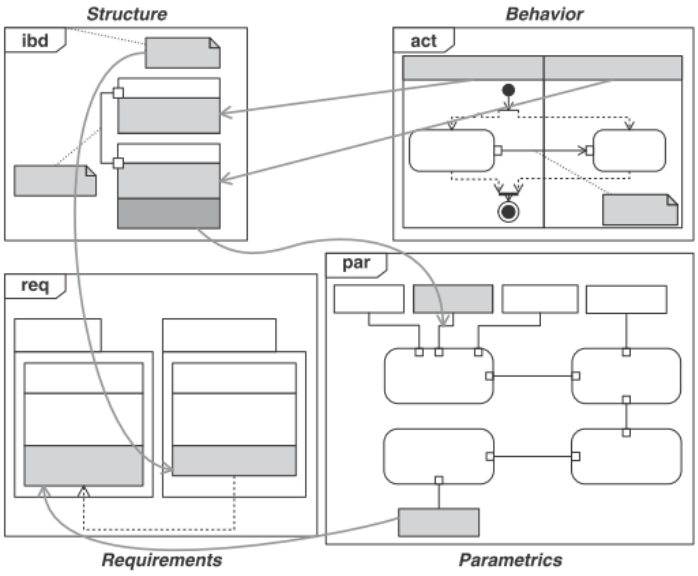


Figure 6 « Structure d'un modèle SysML »

Les diagrammes SysML

Le langage SysML dans sa version 1.4 (SysML, 2015) se compose de neuf types de diagrammes que l'on peut classer en trois groupes (Roques, 2013) :

Les diagrammes de comportement

- Le diagramme de séquence (**SEQ**) représente les échanges de messages entre différentes entités suivant une chronologie verticale de haut en bas
- Le diagramme d'activité (**ACT**)pour un enchaînement d'actions et de décisions, permettant des modéliser un processus par exemple.
- Le diagramme de cas d'utilisation, pour les interactions de type fonctionnelles entre des acteurs externes au système et des services ou fonctionnalités de haut niveau que le système doit offrir.
- Le diagramme d'états semblable à un automate à états finis

*Le diagramme d'exigences (**REQ**)*

- Il représente les exigences, leur organisation ainsi que leurs relations avec les éléments du système

Les diagrammes structurels

Ces diagrammes sont utilisés pour modéliser l'architecture du système, incluant :

- Le diagramme de définition de blocs (**BDD**) présente les blocs ainsi que les liens de composition, de spécialisation ou de référence.
- Le diagramme de blocs internes (**IBD**) représente l'organisation interne d'un bloc et de ses sous-blocs, et en particulier les interconnections entre les ports.
- Le diagramme paramétrique (**PAR**) représente des équations de la physique du système, ou également un réseau de blocs contraintes.

Avantages et limitations de SysML

Dans (Desfray, 2012), une étude concernant la diffusion de SysML dans le domaine de l'ingénierie système est présentée. SysML connait un succès croissant dans l'industrie, avec un grand nombre d'outils implémentant ce langage. Les retours des utilisateurs industriels (128 réponses analysées) sont positifs : SysML permet de représenter les concepts clé du système et sa représentation est jugée suffisamment précise en non ambiguë. La modélisation des exigences en particulier représente une grande valeur ajoutée de SysML. Les utilisateurs sont pour la plupart des ingénieurs systèmes mais également logiciels et HW, ainsi que des spécialistes du test. Les diagrammes les plus utilisés sont les BDD, IBD, REQ tandis que le diagramme paramétrique est moins bien compris et employé.

Parmi les limitations pointées par les utilisateurs dans (Desfray, 2012), on peut citer la faible représentation du temps ou des performances, ainsi que de multiples façons de représenter la même chose. Les concepts permettant d'analyser le système et donc de réaliser des choix

de conception dans les phases amont du développement sont également peu nombreux, en dehors du diagramme paramétrique.

Le langage SysML reste général, et lorsque l'on doit modéliser un système complexe hétérogène composé par exemple de partie mécanique, de HW et de SW, les niveaux de détail des concepts SysML sont insuffisants. SysML et AADL peuvent également être combinés (De Saqui-Sannes, 2012) lorsqu'il s'agit d'un système embarqué temps réel. Si l'on souhaite utiliser des langages dérivés d'UML, on peut alors utiliser SysML conjointement avec un autre profil, comme le profil MARTE pour modéliser notamment la partie logicielle du système et les temps de réponse.

Utilisation conjointe de SysML/MARTE

Le Profil MARTE

MARTE (MARTE, 2011) est le profil UML 2 dédié à la modélisation et l'analyse des systèmes embarqués. Il définit des mécanismes décrivant le temps, comme des événements temporels et des horloges. Il permet également de modéliser et de simuler (Mraidha, 2008) une plateforme d'exécution HW (processeur, réseaux embarqué, mémoire), ainsi qu'un logiciel composé de tâches (Schedulable resources) et d'un ordonnanceur (OS temps réel).

Combinaison SysML/MARTE (Sélic, 2014)

Pour modéliser un système complexe hétérogène, le système lui-même (le tout) doit être modélisé dans son ensemble, tout comme les parties qui le composent (de type HW, SW, mécanique ou autre) et les relations entre le système et ses parties. Pour modéliser cette hiérarchie, (Espinoza, 2009) préconise de commencer par modéliser le système par des blocs SysML, puis de rajouter ensuite de la sémantique à ces blocs, à l'aide d'un profil MARTE par exemple. Ces deux étapes de modélisation (modélisation système, modélisation détaillée) sont présentées dans la figure suivante.

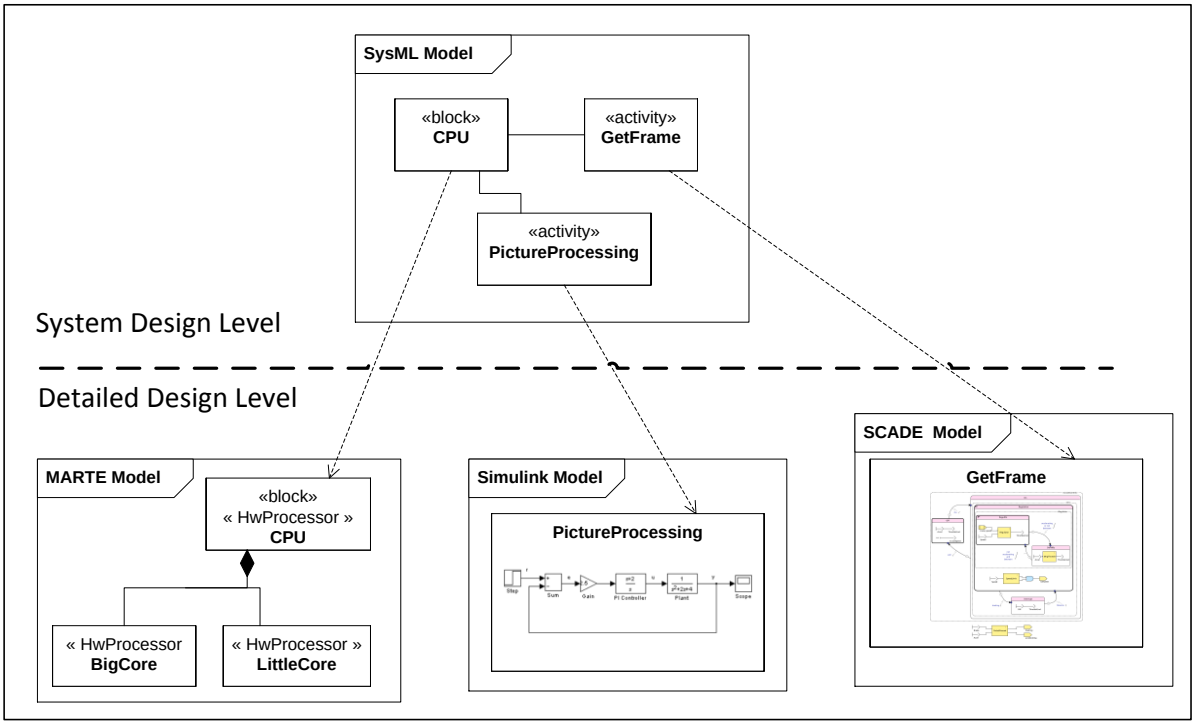


Figure 7 « Modélisation système et modélisation détaillée »

Deux approches sont possibles pour maintenir une cohérence entre le modèle du système et la modélisation détaillée. Dans la première approche dite par raffinement, un langage comme SysML est utilisé pour modéliser l'architecture du système dans un premier modèle avec des blocs, et d'autres langages comme MARTE, Simulink ou SCADE pour modéliser les sous-parties du système, dans une seconde étape de conception détaillée. Dans ce cas les modèles sont disjoints et maintenus généralement par des outils différents. Les liens de raffinement entre les éléments hétérogènes peuvent être maintenus par un outil tierce chargé de gérer cette traçabilité. Par exemple une fonction d'acquisition de température peut être modélisée par un bloc SysML puis raffinée par un composant logiciel, le driver, ainsi que par un composant HW, le capteur.

L'autre option consiste à ne conserver qu'un seul niveau de modélisation mais de séparer le modèle en différentes parties également reliées. Dans notre exemple précédent, on modéliserait le capteur (HW) et le driver (SW) ainsi qu'une relation entre ces deux éléments. Ces deux modélisations peuvent être réalisées dans deux outils différents, ou bien dans un seul si les langages le permettent. Ainsi comme les profils MARTE et SysML sont tous les deux dérivés d'UML, il est possible dans le même outil de partitionner (avec des Packages) les éléments SW et HW en utilisant MARTE, et les autre éléments (mécanique, optique, etc.) avec SysML. On peut également utiliser les deux profils par recouvrement, en appliquant des stéréotypes MARTE à des éléments SysML lorsque c'est nécessaire et en vérifiant qu'il n'y a pas de conflit de sémantique. La Figure 42«Exemple d'allocation variable» de la page 85 présente l'utilisation conjointe des stéréotype SysML « block » et « activity » avec les stéréotype MARTE « SchedulableResource » et HwComputingResource ».

Ces deux langages (SysML/MARTE) permettent de modéliser un système. Cependant lorsqu'il faut modéliser les variantes d'un système (ou les alternatives du système), les possibilités offertes sont limitées. Une variante représente soit une possibilité pour le système réel (par exemple une combinaison d'options pour un véhicule donné) soit une hypothèse de conception pour le système final. La notion d'héritage ou les diagrammes d'instance permettent de modéliser certaines variantes dans ces langages. Pour modéliser des variations plus complexes, d'autre techniques sont utilisées.

1.1.4. Modéliser les variantes d'un système

La modélisation des variantes d'un système est rendue nécessaire au regard des contraintes et des besoins des parties prenantes. Bien souvent, il n'y a pas de solution unique au problème posé, et le concepteur du système doit modéliser et analyser plusieurs possibilités. Cette notion de modélisation des variantes d'un système est également appelée la variabilité. La variabilité définit un ensemble de changements « à priori » dans un système, par opposition à des changements provoqués par des erreurs détectées et corrigées sur le système (Galster, 2014). Ces changements « a priori » nous intéressent dans le cadre de cette thèse, car cette notion est proche de celle de « degré de liberté » d'un élément du système conduisant à différentes variantes du système complet.

Lorsque le système est de type logiciel, la variabilité est une manière de gérer les différents artéfacts résultants du processus de développement, pour différencier les différents produits logiciels (Svahnberg, 2005). Lorsque le système n'est pas seulement logiciel, la variabilité permet de décliner différentes lignes de produits ; citons par exemple les multiples configurations d'un véhicule de série en fonction des options choisies par le client. Les différences entre ces lignes de produits sont définies avec la notion de caractéristique ou « *features »* (Kang, 1990). Dans tous les cas, la variabilité d'un système peut être décrite par des annotations ou des relations, à partir d'un modèle décrivant la partie invariante du système (un modèle de base). Une autre option consiste à utiliser un modèle séparé représentant uniquement la variabilité du système. Dans ce dernier cas, les modèles de caractéristiques ou « feature models » sont largement utilisés, mais également des langages comme OVM ou CVL.

Les modèles de caractéristiques (Feature Models)

Ces modèles décrivent par eux-mêmes la variabilité du système, sans avoir de lien avec d'autres modèles. Les principes d'utilisation des modèles de caractéristiques et la méthode d'analyse associée (FODA) sont présentés dans (Kang, 1990). Une caractéristique y est définie comme un aspect important ou distinctif d'un système, visible par l'utilisateur ou influant la qualité du produit final. Cette notion peut concerner les exigences, les fonctions, les constituants ou les propriétés physiques du système.

Les modèles de caractéristiques représentent les caractéristiques sous forme d'une arborescence comme l'on peut voir dans la Figure 8 « Exemple de modèle de caractéristiques » .

Dans cette arborescence, les caractéristiques sont représentées par des nœuds, selon une relation d'affinement. Les relations entre les nœuds père et fils modélisent quatre types de relations entre les caractéristiques : obligatoire (le véhicule doit avoir une boîte de vitesse), optionnelle (accès au véhicule sans clé de contact), à choix alternatif (boîte manuelle ou automatique exclusivement), à choix multiple (moteur électrique, au gaz ou hybride).

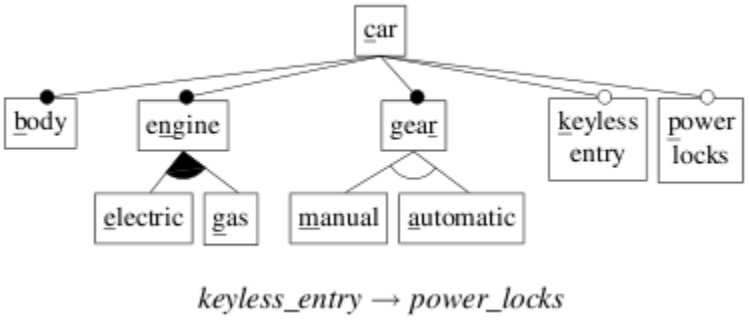


Figure 8 « Exemple de modèle de caractéristiques »

Cependant les modèles de caractéristiques définis dans (Kang, 1990) sont limités par leur pouvoir d'expression, car le concept de caractéristique présente ou pas dans le système est booléen par définition. Il est également impossible de spécifier des ensembles ou d'avoir un concept d'instanciation comme dans les langages orientés objets comme UML ou SysML. Les modèles de caractéristiques ont donc été étendus, tout d'abord en introduisant la cardinalité des caractéristiques (Czarnecki, 2005). Les multiplicités d'UML ont été utilisées dans ce modèle comme extension de la notation originale des caractéristiques définies dans (Kang, 1990), avec la notion de clonage d'une caractéristique.

Les modèles de caractéristiques sont donc bien adaptés pour exprimer la variabilité visible par le client dans le produit final, sous forme d'une collection ordonnée de caractéristiques. Ces modèles sont intuitifs, faciles à interpréter et à spécifier, même par des non-spécialistes de l'ingénierie système. Cependant lorsqu'il s'agit de modéliser des choix de conception, ces modèles sont peu adaptés. D'une part en raison de la structure arborescente de ces modèles et d'autre part parce qu'il s'agit d'un formalisme indépendant d'un langage de modélisation système, comme SysML. On peut penser que les caractéristiques d'un système sont déjà présentes dans un modèle SysML, et dans ce cas il est inutile et coûteux de les modéliser à nouveau sous forme de caractéristiques.

La variabilité définie par rapport à un modèle de base

Sous cette forme, la variabilité est associée aux éléments d'un modèle de base, ce modèle de base décrivant la partie invariante du système. Deux approches ont été développées jusqu'à présent.

La première est présentée dans (Ziadi, 2004) et dans (Fontoura, 2001). Il s'agit d'une documentation intégrée au modèle, sous forme d'annotations ou de stéréotypes. De nouveaux

stéréotypes dédiés à la variabilité sont proposés dans (Fontoura, 2001) alors que (Ziadi, 2004) propose d'utiliser des mécanismes comme l'héritage, les « templates » ou les « framework » à partir du langage UML. L'avantage de ces approches est que les éléments du modèle affectés par la variabilité sont clairement identifiés. Par contre la variabilité et le modèle de base sont au même niveau, ce qui peut poser problème lorsqu'un grand nombre de point de variabilités sont présents. De plus cette approche a été développée dans un but de définition de lignes de produit, et nécessite d'être adaptée pour un autre objectif.

La seconde approche consiste à réaliser un modèle indépendant pour la variabilité (un modèle orthogonal), tout en maintenant des liens entre ce modèle et les éléments du modèle de base. Dans cette catégorie, nous trouvons **le langage OVM** (Orthogonal Variability Model) dont le formalisme est décrit dans (Pohl, 2005). Un modèle OVM permet de définir les propriétés variables de tous les produits d'une même famille. A partir de plusieurs modèles d'exigences, il est possible d'extraire l'information de variabilité dans un seul modèle indépendant. Ceci pour répondre à la question : quels sont les éléments qui varient ? De quelle façon ces éléments varient dans une ligne de produit logicielle ?

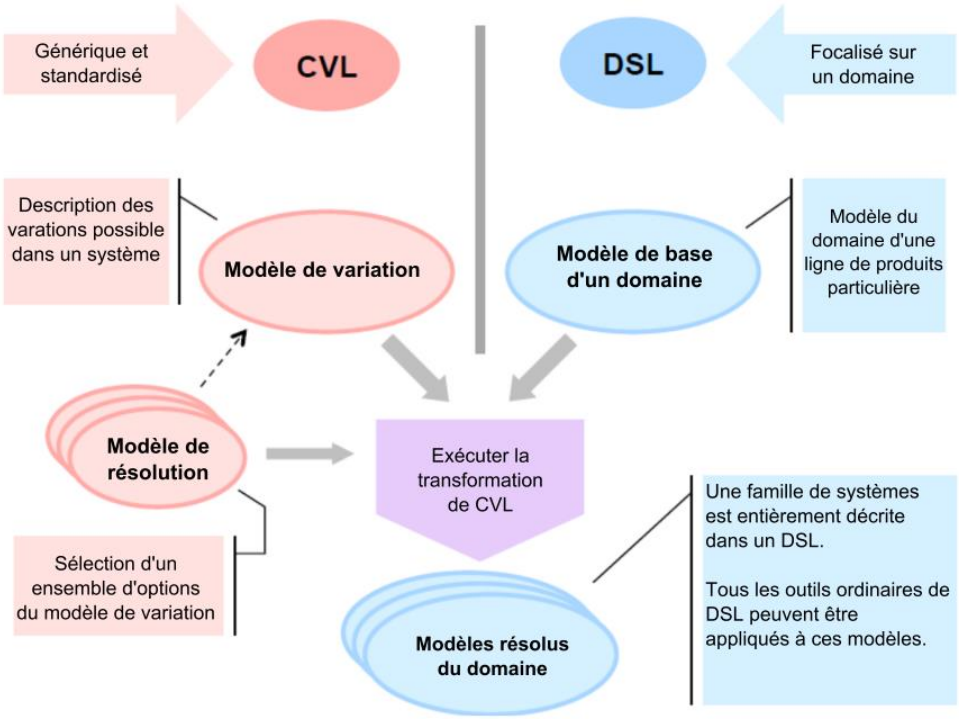


Figure 9 « Le langage CVL et le DSL associé »

Le langage CVL (Haugen, 2008) est comparable à OVM. Il s'agit d'un langage de domaine indépendant pour la spécification et la résolution de la variabilité à partir d'un DSL (Domain Specific Langage). Bien que spécifié à l'OMG, ce langage n'a pas encore été adopté et ne constitue pas une norme à ce jour. Son objectif est de standardiser la variabilité dans un contexte MBSE. La Figure 11 présente les relations entre les deux langages. Le modèle de variation définit comment les éléments du DSL peuvent varier. Chaque modèle de résolution décrit une variante du modèle de base, par exécution d'un script. La Figure 10 « Extrait du

métamodèle de CVL » dans (Rouillé, 2014) présente la partie abstraction de la variabilité de CVL. La variabilité est définie avec une spécification de variabilité (VSpec). Cette spécification peut être un choix (Choice). Un choix est une caractéristique qui appartiendra ou non au modèle résolu. Une spécification de variabilité est également un conteneur pour d'autres spécifications de variabilité, qui deviennent alors ses enfants (child). Une spécification de variabilité dispose également d'une multiplicité de groupe définissant les nombres minimum et maximum (Classe MultiplicityInterval) d'enfants pouvant être résolus.

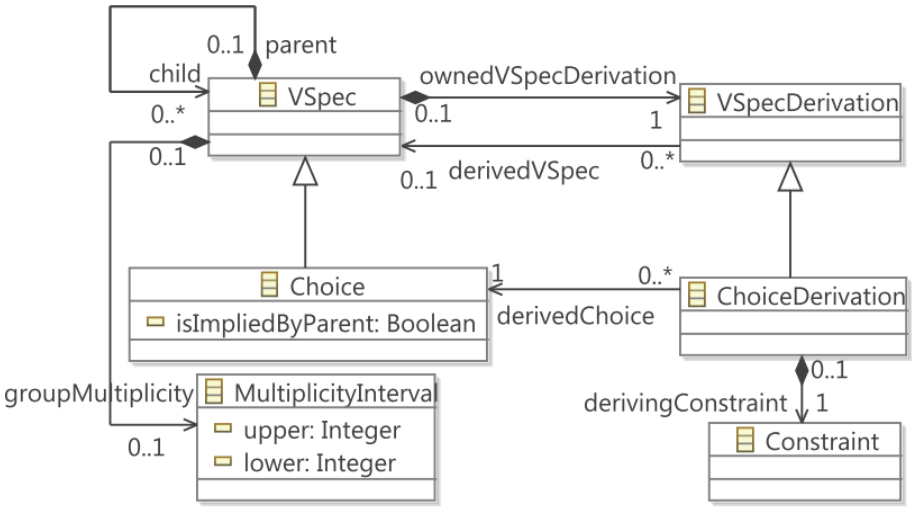


Figure 10 « Extrait du métamodèle de CVL »

Comparaison de CVL et OVM

La compréhension et l'interprétation par l'utilisateur des langages CVL et OVM ont été étudiées dans (Reinhartz-Berger, 2014). Dans cette étude, plusieurs groupes d'utilisateurs ont utilisé successivement les deux langages pour modéliser la variabilité d'un système. Il résulte de cette étude qu'il n'y a pas de différence significative dans l'utilisation de ces deux langages. La compréhension des modèles de variabilité par l'utilisateur est bonne, mais par contre les relations entre les éléments de ces modèles et ceux du modèle de base le sont moins. En effet l'utilisateur doit assimiler les informations de deux modèles différents (le modèle de variabilité et le modèle de base) simultanément. De plus la navigation entre les diagrammes des deux modèles est également problématique.

Conclusion sur la variabilité et l'ingénierie système

Plusieurs langages et modèles ont été développés pour représenter la variabilité dans les lignes de produit. D'abord avec les modèles de caractéristique, se suffisant à eux-mêmes pour décrire la variabilité, puis avec des mécanismes complétant un modèle « de base » décrivant la partie invariante du système. Dans cette catégorie, les annotations et stéréotypes permettent d'étendre un modèle existant. Puis des langages comme OVM et CVL ont vu le jour, permettant de relier un modèle de variabilité à un modèle de base.

Dans ces différentes approches, l'objectif pour la variabilité est de décrire les différentes variantes du système ou du logiciel ou du système final, du point de vue de l'utilisateur. Dans notre contexte d'optimisation du système, il s'agit plutôt de positionner des degrés de libertés pour que le concepteur puisse réaliser différentes analyses. Le mécanisme choisi doit donc être facile à mettre en œuvre par le concepteur, pour lui permettre d'ajouter/supprimer ou repositionner les différents points de variabilité entre deux analyses. C'est pourquoi la mise en œuvre de OVM ou CVL n'est pas adaptée pour cet usage, car les relations entre ces langages et le modèle de base sont complexes à gérer pour l'utilisateur comme le montre l'étude (Reinhartz-Berger, 2014). Il semble préférable de définir des annotations et des stéréotypes à partir d'un langage comme SysML (le langage du concepteur système) pour exprimer les concepts de variabilité. Ces nouveaux stéréotypes pourront cependant être compatibles avec ceux déjà définis dans CVL pour permettre une extension future à ce langage.

1.1.5. Standards et méthodes MBSE

Après avoir passé en revue les langages support au MBSE ainsi que les techniques permettant de modéliser les variantes d'un modèle, nous présentons ici différentes méthodes et MBSE. En particulier, nous nous intéressons à l'obtention et l'évaluation d'une architecture de compromis (optimisation) par ces méthodes.

La méthode ARCADIA

La méthode ARCADIA (ARChitecture Analysis & Design Integrated Approach) a été proposée par Thales en 2007 et les principes sont détaillés dans (Voirin, et al., 2015). La méthode est en cours de standardisation auprès de l'AFNOR par le biais du projet collaboratif français Clarity (Clarity, 2016). L'outil associé à cette méthode est Capella, il est implanté dans l'environnement Eclipse.

Le développement d'ARCADIA part du constat que les méthodes actuelles d'ingénierie système sont trop focalisées sur les exigences et la gestion de leur allocation sur les éléments du système. La méthode ARCADIA propose de placer l'architecture du système et la collaboration entre les acteurs au centre de l'ingénierie système. La méthode ARCADIA est destinée à la conception des systèmes complexes et des systèmes critiques, en prenant en compte les contraintes fonctionnelles et non fonctionnelles. Les principes généraux d'ARCADIA sont les suivants :

1. Le modèle est partagé entre tous les acteurs de l'ingénierie système.
2. Différents points de vue permettent de rassembler des ensembles de contraintes (sécurité, performance, coût).
3. L'architecture est vérifiée au plus tôt avec des règles à différent niveaux.

La méthode se décompose en plusieurs étapes présentées dans la Figure 11 « vue d'ensemble de la méthode ARCADIA »:

1. L'analyse opérationnelle. Dans cette phase, la problématique des utilisateurs opérationnels va être analysée, en identifiant les acteurs, leurs activités et les échanges entre eux.
2. L'analyse des besoins systèmes. Il s'agit de définir ce que le système doit réaliser en réponse à l'analyse opérationnelle, par une analyse fonctionnelle externe prenant en compte les contraintes non-fonctionnelle comme la fiabilité ou le coût du système.
3. L'architecture logique va déterminer les sous-fonctions à réaliser pour répondre à l'analyse de besoin. On répond à la question « comment le système fonctionne-t-il ? » dans cette étape.
4. L'architecture physique permet de construire le système en proposant une architecture finale. Les éléments comportementaux du système sont introduits dans cette étape.
5. L'EPBS (End-Product Breakdown Structure) permet de fournir une spécification d'intégration de chaque composant.

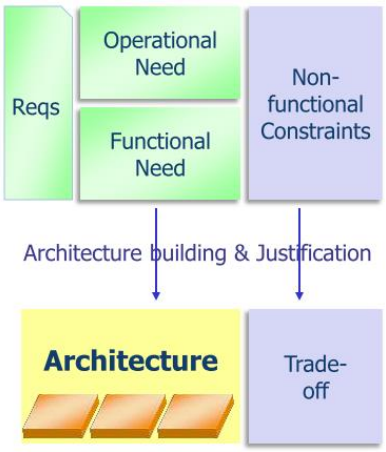


Figure 11 « vue d'ensemble de la méthode ARCADIA »

Il existe également un mécanisme dit REC-RPL permettant d'instancier un bloc de façon multiples, pour modéliser un déploiement. En effet par défaut il n'y a pas de différence entre les types et les instances, c'est le contexte d'utilisation qui permet de faire cette différence. De même il est possible d'utiliser un bloc pour typer plusieurs parties d'un bloc même si ce n'est pas dans la configuration par défaut de l'outil Capella.

La méthode ARCADIA s'appuie sur un langage spécifique (DSL) de description du système, développé par l'industriel Thales. Cependant La méthode ARCADIA est compatible avec les standards SysML, UML2 et MARTE. En particulier il dispose des concepts suivants, similaires à ceux de SysML/MARTE :

- Concepts proches de SysML :
  - Blocs, parties, Données, propriétés de type contrainte
  - Ports de communication, flux, Profils
  - Diagramme d'activité, diagramme BDD, IBD, Package, Requirement, table d'allocation

- Concepts inspirés de MARTE/AADL
    - Processeur, mémoire, tâche, processus

SysML, MARTE et AADL peuvent inter opérer avec ARCADIA, plusieurs outils d'importation/exportation sont d'ailleurs prévus dans ARCADIA.

Obtenir une architecture de compromis avec ARCADIA

Le paragraphe 1.1 l'a rappelé : obtenir la ou les architectures de compromis est un point clé de l'ingénierie système. La méthode ARCADIA répond à cette problématique avec la notion de « Multi vue » également définie par le standard ISO 42010 (ISO-42010, 2011). Plusieurs vues sont associées à une architecture, chacune d'entre elle correspondant à un objectif. Avec la méthode ARCADIA, les différents objectifs d'une architecture sont présentés à l'utilisateur (Figure 11 « vue d'ensemble de la méthode ARCADIA »), puis par itération, on modifie l'architecture jusqu'à parvenir à la satisfaction globale des objectifs en pondérant chacun d'entre eux. Ceci n'est possible que lorsque l'espace de recherche est relativement limité, de l'ordre d'une dizaine de possibilités d'architectures.

Figure 12 « Multi-vue sur l'architecture avec ARCADIA »

La méthode OOSEM

Lorsque le langage SysML a été spécifié par l'OMG, une initiative a vu le jour à l'INCOSE, celle de proposer une méthode d'ingénierie système associée à ce langage. La méthode OOSEM (Object-Oriented System Engineering Method, (Friedenthal, 2007)) a donc vu le jour. Cette méthode a été développée en 1998 par Lockheed Martin Corporation et le System and Software Consortium (SSCI). La méthode OOSEM met en œuvre également des concepts issus de

l'ingénierie logicielle, comme l'encapsulation et la spécialisation, mais appliquée à la modélisation d'un système. Elle inclue les activités fondamentales de l'ingénierie système, à savoir l'analyse des besoins, l'analyse des exigences, la conception de l'architecture système et la vérification, tout en s'appuyant sur le langage SysML. La Figure 13 « Conception d'architecture système avec OOSEM » présente les différentes activités de la conception de l'architecture système, sans présenter le détail des différents rebouclages permettant d'obtenir une architecture de compromis.

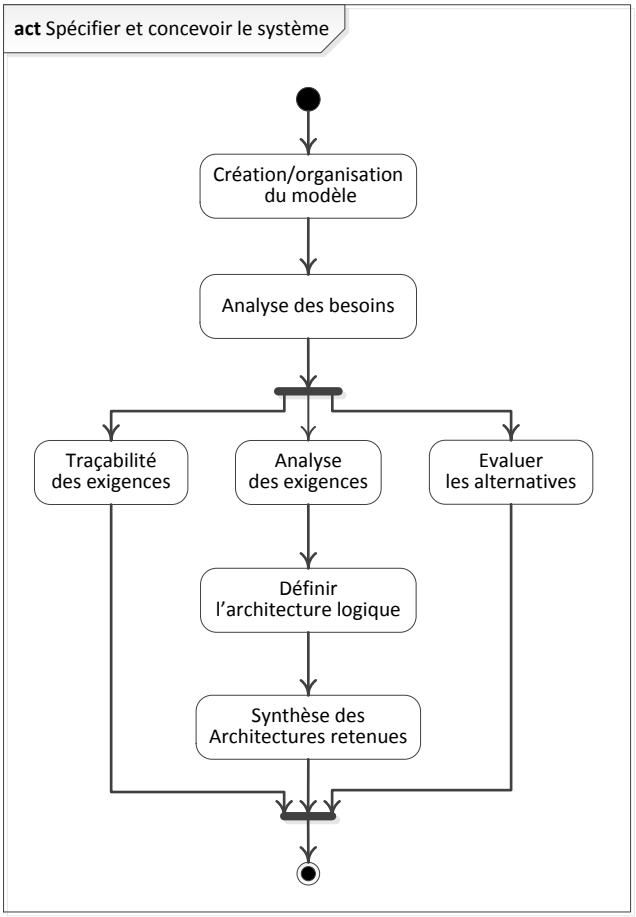


Figure 13 « Conception d'architecture système avec OOSEM »

L'analyse des besoins consiste à définir les cas d'utilisations du système à partir des besoins des parties prenantes. Pour cela, le domaine opérationnel du système est représenté avec un diagramme de bloc figurant la structure générale du système et ses liens avec les parties prenantes (utilisateurs, environnement...). Ce diagramme permet également de délimiter les contours du système par rapport à son environnement. Les missions opérationnelles du système sont représentées par un diagramme d'exigences, auxquelles sont associées des métriques pour mesurer l'efficacité du système. Chaque métrique est représentée par un diagramme paramétrique.

L'analyse des exigences du système est réalisée à partir des cas d'utilisation précédents. Il s'agit d'un processus itératif, où chaque cas d'utilisation est analysé pour déterminer un

comportement du système sous forme de diagramme d'activité ou de diagramme de séquence. Ces diagrammes représentent l'enchaînement des fonctions supportées par le système, ce qui constitue un premier niveau d'analyse fonctionnelle. Lorsque tous les cas d'utilisation sont analysés, une synthèse du système est produite sous forme d'un bloc de haut niveau, dans lequel sont représentés les fonctions à réaliser (les opérations), et les interfaces reliant le système à son environnement. Des attributs permettant d'évaluer les performances du système sont également représentés avec le stéréotype « mop ». Des contraintes sur ces attributs permettent de définir des exigences non-fonctionnelles comme des temps de réponse ou un MTBF (Mean Time Between Failures).

La définition de l'architecture logique consiste à décomposer le système en composants logiques, constituant une abstraction de composants matériels ou logiciels. Par un exemple un capteur de pression, qui peut ensuite être réalisé par un capteur hydrostatique ou un capteur à déformation de solide. Ce niveau d'architecture permet une indépendance entre les exigences et l'architecture physique, pour en évaluer plusieurs par exemple. Cette architecture logique est définie par des blocs, avec un diagramme BDD et un diagramme IBD pour modéliser les interconnections entre les composants.

L'architecture retenue inclut les composants HW, SW, les données stockées par le système (les données persistantes). Cette architecture est établie à partir des composants logiques du système, en suivant une démarche de partitionnement, mais aussi à partir des exigences du système. Le partitionnement de l'architecture physique répond à plusieurs critères, comme le regroupement de fonctions communes dans un même composant, des contraintes temporelles (même périodicité) ou spatiales lorsque des données sont partagées. Le résultat de cette étape est une bibliothèque de composants physiques sous forme d'un ensemble de blocs, ainsi que des tables d'allocation des composants logiques sur les composants physiques.

L'obtention d'une architecture de compromis

Dans la méthode OOSEM, cette activité de recherche des solutions de compromis est intégrée tout au long de la conception du système. Elle consiste à identifier les analyses à effectuer (coût, masse, fiabilité…), définir un contexte d'analyse incluant plusieurs variantes du système à analyser, puis à exécuter les analyses en utilisant un outil externe pour résoudre les équations des diagrammes paramétriques. Les attributs du système correspondant aux mesures d'efficacité « moe » sont alors mis à jour à partir des outils externes. Cette étude des solutions compromis repose sur l'utilisation de blocs de contrainte SysML ainsi que de spécialisations du système. Une variante d'un composant est représentée par une spécialisation d'un composant générique. Chaque solution candidate pour le système est caractérisée par un ensemble de mesures d'efficacité « moe » correspondant à des critères d'évaluation des solutions. Une « moe » est évaluée par une fonction objectif et les différentes variantes sont évaluée et comparées pour sélectionner une solution préférée par le concepteur. Le stéréotype « moe » défini dans (SysML, 2015) est un nouveau type de propriété d'un bloc.

De même le stéréotype « objective function » peut s'appliquer à un bloc de contrainte pour modéliser une fonction objectif. Un diagramme paramétrique permet de modéliser les paramètres et l'expression d'une fonction objectif.

Dans l'exemple présenté dans la Figure 14 « Variantes d'un système » extrait de (Friedenthal, 2014), deux variantes d'une caméra embarquée dans un véhicule sont proposées, l'une étant stéréoscopique et l'autre ne disposant que d'un seul ensemble lentille/capteur. Trois mesures d'efficacité sont associées à ces deux variantes : la consommation énergétique, les performances et le coût.

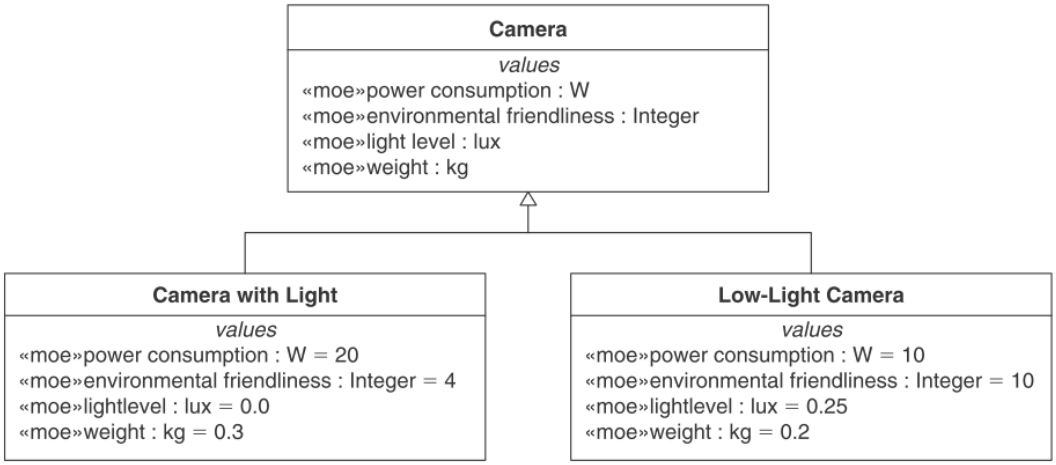


Figure 14 « Variantes d'un système »

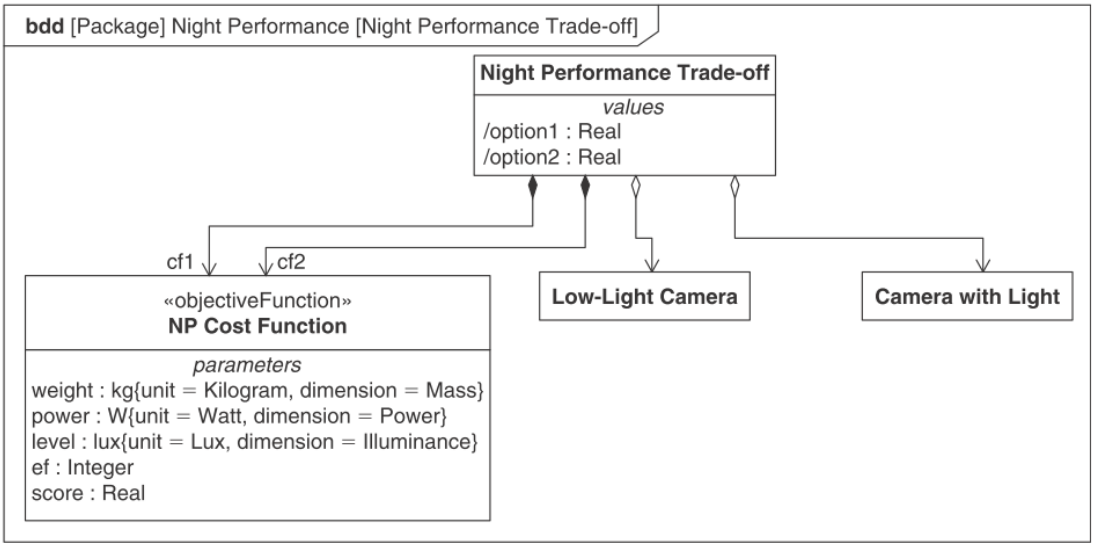


Figure 15 « exemple de Mesure d'efficacité »

L'analyse de compromis est représentée par le diagramme bdd Figure 15 « exemple de Mesure d'efficacité » ainsi que par le diagramme paramétrique associé Figure 16 « fonction objectif : diagramme paramétrique ». Les fonctions de coût cf1 et cf2 sont calculées en effectuant une moyenne pondérée des différentes « moe ». La solution stéréoscopique obtient un meilleur score et elle peut être retenue par le concepteur.

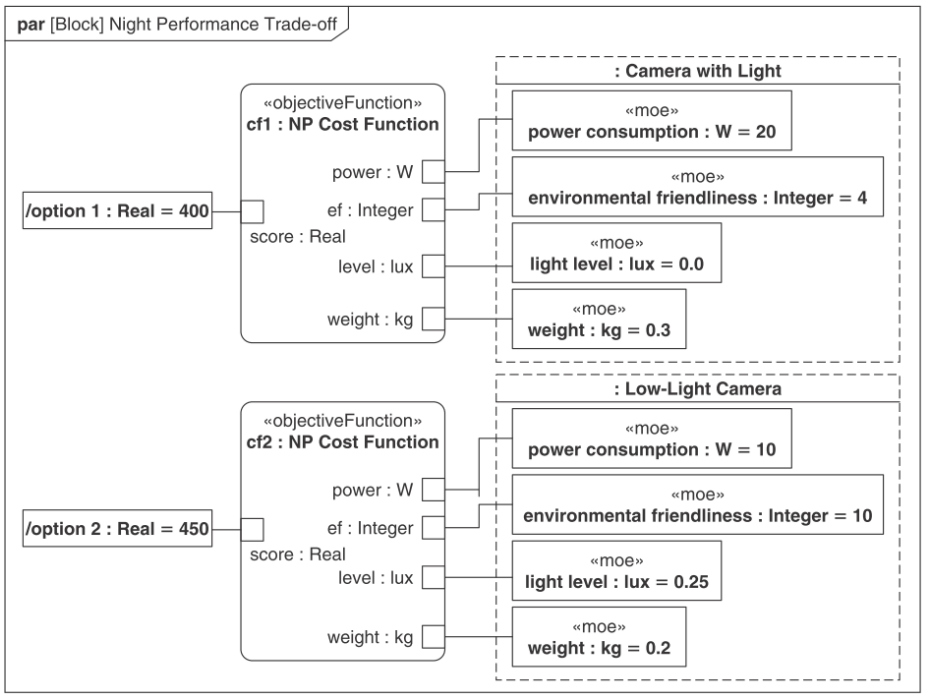


Figure 16 « fonction objectif : diagramme paramétrique »

Conclusion OOSEM

La méthode OOSEM permet de réaliser les différentes activités nécessaires à l'ingénierie système, tout en s'appuyant sur le langage SysML. En partant de l'analyse opérationnelle du système, les différentes exigences sont modélisées. La méthode permet de définir les composants logiques, puis physique pour arriver au partitionnement de ces composants dans des éléments HW, SW et en intégrant également les données du système. Les analyses de compromis font partie également de la méthode OOSEM. Les mesures d'efficacité « moe » et les fonctions objectifs « objective function » de SysML permettent d'analyser différentes variantes d'un système en utilisant un contexte d'analyse. Par contre cette démarche possède plusieurs limitations. Le premier est de devoir modéliser de façon exhaustive les différentes variantes du système, ce qui peut être coûteux voir impossible à réaliser lorsqu'il y a trop de variantes possibles. La deuxième limitation concerne la démarche proposée qui reste une analyse de solution et non une démarche d'optimisation dans laquelle on définirait les degrés de liberté du système avant d'en déterminer les valeurs optimales. La troisième limitation concerne l'agrégation de différents objectifs dans un objectif global par pondération, ce qui n'est pas toujours possible ni même souhaitable lorsque ces objectifs sont de même importance.

Synthèse des méthodes MBSE

	Approche générale	Principales activités	Langage principal	Outil support
ARCADIA	Approche top-down, multi-vues	Analyse fonctionnelle, définition de exigences, architecture logique/physique	DSL compatible SysML	Capella, pour la modélisation.
OOSEM	Cycle en V, approche top-down	Analyse opérationnelle, définition de exigences, architecture logique/physique	SysML	Papyrus (Open source), MagicDraw, modélisation et generation de code
IBM Rational	Modèle en spirale : développement itératif et incrémental	Modélisation d'entreprise, exigences, analyse et conception, test	UML/SysML	Rational method composer, pour la modélisation et la génération de code

Conclusion sur les méthodes MBSE

Les méthodes MBSE supportent les différentes activités requises de l'ingénierie système. Ces méthodes utilisent le langage SysML ou un langage spécifique (DSL) compatible ou proche de SysML, comme la méthode ARCADIA développée par l'industriel Thales. Concernant l'évaluation des variantes d'un système et la recherche d'une solution optimale, les méthodes étudiées proposent d'évaluer séparément différentes variantes, ce qui impose de les modéliser chacune entièrement. La construction des solutions est faite manuellement, avec la possibilité de manquer une solution optimale (Broodney, 2012). Lorsque les variantes sont très nombreuses, cette problématique rentre dans la cadre de l'optimisation, et aucune méthode actuelle ne propose d'intégrer cette activité d'optimisation en tant de telle efficacement dans le MBSE. Il faut alors traiter ce problème d'optimisation séparément, à partir d'un modèle mathématique à part, et avec des outils d'optimisation dédiés. Les liens entre le modèle initial MBSE et ce modèle mathématique doivent être maintenus pour garantir la cohérence de l'ensemble. Ce besoin est également mentionné dans (Whittle, 2013), un article réalisant une étude des différents outils MBSE disponible sur le marché, ainsi que des besoins associés. La majorité des outils actuels supportent la transition depuis la conception jusqu'à la génération de code ou les tests de bas niveau. Cependant les analyses de haut niveau, réalisées très tôt dans le cycle en « V », et permettant de guider le concepteur dans ses choix d'architecture, sont peu explorées actuellement par les communautés de recherche. Avant d'examiner les travaux proposant un couplage entre le MBSE et l'optimisation, nous allons présenter la notion d'optimisation exploitable dans le domaine de l'ingénierie système dans la section suivante.

1.2.Optimisation et problèmes de satisfaction de contraintes

Dans cette section nous présentons la notion d'optimisation issue de deux approches : la programmation mathématique (Collette, 2013) et la programmation par contrainte (Rossi, 2006). Ces deux approches sont utilisables en ingénierie système, pour obtenir une architecture optimale. Puis nous proposons une formulation globale de ce problème d'optimisation englobant les deux approches, que nous appelons la formulation d'un problème « CSMOP », utilisée par la suite dans ce mémoire.

1.2.1. Optimisation issue de la programmation mathématique

Dans cette sous-section nous étudions les problèmes d'optimisation issus de la programmation mathématique. Le terme programmation fait référence à la description de la planification et de l'ordonnancement au sein de grandes organisations, dans les années 40 initialement (Fourer, 1990). Ces problèmes d'optimisation correspondent à la recherche du minimum ou du maximum (de l'optimum donc) d'une fonction donnée. Généralement les variables de la fonction à optimiser sont contraintes d'évoluer dans une certaine partie de l'espace de recherche. On parle alors de problème d'optimisation sous contraintes.

Pour le concepteur d'un système, ce besoin d'optimisation vient de la nécessité de concevoir un système répondant au mieux au cahier des charges, de manière à :

- Consommer le minimum d'énergie
- Disposer d'une masse minimale pour l'ensemble des composants matériels
- Offrir le maximum de performance

Classification des problèmes d'optimisation

On peut classer les différents problèmes d'optimisation en fonction de leurs caractéristiques.

- Type des variables de décision : réel, entier, mixte
- Type des fonctions objectifs : linéaire, non linéaires
- Un ou plusieurs objectifs : mono ou multi-objectifs

Optimisation mono-objectif

Un problème d'optimisation mono-objectif est défini par un ensemble de variables, une fonction objectif et un ensemble de contraintes. Une classification générale des méthodes mono-objectif est présentée par (Collette, 2013) et nous en présentons les principales dans la Figure 17 «Méthodes d'optimisation mono-objectif» :

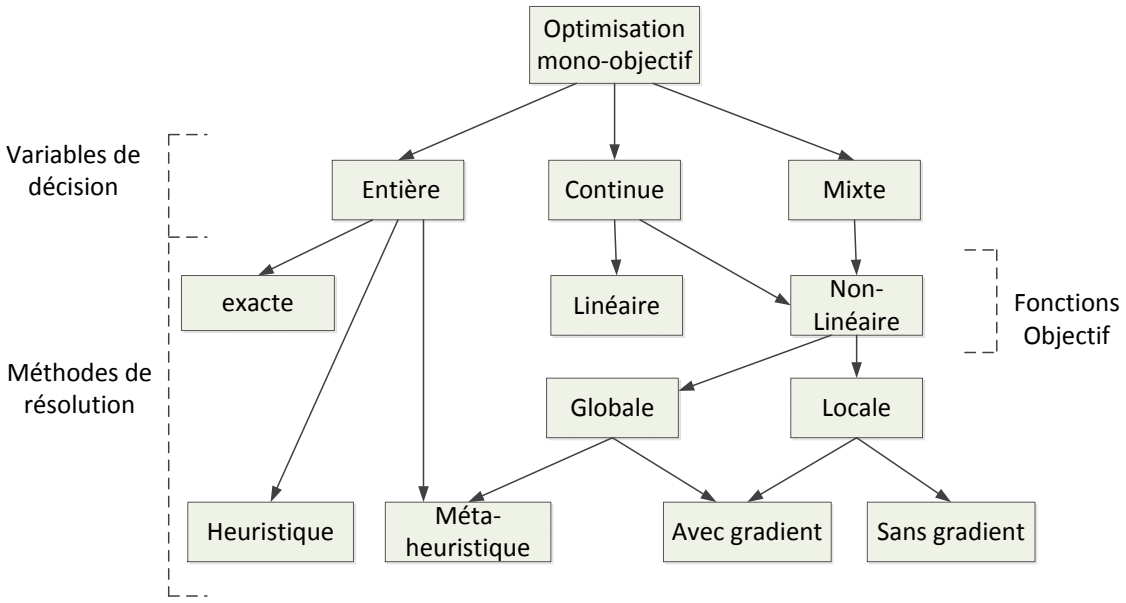


Figure 17 «Méthodes d'optimisation mono-objectif»

Pour l'optimisation en nombre entiers (variables entières) nous pouvons utiliser :

- Les méthodes exactes qui vont rechercher un optimum global. Pour cela, une énumération des solutions est effectuée comme dans la méthode « Branch and bound » intégrant les étapes suivantes : division de l'espace de recherche en sous-espaces, recherche d'une borne minimale associée à chaque sous espace, élimination des sous espaces ne respectant pas les contraintes. Ces étapes sont répétées jusqu'à obtenir un optimum global. Cette méthode n'est pas applicable à un espace de recherche trop grand.
- Une heuristique, qui est une méthode développée pour résoudre uniquement un problème particulier. Par exemple l'heuristique « Gloutonne » (Rossi, 2006) permet de résoudre le problème du voyageur de commerce sous certaines conditions.
- Une métaheuristique, qui est une méthode réutilisable permettant de résoudre au mieux un problème d'optimisation. Ces méthodes sont d'ailleurs utilisables également pour les problèmes avec des variables continues. Les métaheuristiques sont stochastiques et directes : il n'est pas nécessaire de déterminer le gradient de la fonction pour obtenir le résultat.

Pour les variables de décision continues ou mixte nous pouvons utiliser :

- Les méthodes issues de la programmation linéaire
- Pour les problèmes linéaires, les méthodes avec gradient exploitent le gradient d'une fonction pour déterminer un optimum local ou global.
- Les méthodes sans gradient incluent les méthodes constructives où l'on construit la solution une variable après l'autre et les méthodes stochastiques où une solution est recherchée dans le voisinage d'un point courant, quelle que soit la valeur de la fonction objectif.

Optimisation multi-objectif

Lorsque l'on modélise un système, on cherche souvent à satisfaire plusieurs objectifs. Le concepteur souhaite par exemple un système peu coûteux et avec le moins de défauts possible. Dans ce cas on parle de problème d'optimisation multi-objectif, dont nous avons présenté la définition dans l'introduction de cette thèse.

Une fonction $f_i(\boldsymbol{x})$ est appelée **fonction objectif**. On l'appelle également fonction de coût ou critère d'optimisation. Les **variables de décision** x_i sont regroupées dans le vecteur $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}}$. Les vecteurs $\boldsymbol{h}(\boldsymbol{x})$ et $\boldsymbol{g}(\boldsymbol{x})$ représentent respectivement le vecteur de contraintes d'égalité et le vecteur de contraintes d'inégalité. En général on trouve deux types de contraintes :

- Des contraintes de type $B_{inf_i} \leq x_i \leq B_{sup_i}$ permettent de délimiter un espace de recherche
- Des contraintes de type $\boldsymbol{c}(\boldsymbol{x}) \geq 0 \; ou \; \boldsymbol{c}(\boldsymbol{x}) \leq 0 \; avec \; \boldsymbol{c}(\boldsymbol{x}) \in \mathbb{R}^{\boldsymbol{m}}$ permettent de délimiter l'espace des valeurs réalisables

Pour résoudre un problème d'optimisation multi objectif on cherche à minimiser "au mieux" les différents objectifs. Bien souvent et en particulier lors de la conception d'un système, les objectifs à atteindre sont contradictoires. Deux objectifs sont contradictoires lorsque la diminution d'un objectif implique une augmentation de l'autre objectif. De ce fait lorsque l'on résout un problème d'optimisation multi-objectifs, on obtient la plupart de temps une multitude de solutions. Ces solutions ne sont pas optimales au sens où tous les objectifs du problème ne sont pas minimisés. La notion de compromis nous permet de classer les solutions selon une relation de dominance. Dans la Figure 18 «Dominance et front de Pareto» une solution A domine une solution C car A est au moins aussi bon dans tous les objectifs, et A est strictement meilleur que C dans au moins un objectif, le taux de défaut en l'occurrence.

$$\begin{cases} \forall i \in \{1..n\} \;\; f_i(\boldsymbol{A}) \leq f_i(\boldsymbol{C}) \\ \exists i \in \{1..n\} \;\; f_i(\boldsymbol{A}) < f_i(\boldsymbol{C}) \end{cases}$$

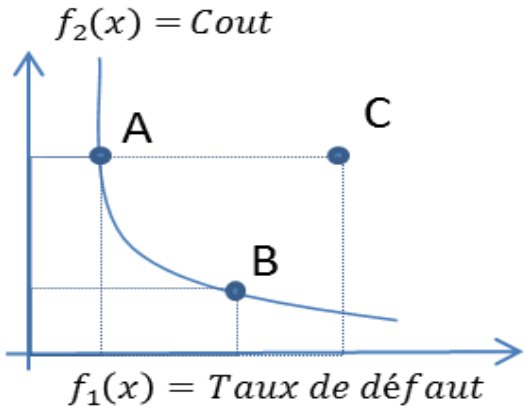


Figure 18 «Dominance et front de Pareto»

Les solutions qui dominent les autres mais ne se dominent pas entre elles sont les solutions optimales au sens de Pareto présenté sur la Figure 18 «Dominance et front de Pareto». Finalement, résoudre un tel problème consiste à obtenir les solutions optimales au sens de Pareto, charge ensuite au décideur de choisir l'une ou l'autre des solutions du front de Pareto. Il s'agit d'un outil d'aide à la décision. Plusieurs méthodes existent pour résoudre les problèmes d'optimisation multi-objectif (Hachimi, 2013). Nous présentons ici la méthode pondérée ainsi qu'un exemple de méthode interactive et une méta-heuristiques.

Méthode pondérée

Dans cette méthode, il s'agit de revenir à un problème mono-objectif, dont il existe de nombreuses méthodes de résolution. La manière la plus simple consiste à prendre chacune des fonctions objectifs et à leur appliquer un coefficient de pondération. On obtient alors une nouvelle fonction objectif de la forme :

$$\min\left[\sum_{i=1}^{i=n} w_i f_i(\boldsymbol{x})\right] \; with\, w_i \geq 0 \;\; and \sum_{i=1}^{i=n} w_i = 1$$

Cette résolution par pondération fait partie des méthodes « à priori », tout comme la méthode par but et la méthode « min-max », car le décideur doit affecter et donc connaitre à l'avance l'importance relative de chaque objectif. Un inconvénient de cette méthode est que les solutions potentielles situées dans les portions concaves de l'espace des solutions ne sont pas retenues. L'autre inconvénient de cette méthode est qu'il peut être délicat de combiner des objectifs très différents avec des coefficients. Une légère variation des coefficients va provoquer des variations importantes dans l'ordre des solutions obtenues. C'est problématique car le décideur n'est pas toujours en capacité de pondérer correctement des différents objectifs.

Méthodes interactives

Ces méthodes permettent de chercher un et une seule solution. L'utilisateur de ces méthodes va déterminer ses préférences par rapport à plusieurs compromis au cours de l'optimisation. Dans la méthode *ISWT* (Interactive Surrogate Worth Trade-off) (Chankong, 1978) ou méthode de compromis par substitution, un processus interactif est proposé. Le processus se décompose en sept étapes. On commence par résoudre un problème mono-objectif, en minimisant la fonction $f_{\mathrm{j}}(\boldsymbol{x})$ avec les contraintes de départ. Puis on choisit une valeur de départ. $\varepsilon_j \geq f_{j_{min}}, j = \{2, .., k\}$. Dans une troisième étape, nous allons chercher à minimiser $f_1(\boldsymbol{x})$ (fonction dite de référence) en rajoutant les $k-1$ contraintes suivantes : $f_2(\boldsymbol{x}) \leq \varepsilon_2, \ldots f_k(\boldsymbol{x}) \leq \varepsilon_k$. Le processus se poursuit jusqu'à demander à l'utilisateur une série de coefficients W_{1j} correspondants à son avis concernant l'accroissement de la fonction $f_{\mathrm{j}}(\boldsymbol{x})$ de $\lambda_{1j} = -\frac{\partial f_1}{\partial f_j}$ unités. Cet avis s'exprime par une valeur allant de -10 (avis défavorable) à +10 (avis favorable) en passant par 0 (indifférent). Ces coefficients permettent de

déterminer une zone de satisfaction à l'intérieur de laquelle la solution optimale va être recherchée. L'inconvénient de cette méthode est la détermination des coefficients W_{1j} par l'utilisateur qui peut être tenté de répondre au hasard, ainsi que le fait que la fonction $f_1(\boldsymbol{x})$ doit être dérivable.

Métaheuristiques

Les métaheuristiques sont des méthodes générales de recherche dédiées aux problèmes d'optimisation difficiles à résoudre. Ces méthodes stochastiques peuvent reprendre des concepts de la physique comme le recuit simulé ou de la biologie comme les algorithmes évolutionnaires. Les principales métaheuristiques sont le recuit simulé, la recherche tabou et les algorithmes génétiques dont nous allons décrire les principes. Ces méthodes exploitent des processus aléatoires pour explorer l'espace de recherche. Elles sont également adaptables à un grand nombre de problèmes sans changer d'algorithme de base. Une approximation de l'optimum global est obtenue par un processus itératif. Les algorithmes génétiques recherchent à partir d'une population initiale, les meilleurs individus par analogie avec la génétique. Pour résoudre un problème d'optimisation, un algorithme génétique ne va pas utiliser directement les variables de décision mais plutôt une séquence de codes représentant un individu. Une fonction d'adaptation (fitness function) permet d'évaluer les chances qu'un individu soit sélectionné ou pas, afin de reproduire de nouvelles solutions. Les contraintes sont prises en compte en pénalisant les individus qui ne les respectent pas. Deux mécanismes permettent d'explorer l'espace des solutions : le croisement et la mutation. Les différentes étapes d'un algorithme génétique sont l'évaluation, la sélection, puis le croisement/mutation. L'avantage d'une métaheuristique comme un algorithme génétique est qu'elle est capable d'optimiser un problème avec un nombre minimal d'informations. Par contre l'optimalité de la solution n'est pas garantie, et le codage des individus à partir des variables de décision ainsi que l'expression de la fonction d'adaptation ainsi que des opérateurs de croisement/mutation sont autant de réglages qui nécessitent une certaine expertise de la part de l'utilisateur. Les temps de calcul sont également un inconvénient de ces méthodes.

1.2.2. Programmation par contrainte et optimisation

La programmation par contrainte est une méthode de modélisation et de résolution de problèmes qui a émergé dans les années 1980. Cette technique est issue de la programmation logique et de l'intelligence artificielle. Initialement dédiée à la résolution de problèmes combinatoires elle a évolué ensuite vers la résolution de problèmes d'optimisation. Elle consiste à représenter un problème par des variables ainsi que par un ensemble de relations logiques, des contraintes. Ces contraintes s'appliquent aux variables pour définir une ou plusieurs solutions du problème. Un solveur va calculer une solution en instanciant chacune des variables avec une valeur satisfaisant l'ensemble des contraintes initiales. La démarche permet d'avoir une séparation entre un langage permettant de représenter le problème et les algorithmes employés pour le résoudre. Parmi les domaines d'application on peut citer le problème d'affectation de ressources, la planification, la vérification de spécifications ou le diagnostic de pannes.

Problèmes de satisfaction de contrainte (CSP)

Initialement dédiée à la résolution de problèmes à variables entières (*discrètes*), cette technique a évolué vers les problèmes à variable réelles (*continues*) puis vers les problèmes d'optimisation. Dans tous les cas, le problème à résoudre est modélisé en terme de problème de satisfaction de contrainte (CSP) et a pour définition (Brailsford, 1999) :

Définition 2 : Un problème CSP est un triplet $\{X, D, C\}$ composé de :

- Un ensemble de variables $X = \{x_1, x_2, .. x_n\}$,
- Un ensemble de domaines $D = \{D_1, D_2, .. D_n\}$ $avec$ $x_i \in D_i$,
- Un ensemble de contraintes $C = \{C_1, C_2, .. C_m\}$ où C_i définit un sous-ensemble du produit cartésien des domaines : $C_i(x_{i1}, x_{i2}, .. x_{im}) \subseteq D_{i1} \times D_{i2} \times .. \times D_{im}$.

$X(C_i) = \{x_{i1}, x_{i2}, .. x_{im}\}$ est appelé le scope de C_i et m est l'arité de la contrainte.

Dans ce mémoire, nous ne traiterons que des CSP statiques, pour lesquels l'ensemble des contraintes ne change pas pendant la résolution du problème.

Le domaine d'une variable désigne l'ensemble de valeurs que peut potentiellement prendre cette variable. Cette notion fait référence à des ensembles de nature très différente dans le contexte de l'ingénierie système :

- Un ensemble de valeurs énumérées. Par exemple on peut vouloir représenter l'allocation de deux traitements logiciels s_1 et s_2 à un CPU parmi trois, e.g.

$D_1 = D_2 = \{ CPU_1, CPU_3, CPU_5 \}$ avec deux variables $X = \{x_1, x_2\}$,

$$x_1 = CPU_3 \Leftrightarrow s_1 est\ allou\acute{e}\ \grave{a}\ CPU_3$$

- Un ensemble d'entiers non contigus, pour représenter un choix d'instance d'un bloc B_1 parmi une liste d'instances $I = \{i_1, i_2 \ldots i_n\}$ disponibles « sur étagère » :

$D = \{ 5, 8, 12 \}$ avec une variables x_1,

$$x_1 = 5 \Leftrightarrow B_1\ est\ instanci\acute{e}\ par\ i_5\ dans\ le\ syst\grave{e}me$$

- Un intervalle d'entiers pour représenter par exemple le niveau de redondance d'un composant. Dans ce cas on définit un ensemble d'entiers contigus, en ne spécifiant que les bornes inférieure et supérieure de l'intervalle, par exemple $D = \{ 1, .., 4 \}$ pour spécifier un niveau de redondance de 1 à 4.
- Un intervalle de réels, représentant par exemple l'angle d'inclinaison d'une caméra embarquée dans un véhicule. $D = [-\pi / 2, \pi / 2]$.

Lorsque la valeur de la variable est à chercher dans $\mathbb{N}$, dans $\mathbb{Z}$, ou dans tout ensemble discret de valeurs, on parlera de variable discrète. Si sa valeur est à trouver dans $\mathbb{R}$ ou dans tout

intervalle continu de valeurs, on parlera de variable continue. En fonction du type des variables du problème, on distingue trois grandes catégories de problèmes CSP : selon qu'il ne contient que des variables discrètes, que des variables continues, ou bien à la fois des variables discrètes et continues, on dira d'un *CSP* qu'il est ***discret, continu* ou *mixte***. Les contraintes relient les variables les unes aux autres, réduisant ainsi les combinaisons de valeurs autorisées. Il existe des contraintes de différentes natures et dans la suite de ce mémoire, nous nous intéresserons aux contraintes suivantes directement applicables à l'ingénierie système :

- Les contraintes en intension. Il s'agit d'une expression arithmétique linéaire ou non-linéaire pour définir une relation entre plusieurs variables, e.g. $x + y \leq 2$
- Un prédicat portant sur n variables comme $isDifferent(x_1, x_2, x_3)$ pour indiquer que trois traitements logiciels ne peuvent être alloués à un même CPU.

Les autres types de contraintes non étudiées dans cette thèse sont les contraintes en extension spécifiant des tuples de valeurs, les contraintes dynamiques évoluant en cours de résolution, les contraintes par morceaux et les contraintes douces pouvant ne pas être respectée. Une solution d'un *CSP* est un *n*-uplet $S = (s_1, s_2, \ldots, s_n)\ avec\ s_i \in D_i \forall i$, pour lequel toute contrainte $C_j \in C$ est satisfaite : aucune contrainte $C_j(x_{j1}, x_{j2}, \ldots x_{jm})$ n'est violée quand chacune des variables x_{ji} sur lesquelles elle porte se voit affecter sa valeur correspondante s_{ji} dans la solution.

Méthodes de résolution d'un problème CSP

Le filtrage des domaines est la première technique permettant de résoudre un problème CSP et ce processus est décrit dans (Brailsford, 1999). En définissant certaines propriétés que doivent respecter les solutions du problème, il est possible d'éliminer des valeurs dans les différents domaines des variables. L'une de ces propriétés, la *consistance*, permet de différencier les valeurs que l'on garde de celles que l'on élimine. Lorsque toutes les contraintes d'un problème CSP sont binaires, les variables et les contraintes peuvent être représentées par un graphe dont les sommets sont les variables et les arrêtes des contraintes entre deux variables. Pour les contraintes non binaires, un hyper-graphe peut également être défini. Si il y a une contrainte C_{ij} entre les variables x_i et x_j , alors l'arc orienté (x_i, x_j) est consistant si pour chaque valeur $a \in D_i$ il existe une valeur $b \in D_j$ telle que l'affectation $x_i = a\ et\ x_j = b$ satisfasse la contrainte C_{ij}. Une valeur $a \in D_i$ ne respectant pas cette propriété peut être éliminée du domaine D_i. Si tous les arcs d'un problème CSP sont consistants, le problème entier est dit arc-consistant et sera plus facile à résoudre. Pour rendre un problème arc-consistant, plusieurs algorithmes sont disponibles dont une version optimale avec l'algorithme AC7 (Bessière, 1999) disposant d' une complexité temporelle en $O(ed^2)$, où e est le nombre de contraintes et d la taille du plus grand domaine. L'arc consistance généralisée est une extension de cette notion pour les contraintes d'ordre supérieur à 2 comme la contrainte « All-different » et dans ce cas d'autres algorithmes de filtrage sont possibles (Régin, 1994).

Une fois le filtrage des domaines effectué, l'étape suivante est l'exploration de l'espace de recherche, car nous pouvons avoir encore plusieurs valeurs dans chaque domaine. Plusieurs techniques existent pour effectuer cette exploration. L'idée de base est de générer une affectation des variables du problème, de tester cette affectation puis d'en générer une autre, la tester, etc. En prenant les variables l'une après l'autre, cette recherche est semblable à une recherche arborescente. Chaque nœud de l'arbre correspond à une solution partielle, pour laquelle un sous-ensemble de variables est affecté et l'autre non. En choisissant une branche de l'arbre, l'algorithme instancie une variable. Une impasse est détectée lorsque le domaine d'une variable restant à instancier (une variable future) devient vide. Les différents algorithmes se distinguent par le choix de la variable future. Nous présentons ici les trois plus courants : retour-arrière, recherche en avant et MAC.

Dans l'algorithme de retour-arrière ou « backtracking », on affecte à une variable courante une valeur de son domaine. Cette affectation est vérifiée avec la solution partielle courante. Si une contrainte impliquant la variable courante et les variables déjà instanciées est violée, l'affectation de la variable courante est abandonnée au profit d'une autre. Si toutes les valeurs sont épuisées, l'algorithme procède à un retour-arrière vers la précédente variable et lui affecte une nouvelle valeur. L'algorithme se termine lorsque toutes les variables disposent d'une affectation, ou bien il peut continuer pour trouver une autre solution. S'il n'y a pas de solutions, l'algorithme se termine lorsque toutes les possibilités sont testées.

Tandis que l'algorithme de retour arrière vérifie les contraintes entre la variable courante et les variables déjà instanciées, les algorithmes de recherche en avant et MAC vérifient les contraintes entre la variable courante et les variables futures. Dans l'algorithme de recherche en avant (Bessière, 2002), quand une valeur est affectée à la variable courante, toutes les valeurs d'une variable future qui ne respectent pas les contraintes sont temporairement éliminées du domaine. Si le domaine de cette variable devient vide, on peut tout de suite passer à une autre valeur pour la variable courante ou bien procéder à un retour-arrière, ce qui est un avantage. De ce fait les impasses dans la recherche arborescente sont plus rapidement détectées par rapport à l'algorithme de retour-arrière.

L'algorithme MAC (Maintaining Arc Consistency) et sa version améliorée (Sabin, 1997) va plus loin dans la recherche en avant. Lorsque l'affection de la variable courante est réalisé, le sous-problème correspondant est rendu arc-consistant par la procédure que nous avons décrit précédemment. Puis les valeurs des variables sont également vérifiées, de la même façon de pour l'algorithme de recherche en avant, mais avec de meilleures performances en raison du filtrage.

CSP continus et mixtes

L'arithmétique des intervalles permet d'étendre aux domaines continus les notions de filtrage par consistance pour les problèmes CSP (Berger, 2010), que nous avons présentés pour le domaine discret. L'idée de l'arithmétique des intervalles est de remplacer les calculs sur

les nombres flottants par des calculs sur des paires de nombres. Cette notion a été intégrée dans les langages de programmation logique comme *Prolog* par (Older, 1990), et il est possible de raisonner sur des problèmes CSP dont les variables sont de type réel, en exprimant ces variables sous forme d'un encadrement avec une borne inférieure et une borne supérieure. (Moore, 1979) développe également une arithmétique propre aux intervalles, en donnant leur principales caractéristiques comme la notion d'enveloppe et l'extension aux intervalles d'une fonction réelle. Ces notions permettent de définir de nouvelles techniques de filtrage dédiées aux problèmes CSP continus, en se basant non plus la consistance d'arc mais la consistance d'enveloppe pour une contrainte donnée. Différents algorithmes de filtrage basés sur la consistance d'enveloppe sont proposés dont l'algorithme AC4 (Goualard, 2000). Lorsque le problème CSP est mixte, il est possible de réaliser le filtrage des domaines en associant aux variables entières des intervalles de variables réelles.

Une fois que le filtrage des domaines est réalisé, les algorithmes d'explorations doivent également être adaptés aux variables continues. Plutôt que de procéder à une énumération (cas des variables discrètes) ces algorithmes vont procéder à une bissection des intervalles, en coupant en deux moitiés le domaine d'une variable lors de l'exploration arborescente. Chaque moitié est alors explorée dans une branche de l'arbre de recherche, les autres moitiés étant conservées à l'identique. Cet algorithme dit de *branch-and-prune* est présenté dans (Van Hentenryck, 1997).

Variante d'un CSP pour l'optimisation

Selon les situations, la résolution d'un *CSP* consiste à montrer qu'une ou aucune solution existe, à trouver une solution ou l'ensemble des solutions du problème. Lorsque les solutions peuvent être évaluées avec des fonctions objectifs, on peut également rechercher l'ensemble des solutions optimales du problème *CSP*. Dans ce cas la définition d'un CSP est augmentée d'une fonction objectif et il est possible de résoudre le problème en utilisant la méthode suivante (Rossi, 2006) :

On suppose que le problème CSP est enrichi d'une variable c et d'une contrainte dite d'objectif $c = f(X)$ avec f la fonction objectif à minimiser et X l'ensemble des variables de décision du problème CSP. Nous pouvons résoudre ce problème d'optimisation en résolvant une série de problèmes CSP, par une variante de l'algorithme « Branch-and-bound ». Tout d'abord une solution initiale au problème CSP est obtenue avec l'algorithme simple retour arrière (Backtracking). Une contrainte de la forme $c < f(S)$ est alors rajoutée au problème CSP, excluant les solutions qui ne sont pas meilleures que la solution initiale. Une nouvelle solution est alors recherchée pour ce problème, jusqu'à ce que aucune solution ne soit trouvée. La dernière solution trouvée est alors optimale.

1.2.3. Formulation générale CSMOP

Il est possible d'englober la définition d'un problème d'optimisation multi-objectif issue de la programmation mathématique ainsi que celle issue des CSP.

Dans ce cas on parle de problème CSMOP (Constraint Satisfaction Multi-criteria Optimization Problem) . La résolution de ce problème (I.B. Jaâfar, 2004) consiste à déterminer parmi un ensemble de solutions admissibles et en présence d'objectifs multiples (les critères), les meilleures solutions par rapport aux objectifs considérés. Contrairement à une approche mono-critère, c'est le concepteur qui choisit une solution parmi le front de Pareto.

Définition 3 : Un problème CSMOP dans sa forme générale est un quadruplet $\{X, D, C, F\}$ composé de :

- Un ensemble de variables $X = \{x_1, x_2, .. x_n\}$,
- Un ensemble de domaines $D = \{D_1, D_2, .. D_n\}\ avec\ x_i \in D_i$,
- Un ensemble de contraintes $C = \{C_1, C_2, .. C_m\}$ où C_i définit un sous-ensemble du produit cartésien des domaines : $C_i(x_{i1}, x_{i2}, .. x_{im}) \subseteq D_{i1} \times D_{i2} \times .. \times D_{im}$.
- Un ensemble de fonctions objectifs $F = \{f_1, f_2, .. f_k\}$ ou$\forall i\ f_i$ est une fonction numérique définie sur un sous-ensemble de X

Nous considérons comme solution d'un problème CSMOP l'ensemble des affectations non-dominantes satisfaisant l'ensemble des contraintes C.

Cette définition va nous servir dans le reste de cette thèse, car c'est le type de définition que nous cherchons à générer à partir d'un modèle enrichi du système.

1.2.4. Outils pour l'optimisation

Cette sous-section présente différents outils de résolution de problème d'optimisation de type CSMOP. Nous nous intéressons aux outils non commerciaux, car nous voulons avoir accès au code source de ces outils. Nous souhaitons pouvoir également intégrer les outils retenus dans un environnement Eclipse/JAVA pour pouvoir les interfacer avec un outil de modélisation système comme Papyrus (Papyrus, 2015) . Nous pouvons classer les outils d'optimisation en trois catégories :

- Des outils boites boîte-noire comme Minion (Gent, 2006) ou RealPaver (Granvilliers, 2006) prenant en entrée un fichier contenant le modèle écrit dans un langage dédié à la modélisation. Ces outils ne permettent pas de s'interfacer facilement avec les outils de modélisation système, nous ne les retenons donc pas.
- Des interpréteurs, utilisant des modèles définis sous la forme de programmes écrits dans un langage déclaratif comme Prolog étendu. Dans cette catégorie nous trouvons l'outil et le langage du même nom ECL^{i}PS^e (Apt, 2006) dont l'origine remonte aux années 1990. Cet outil est également difficilement intégrable dans un environnement ECLIPSE/JAVA. De plus d'après les tests réalisés dans (Berger, 2010) ECL^{i}PS^e est difficilement opérationnel lorsqu'il s'agit de résoudre un problème continu ou mixte.

- Des bibliothèques d'objets et de fonctions dédiés au CSP ou aux problèmes d'optimisation, écrits dans un langage comme Java ou Python, permettant d'exprimer les modèles sous forme de programmes compilables dans ce langage. C'est cette catégorie que nous avons explorée, avec notamment les outils Labix, PyOpt, Choco. Labix (Niemeyer, 2016) est une bibliothèque de résolution de problème CSP en Python, aux performances limitées mais avec une mise en œuvre simplifiée.

PyOpt

PyOpt (Perez, 2012) est un logiciel libre dédié à l'optimisation, orienté-objet et développé avec le langage Python. Il permet de formuler et de résoudre les problèmes d'optimisation non-linéaires, mono et multi-objectif. PyOpt intègre lui-même une multitude d'algorithmes d'optimisation, développé dans différents langages dont C++ et fortran. Les concept objet d'héritage et de surcharge d'opérateurs permettent d'offrir une interface commune entre l'application intégrant PyOpt et les différents algorithmes disponibles. Les principales caractéristiques de PyOpt sont les suivantes :

- Une séparation entre la formulation du problème (Figure 19 « Interface de PyOpt »)avec le langage Python et sa résolution par un algorithme spécifique écrit dans un autre langage
- Possibilité d'intégrer d'autres algorithmes d'optimisation
- Compatible avec différentes plateformes comme Windows, Linux et OS X ainsi qu'avec l'environnement Eclipse où il est possible d'intégrer le langage Python via le plugin PyDev (Ertl, 2009)
- Parallélisation possible des algorithmes
- Optimisation multi-objectifs disponible, avec un algorithme génétique NSGA-2 uniquement.

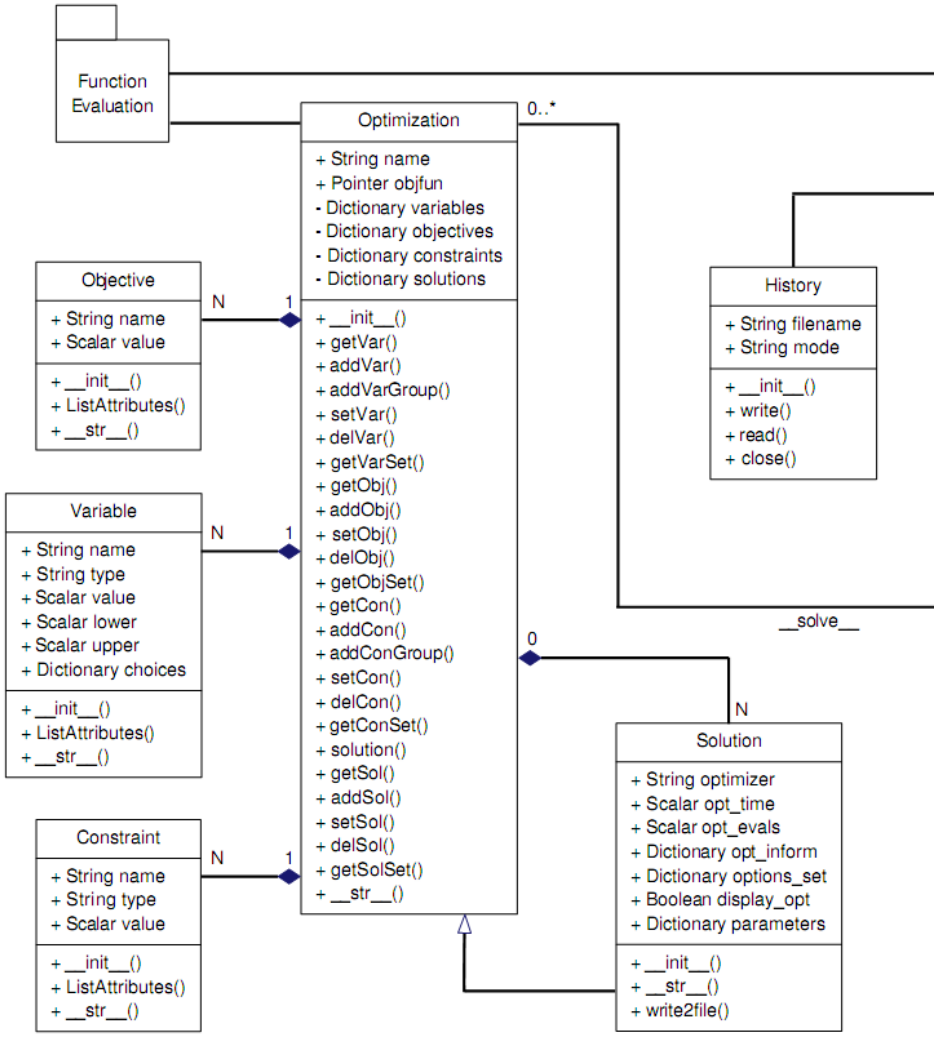


Figure 19 « Interface de PyOpt »

Choco

Choco (Jussien, 2008) (CHOCO, 2016) est une bibliothèque écrite en JAVA permettant la modélisation et la résolution de problèmes sous contraintes, développée par l'Ecole des Mines de Nantes. Il est possible de rajouter des extensions à Choco, qui dispose également d'une gestion optimisée de la propagation des contraintes. La modélisation des problèmes continus est rendue possible avec l'instanciation de variables de type réel. Cependant les limites de Choco par rapport aux problèmes continus se trouvent dans les opérations arithmétiques proposées, trop peu nombreuses par rapport à PyOpt par exemple. Il est également possible avec Choco de résoudre un problème d'optimisation sous contrainte, en intégrant une fonction à optimiser dans la définition du problème sous contraintes à l'aide d'une variable (Algorithme 1 : « déclaration d'une variable d'optimisation avec Choco »). Dans ce cas, les solutions optimales seront extraites lors de la résolution. Choco peut résoudre un problème d'optimisation à deux objectifs, il utilise alors une démarche simplifiée consistant à rajouter une contrainte à chaque étape de résolution du problème mono-objectif.

```
// Function to maximize: 3X + 4Y
IntVar OBJ = VF.bounded("objective", 0, 999, solver);
solver.post(ICF.scalar(new IntVar[]{X,Y}, new int[]{3,4}, OBJ));
solver.findOptimalSolution(ResolutionPolicy.MAXIMIZE, OBJ);
```

Algorithme 1 : « déclaration d'une variable d'optimisation avec Choco »

Tableau comparatif des solveurs

			Minion	PyOpt	Labix	Choco	ECL^{i}PS^e
Langage			matri-ciel	Python	Python	Java	ECL^{i}PS^e
Domaine/ Variables	Booléen		+	+	-	+	+
	Entier		+	+	+	+	+
	Réels		-	++	-	+	+
	Ensemble de variables		+	+	-	++	+
Contraintes							
	arithmétique	entière	+	+	+	+	+
		réelle	-		-		
	globale		+	-	-	++	+
Optimisation			non	oui	non	oui	non
	Mono/Multi-objectif		-	multi ++(gén.)	-	multi +	-
	Différents algorithmes disponibles		-	++	-	+	-
Intégration possible dans l'environnement Eclipse/Java			-	+	+	++	-
Extension possible du solveur			-	++	-	++	+

Tableau 1 « Comparaison de différents solveurs CSMOP »

Conclusion sur les solveurs

Labix est un solveur facile à mettre en œuvre, nous le retenons pour nos premières expérimentations. PyOpt permet de mettre en œuvre les variables réelles et il permet d'optimiser des fonctions non-linéaire. Choco est un solveur performant pour les problèmes combinatoires (variables entières), et il dispose d'une bibliothèque de contraintes très riche, notamment pour les contraintes globales. Nous retenons donc les deux solveurs PyOpt et Choco pour nos expérimentations.

1.3.Les approches existantes concernant l'optimisation et le MBSE

Dans cette section nous étudions les différents travaux combinant optimisation, l'ingénierie système et le MBSE. La première approche « RAP » est antérieure au MBSE, et combine ingénierie système et optimisation. Les autres approches présentées associent l'optimisation et le MBSE. Les systèmes où la partie logicielle est prépondérante, n'incluant pas de composants physiques, sont également passés en revue.

1.3.1. L'optimisation appliquée aux systèmes, le problème RAP

Le problème RAP (*Redundancy Allocation Problem*) consiste à optimiser la fiabilité ainsi que divers paramètres comme le coût ou la masse pour un système constitué de composant sur étagère, avec une structure série-parallèle Figure 20 « Problème RAP ». Chaque composant est caractérisé par un coût, une fiabilité, une performance. Il s'agit de déterminer une stratégie pour identifier quel composant utiliser et quel niveau de redondance lui appliquer pour atteindre les objectifs globaux du système.

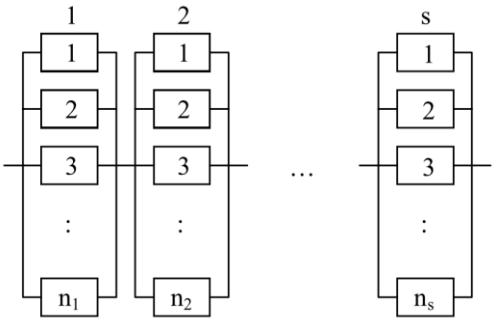


Figure 20 « Problème RAP »

Ce problème est difficile à résoudre compte tenu du grand nombre de solutions possibles, et l'analyse de la complexité de ce problème montre qu'il est NP-difficile même lorsque les contraintes sont uniquement linéaires. (Chern, 1992)

Le problème RAP, dans sa formulation générale (Coit, 1995) , se compose de s sous-systèmes en série. Chaque sous-système $i = (1,2..s)$ se est constitué de n_i composants dont les caractéristiques (fiabilité, coût et masse) sont connus. Pour chaque sous-système, une liste de m_i composants interchangeables est disponible. De plus il est possible pour chaque sous-système de fiabilité R_i et de coût C_i, d'utiliser plusieurs fois le même composant, plutôt que d'utiliser un seul composant plus fiable. Il s'agit donc de déterminer la combinaison optimale de composants utilisés ainsi que le niveau de redondance de chaque composant pour obtenir le système le plus fiable possible et à moindre coût. Le premier objectif à maximiser est donc la fiabilité, données par :

$$\max R = \prod_{i=1}^{i=s} R_i(\boldsymbol{x}_i) \; avec \; \boldsymbol{x}_i = (x_{i1}, x_{i2}, x_{i,m_i}), n_i = \sum_{j=1}^{m_i} x_{ij}$$

Avec les variables de décision suivantes :

$$x_{ij} \in \{0,1..n_i\} = nombre\ d'instances\ du\ composant\ j\ dans\ le\ sous\ système\ i$$

Le second objectif est par exemple le coût global du système :

$$\min C = \sum_{i=1}^{s}\sum_{j=1}^{m_i} c_{ij}x_{ij}\ \ avec\ c_{ij}\ le\ coût\ du\ j^{ème}\ composant\ pour\ le\ sous\ système\ i.$$

Le problème RAP a été résolu dans un certain nombre de formes particulières. La programmation dynamique, utilisant une fonction (un *Lagrangien*) associée au problème, a d'abord permis de le résoudre, lorsque le problème est limité à un seul composant par sous-système. Dans une seconde variante, un objectif unique à minimiser a été introduit, en utilisant une contrainte supplémentaire. Il est alors possible de résoudre le problème en utilisant une formulation en nombre entier et de le résoudre comme un problème de type sac à dos (Bulfin, 1985). Cependant la formulation en nombre entier ne permet pas de réaliser une optimisation multi-objectifs. La mise œuvre d'algorithmes génétiques a été expérimentée par (Coit, 1995) pour résoudre ce problème, permettant d'explorer un plus grand espace de recherche par rapport à la programmation dynamique et la programmation en nombre entier. L'inconvénient de cette approche pour l'utilisateur est de devoir ajuster expérimentalement un certain nombre de paramètres, nécessaires aux opérations de mutation et de sélections des différentes solutions. De plus, en raison du problème de la *dérive génétique*, la convergence vers un optimum global n'est pas garantie. Dans ce cas, une convergence prématurée se met en place autour d'un optimum local.

Dans les approches décrites précédemment, il n'y a pas de modèle d'architecture décrivant le système, il s'agit d'un problème d'optimisation traditionnel, difficilement applicable dans un contexte MBSE. Une approche permettant de déterminer les configuration optimales d'un système à partir d'un « feature models » , en utilisant le problème « RAP » est proposée par (Limbourg, 2008). Cette approche propose également de résoudre le problème RAP de façon récursive, ce que permet les « feature models ». De plus, dans le problème RAP initial, il n'est pas possible de rajouter des contraintes transverses entre les composants, comme par exemple un choix exclusif entre un composant très fiable ou deux composant moins fiables en parallèle. A partir d'un « feature models » représenté par une structure arborescente, l'algorithme proposé par (Limbourg, 2008) permet de générer un diagramme de blocs de fiabilité (RBD) sans niveau hiérarchique. L'optimisation de la fiabilité de ce système représenté par ce RBD correspond à un problème RAP. Cette approche permet donc de combiner optimisation et MBSE, mais elle est dédiée aux « feature models » permettant de modéliser une ligne de produits avec une structure arborescente.

1.3.2. Model Center, Consol Optcad (Spyropoulos)

ModelCenter

Dans l'approche « ModelCenter » présentée dans (Min, 2011) et (Kerzhner, 2011), une connexion entre SysML et l'environnement ModelCenter (devenu depuis un outil commercial du

même nom) est proposée. ModelCenter permet de modéliser différentes analyses sous forme de boite noire, en intégrant un certain nombre d'outils pour exécuter ces analyses. En partant d'un modèle SysML, le concepteur modélise la structure du système en utilisant les digrammes de bloc BDD et IBD, puis il rajoute un modèle d'analyse sous forme de diagramme paramétrique. Dans ce diagramme, les différents blocs de contraintes représentent une équation et sont affectés du stéréotype « ModelCenter » pour signifier qu'ils peuvent être analysés avec le framework. Dans l'exemple présenté dans la Figure 21 «Approche ModelCenter », il s'agit d'optimiser le coût, la masse et la performance (la force développée) d'un système hydraulique. Le concepteur dispose d'un grand choix de composants (cylindre, moteur, valve) dont les caractéristiques sont connues. Il s'agit donc d'un problème combinatoire, mais dont les fonctions objectifs sont non-linéaires et complexes.

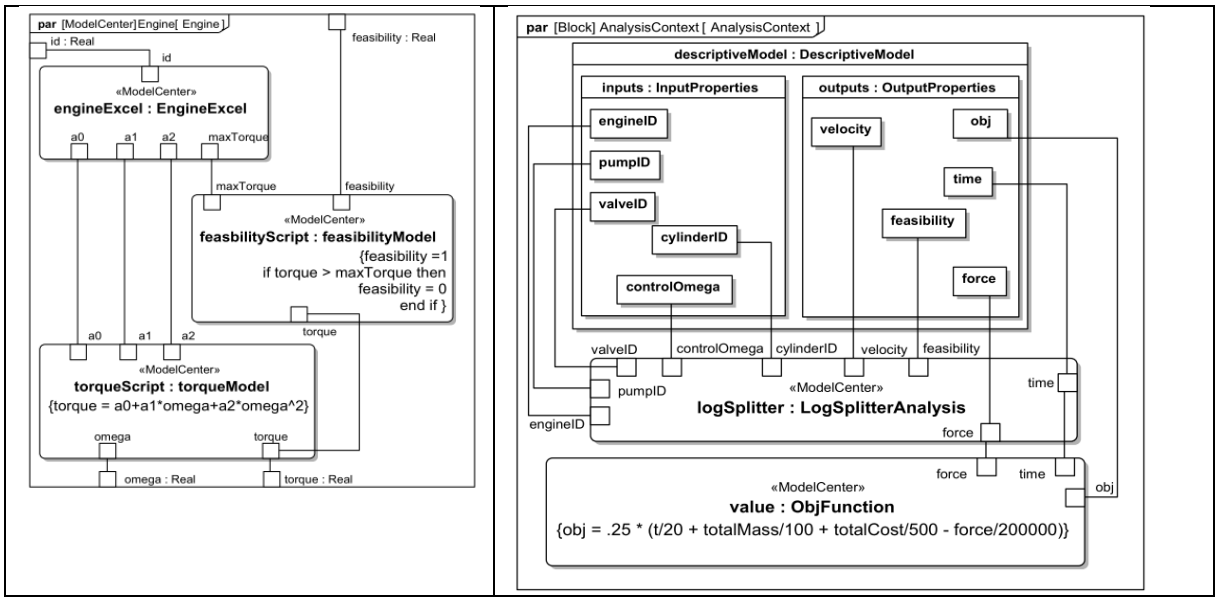


Figure 21 «Approche ModelCenter »

Pour résoudre le problème (Kerzhner, 2011), l'approche « ModelCenter » utilise une analyse MINLP (Mixed Integer Non-linear Programming), adaptée à un problème mono-objectif. Un contexte d'analyse permet de conserver une solution optimale dans le modèle SysML.

Consol-Optcad

L'approche proposée par (Spyropoulos, 2013) permet également de réaliser des analyses de compromis avec SysML, par intégration de l'outil Consol-Optcad et de son langage spécifique. Cet outil d'optimisation multi-objectifs a été développé par l'université du Maryland. Il permet une interaction avec l'utilisateur dans la phase d'optimisation, ce qui facilite la prise en compte de plusieurs objectifs. Il peut gérer les variables continues ou discrètes, ainsi que les fonctions non-linéaires. Les concepts du langage Consol-Optcad sont intégré sous forme de stéréotypes dans un profile SysML dédié. Des règles de transformations permettent de générer le code Consol-Optcad à partir de SysML, implémenté sous forme de plug-in dans l'outil commercial MagicDraw. Le plug-in permet de générer un fichier de description du problème dans le langage Consol-Optcad. Un cas d'étude associé à la méthode est étudié dans

(Spyropoulos, 2013). Il s'agit d'une centrale de production d'énergie thermique, pour laquelle plusieurs sources d'énergies sont disponibles. On souhaite déterminer les instants d'activation (nombres réels) et la puissance générée par chacune des sources pour optimiser le coût, la demande en énergie et les émissions du système. Certaines fonctions objectifs sont linéaires et d'autre quadratiques. Un diagramme paramétrique est utilisé pour faire la liaison entre les attributs du modèle à optimiser et Consol-Optcad. Dans l'outil d'optimisation, un algorithme quadratique est utilisé pour obtenir les solutions optimales par itérations successives à partir de conditions initiales. Le résultat des itérations sont présentés à l'utilisateur Figure 22 « Progression de la solution avec Consol-Optcad ». Pour chaque itération les valeurs courantes des objectifs sont présentées ainsi que les contraintes associées.

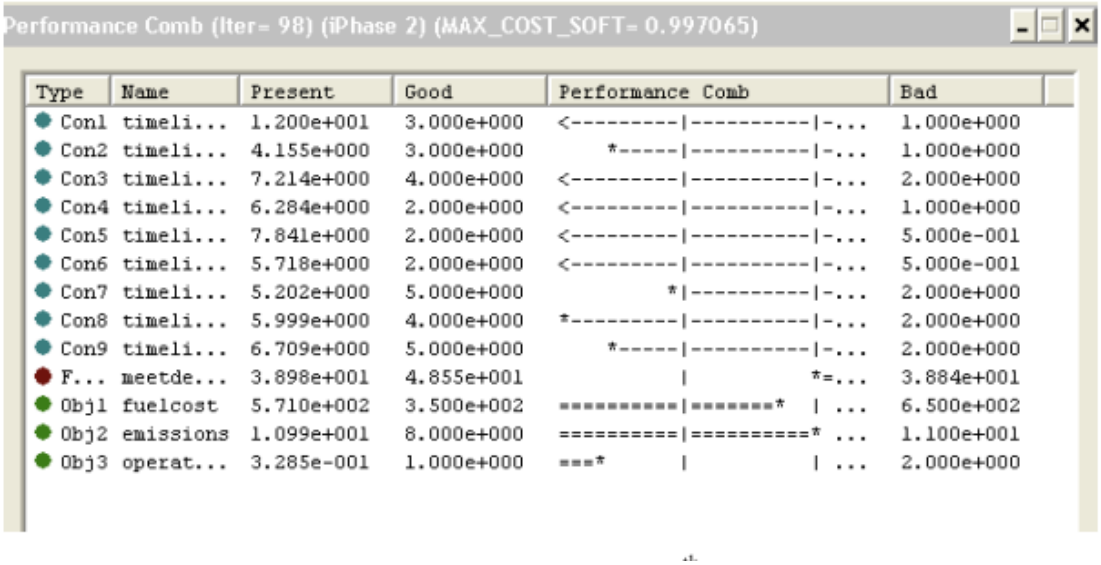


Figure 22 « Progression de la solution avec Consol-Optcad »

1.3.3. DSL et Algorithme génétique (Van Huong)

(Van Huong, 2012) propose un DSL (Domain Specific Language) permettant de décrire l'architecture d'un système embarqué et de procéder à son optimisation. Ce DSL et les variables de décision du problème d'optimisation sont présentés Figure 23 « DSL et paramètres à optimiser ». Les différents objectifs à optimiser concernent la puissance dissipée par le système, la surface occupée et le coût du système. Il s'agit de fonctions linéaires et quadratiques, avec des variables de décision entières. Un objectif global est obtenu par pondération des différents objectifs. Le problème est résolu en utilisant un algorithme génétique. Pour utiliser cet algorithme, l'utilisateur renseigne de nombreux coefficients permettant de calculer le génome à partir des variables de décision. Des itérations successives permettent d'approcher une solution optimale et un outil dédié permet d'approcher cette solution. Cette méthodologie semble difficile à adapter à un autre système, car elle est très liée au DSL proposé. De plus les fonctions objectifs ne sont pas modélisée, de même que les contraintes. L'utilisation exclusive d'un algorithme génétique peut s'avérer problématique pour les temps de calcul ainsi que les choix de paramètres pour le calcul du génome.

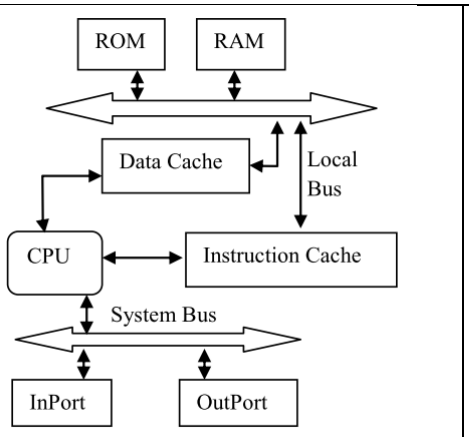


Fig. 1. Architecture of embedded system

Components	Parameters	Symbol	Unit of measurement
CPU	Frequency	x_1	MHz
RAM/ROM	Size	x_2, x_3	MB
Instruction Cache/Data Cache	Size	x_4, x_5	KB
Local bus	Address	x_6	Bit
	Data	x_7	Bit
System bus	Data	x_8	Bit
	Address	x_9	Bit
InPort	Width	x_{10}	Bit
OutPort	Width	x_{11}	Bit

Figure 23 « DSL et paramètres à optimiser »

1.3.4. OOSEM et la recherche des meilleures variantes (Broodney)

Les auteurs de (Broodney, 2012) et (Masin, 2014) proposent une approche dont l'objectif est de compléter le MBSE pour mieux maîtriser la complexité des systèmes en aidant le concepteur dans ses choix d'architecture. Une méthode est proposée permettant de formaliser les exigences, de définir les variantes de conception, et de procéder à l'exploration des solutions. Cette méthode comporte quatre étapes :

1. La formalisation des exigences. Les fonctions sont déterminées puis connectées aux différents composants logiques du système. Les objectifs à minimiser/maximiser comme le coût, la masse ou la fiabilité sont définis par des équations, graphiquement ou textuellement, ainsi que les contraintes s'appliquant au système.
2. Définir les variantes de conception en utilisant les concepts de variabilité, en utilisant une base de données de composants conjointement au modèle du système
3. Générer automatiquement le modèle mathématique du problème d'optimisation, résoudre le problème pour obtenir les solutions.
4. Analyser les solutions obtenues, mettre à jour le modèle avec les solutions ou bien modifier les exigences pour recommencer l'étape d'optimisation.

Dans cette approche, la modélisation des variantes n'est pas réalisée avec un langage comme CVL jugé trop complexe avec son approche arborescente qui n'est pas jugée nécessaire dans un contexte d'optimisation. Une modélisation dite « Concise » de la variabilité est utilisée. Cette modélisation permet de faire le lien entre des blocs abstraits SysML et des tables de composants (les instances de bloc) avec leurs différents attributs. Pour cela des stéréotypes sont appliqués aux blocs SysML (Figure 24 « Modélisation concise pour l'optimisation ») pour indiquer que la valeur des instances de ces blocs ou bien celle des attributs est à rechercher dans des tables externes au modèle.

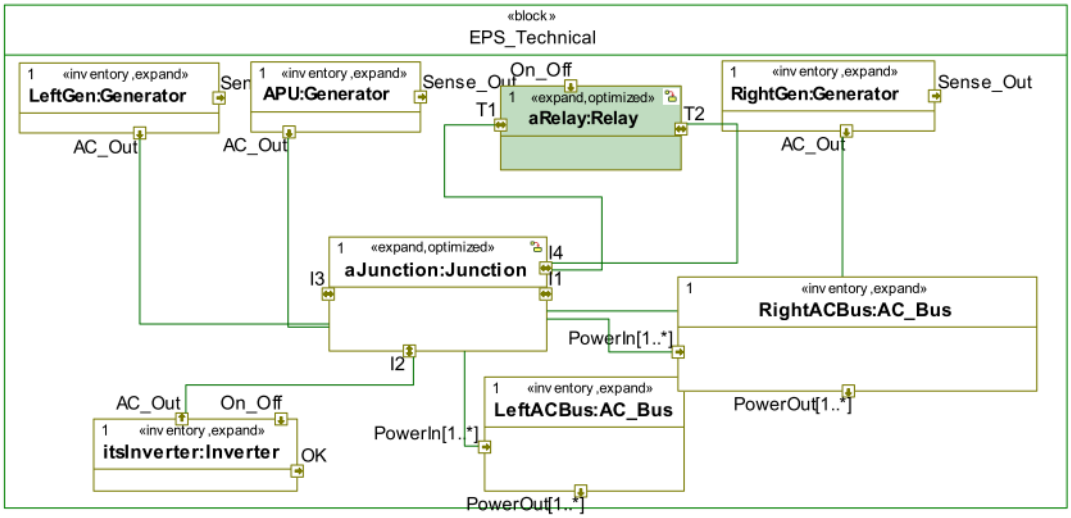


Figure 24 « Modélisation concise pour l'optimisation »

La modélisation des fonctions objectifs est réalisée avec des diagrammes paramétriques, décrivant des équations, pouvant être résolues avec un outil externe, de façon similaire à l'approche « ModelCenter ». Les contraintes sont représentées à l'aide d'un langage spécifique permettant de contraindre l'allocation des fonctions sur les blocs logiques. En effet l'utilisation du langage OCL est jugée beaucoup trop complexe au niveau de la syntaxe dans un contexte de modélisation système.

Cette approche permet donc d'intégrer une étape d'optimisation dans le MBSE, à partir du langage SysML. Par contre, la résolution du problème en tant de tel n'est pas décrite. On ne sait pas comment obtenir et résoudre le problème d'optimisation à partir du modèle enrichi de stéréotypes, en particulier quelle représentation du problème et quel solveur peuvent être utilisés ? La problématique de choix de composant est décrite ; par contre le choix entre redondance de composant et allocation variable n'est pas envisagé.

1.3.5. Optimisation avec EAST-ADL

EAST-ADL est un langage de description d'architecture dédié aux systèmes embarqués automobiles (EAST-ADL, 2016), développé dans plusieurs projets européens de recherche. Ce langage est un profil UML, reprenant également des concepts de SysML et de AADL. Il couvre les aspects opérationnels (les features du véhicule), les fonctions du véhicule, les exigences, les composants HW/SW et les communications entre ECUs (Electronic Control Unit). La notion de variabilité est également intégrée, en raison de la nécessité de décrire toutes les variantes et options d'un véhicule. Dans (Walker, 2013) un processus d'optimisation d'architecture EAST-ADL est proposé pour explorer et évaluer les différentes options d'architectures. Les objectifs d'optimisations sont au nombre de trois : la sûreté de fonctionnement (Dependability), le coût, les performances temps-réel. EAST-ADL permet de modéliser le système véhicule complet. Parmi les niveaux d'analyses utilisés dans la démarche, nous avons le « FAA » (Functional Analysis Architecture) puis les niveaux « FDA » (Functional Design Architecture) et « HDA » (Hardware Design Architecture) représentant la séparation HW/SW. La

variabilité permet de déterminer le degré de liberté dans la conception architecturale et de limiter la taille de l'espace des solutions en entrée de l'optimisation. La variabilité peut être introduite au niveau « FAA » (présence ou non d'une option dans une fonction) ou dans les niveaux « FDA » et « HDA » pour des objectifs de performance ou de coût. Dans ce contexte, trois « variations » possibles sont prises en compte dans le modèle EAST-ADL de départ : Le remplacement d'une fonction par une autre à la conception (même entrées/sortie), la duplication d'une fonction pour créer de la redondance, ainsi que l'allocation. Ces variations sont définies dans le modèle EAST-ADL de départ, fonction par fonction (Abele A., 2012). Une fois ces variations définies, elles doivent être résolues avant de procéder à l'analyse d'une instance d'architecture. Les 3 types de variation par composant : substitution/réplication/allocation engendrent un grand nombre d'architectures possibles qui sont ensuite analysées. Concernant la sûreté de fonctionnement, l'approche qualitative est retenue. Il s'agit de modéliser les liens entre la cause des erreurs et leurs effets (défaillance), en utilisant des fonctions logiques. Cette modélisation est faite dans EAST-ADL (modèle d'erreur), puis une transformation de modèle permet de procéder à une analyse avec l'outil HIP-HOPs qui dispose de son propre modèle d'erreur. Le coût global du système est évalué en prenant un coût par fonction et par élément HW, et en réalisant la somme. Les performances temps réel sont évaluées en utilisant le profil MARTE (diagramme d'activités) pour définir les tâches, ainsi que l'outil d'analyse MAST. La marge temporelle (slack time) cumulée du système (en pourcentage) étant l'objectif à maximiser. La Figure 25 « Optimisation avec EAST-ADL » présente la vue l'architecture de l'outil d'optimisation.

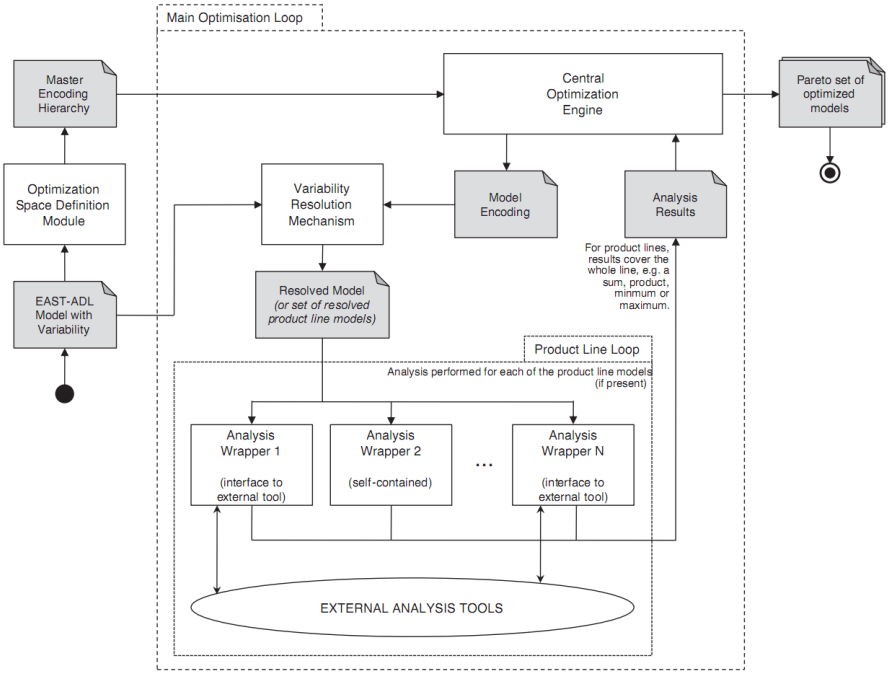


Figure 25 « Optimisation avec EAST-ADL »

1.3.7. Les méthodes d'optimisation d'architectures logicielles

Revue et classification des méthodes (Aleti, 2013)

Dans le contexte des systèmes pour lesquels la partie logicielle est prédominante, une revue systématique des différentes méthodes d'optimisation a été réalisée par (Aleti, 2013). Plus de 180 articles publiés ont été étudiés dans cette revue, à l'exclusion des approches présentées précédemment dans cette thèse, qui ne sont pas spécifiques au systèmes logiciels. Les méthodes retenues incluent un modèle décrivant l'architecture du logiciel, une fonction permettant d'évaluer un ou plusieurs objectifs, ainsi qu'une définition des différents degrés de liberté architecturaux (allocation, sélection de composant ou changement de paramètre).

La Figure 26 « Classification des méthodes d'optimisation d'architecture logicielles »présente la classification retenue pour l'étude.

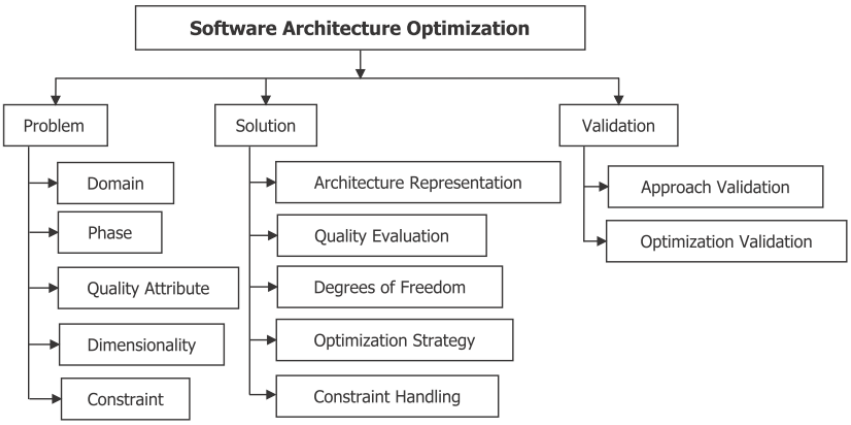


Figure 26 « Classification des méthodes d'optimisation d'architecture logicielles »

La classification propose une organisation des approches existantes en trois niveaux.

Le premier niveau (Problem) concerne le problème lui-même et sa formulation. Le type de système à optimiser (Domain) est soit un système embarqué pour la moitié des approches, soit un système d'information (cloud computing, web application) pour le quart des approches, soit les deux types de système. La phase concerne le positionnement de l'optimisation, soit durant la conception (67%) soit pendant l'exécution du système (30%). Les attributs du problème, ou les objectifs, sont au nombre d'une dizaine, dont la performance, le coût et la fiabilité. Ces trois attributs concernent la majorité des approches. Le nombre d'objectifs ou « Dimensionality » est soit mono-objectif dans 40% des cas, Multi-objectif au sens de Pareto dans 30% des approches et Multi-objectif « pondéré » (ramené à un seul objectif) dans les autres approches. Les contraintes enfin, ne sont pas prise en compte dans 26% des cas, le coût et la performance dans 15% des cas respectivement.

Le second niveau est relatif au processus d'optimisation lui-même. Une première étape de ce processus est la prise en compte d'un modèle d'entrée et de sa représentation (Représentation architecture). Dans 23% des cas ce modèle décrit l'architecture avec un langage

comme UML (3%) ou AADL (4%), ou avec un autre langage très spécifique. Dans 34 % des cas il s'agit d'un modèle propre aux objectifs à optimiser, indépendamment du modèle d'architecture et dans 36% des cas il s'agit d'un modèle mathématique correspondant à un problème d'optimisation spécifique. La seconde étape (Quality Evaluation) correspond à l'évaluation d'un objectif global comme un temps de réponse par rapport à des objectifs locaux. Dans plus d'un tiers des cas les méthodes font intervenir une fonction additive simple, mais on trouve également des fonctions non-linéaires ou des modèles de files d'attentes. La troisième étape concerne les degrés de liberté considérés dans l'architecture. Dans un tiers des cas il s'agit de l'allocation SW/HW, puis la sélection et la duplication de composant HW dans 40% des cas, puis on trouve la sélection et la duplication SW ainsi que l'analyse d'ordonnancement dans 17% des cas. Concernant la stratégie de résolution du problème « Optimization strategy » dans 80% des cas on recherche une solution approchée, en utilisant une heuristique ou une méta-heuristique et dans 20% des cas il s'agit d'une méthode exacte.

Le troisième niveau concerne la validation de l'approche proposée. Dans 80% des cas l'approche est validée par des expérimentations, un cas d'étude ou un exemple relativement basique. Seule 16% des approches ont été validées par un cas d'étude industriel.

L'étude (Aleti, 2013) présente également un certain nombre de recommandations pour les futurs travaux concernant l'optimisation d'architecture logicielle. Tout d'abord **l'interaction entre différents degrés de libertés devrait être étudiée**. Ensuite**, plusieurs algorithmes de résolutions devraient être disponibles pour l'utilisateur, et l'outillage devrait permettre d'intégrer ces algorithmes**. **La modélisation de l'architecture initiale ainsi que de ses variantes est également un problème à résoudre**. **La plupart des approches sont basées sur une représentation spécifique de l'architecture. De ce fait les approches actuelles sont peu réutilisables. Dans ses recommandations pour les futurs travaux,** (Aleti, 2013) **propose pour cela de se baser sur UML ou d'un langage équivalent, qui n'est exploité que dans 3% des approches étudiées.**

Archeopterix

ArcheOpterix est un Framework permettant d'optimiser la conception d'architecture, basé sur AADL dans (Aleti, 2009) . Ce framework est disponible dans un outil basé sur Eclipse. L'outil peut être enrichi avec de nouveaux algorithmes d'évaluation et d'optimisation. Dans les travaux présentés, le langage AADL est utilisé en entrée, et l'optimisation concerne le déploiement des composants logiciels sur les ECUs (mapping). Le résultat de la phase d'optimisation est donc une liste de propriétés de type « Binding » dans le modèle AADL, appliquée à chaque composant logiciel.

Concernant la phase d'analyse, les deux attributs dépendants du déploiement et que l'on souhaite optimiser sont les suivants :

- Data Transmission Reliability (DTR) . Le DTR est la somme du produit de la fréquence d'interaction des composants et de la fiabilité de leur connections, ceci pour toutes les interactions en composants.
- Communication Overhead (CO). Le CO, ou temps d'attente global du système est fonction des temps d'attente réseaux et des interactions entre composants

L'outil permet également d'intégrer des contraintes comme la localisation de composant logiciels, la co-localisation ainsi que des contraintes mémoire. D'autres analyses liées à la SdF pourraient être intégrées par la suite. L'aspect redondance et choix des mécanismes de sûreté n'est pas évoqué, bien qu'étant cité dans une référence. Les analyses proposées par Archeopterix sont basées sur des méthodes analytiques et non par simulation. Une limitation d'archéoptéryx est d'offrir un seul degré de liberté, à savoir l'allocation des composants logiciels, et deux critères assez peu courants : DTR et communication overhead.

Par la suite, la méthode est étendue par (Meedeniya, 2011) pour optimiser la fiabilité de plusieurs fonctions embarquée (ABS, Airbag, ACC), en explorant le déploiement des SW Components sur les ECUs. On cherche à maximiser les fiabilités en explorant l'espace des configurations avec un algorithme génétique. Dans le modèle de fiabilité utilisé, la partie logicielle est modélisée sous forme de composants avec leurs interactions, l'ensemble formant une fonction dont on souhaite estimer la fiabilité. Les composants intègrent des services reliés à d'autres composants (transition) sur laquelle on peut assigner une probabilité. On peut ainsi représenter l'architecture sous forme d'une chaîne de Markov en temps discret (DTMC). D'autres attributs sont renseignés pour les SWC (Composants logiciels) : occupation mémoire, charge CPU de chaque service, probabilité de démarrage d'un service. La plateforme matérielle est également modélisée, comme présenté sur la Figure 27 « modèle d'architecture matériel »

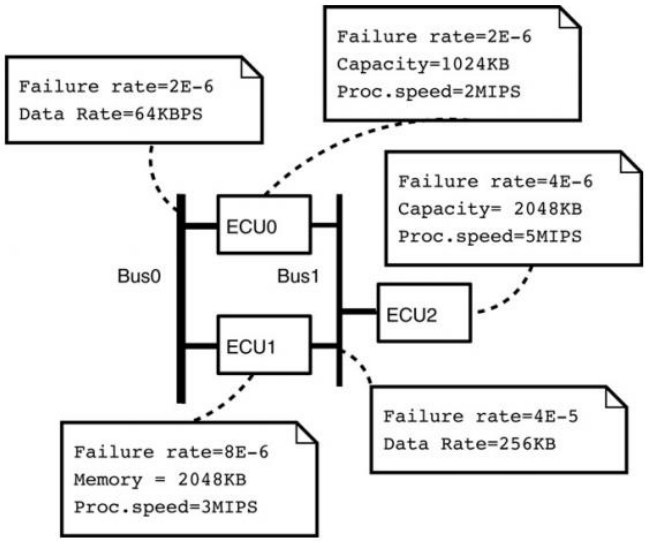


Figure 27 « modèle d'architecture matériel »

Deux types de défaillances sont pris en compte : les défaillances d'un ECU dite d'exécution, d'origine HW elles impactent tous les SWC présents sur cet ECU, et les défaillances de communication, impactant les services utilisant cette ressource. Connaissant les taux de défaut des composants, ECU, liens et réseaux, on peut en déduire la fiabilité de la fonction globale, ceci pour un déploiement donné. Ceci est réalisé par une simulation de Monte-Carlo à partir de la chaîne de Markov. Plusieurs distributions des défauts sont possibles en entrée. La méthode proposée a été testée à partir du framework Archeopterix utilisant le langage AADL.

Dans cette méthode d'optimisation multi-objectifs, seul l'objectifs de fiabilité est pris en compte, mais sur plusieurs fonctions à implémenter. Concernant le modèle de fautes, l'affectation des probabilités aux transitions de la chaîne de Markov peut être délicate. De même la simulation Monte Carlo est coûteuse en temps et en ressource.

1.4.Conclusion et besoin d'enrichir les méthodes actuelles

Le Tableau 2 « Comparatif des méthodes d'optimisation MBSE » présente une synthèse des méthodes étudiées ainsi que celle que nous proposons.

Concernant les langages de modélisation du système et comme le souligne l'étude comparative (Aleti, 2013) , trop peu d'approches utilisent un langage dérivé d'UML, comme SysML ou MARTE. Les approches basées sur des langages DSL (Van Huong, 2012) ou (Aleti, 2013) sont spécifiques aux problèmes d'allocation SW/HW ou aux choix de composants selon les approches. En ingénierie système, le concepteur manipule plusieurs variables de décision. Il souhaite résoudre les alternatives au plus tôt, en utilisant un seul langage et avec l'outillage associé. Pour ce langage d'ingénierie système, nous faisons le choix de SysML. Ce langage est standardisé par l'OMG, notre approche pourra donc être facilement réutilisable. SysML répond aux besoins de l'ingénierie système et il dispose d'un outillage associé non commercial (Papyrus). Cependant notre approche devra rester applicable à d'autres langages de modélisation système comme AADL ou méthode comme ARCADIA.

Concernant les variables de décisions, l'étude des approches permettant de résoudre les problèmes « RAP » de la section 1.3 a montré que le choix de composants ainsi que le niveau de redondance des composants permettent d'optimiser le coût ou la fiabilité d'un système. Plus récemment, les approches MBSE (Aleti, 2013) se sont davantage focalisées sur l'allocation SW/HW. Nous proposons donc de traiter ces trois degrés de liberté (redondance, choix de composants et allocation) ainsi que les variables continues dont l'importance en ingénierie système est soulignée dans (Spyropoulos, 2013). Là encore le langage SysML dispose de concepts assez proches de ces notions. Il permet de modéliser à la fois la structure du système mais également les allocations SW/HW ainsi que les variables continues correspondants aux attributs des blocs.

Nous proposons également d'enrichir SysML avec des stéréotypes permettant de modéliser efficacement les variantes du système. En effet la mise en œuvre de OVM ou CVL n'est pas adaptée pour l'optimisation, car les relations entre ces langages et le modèle de base sont complexes à gérer pour l'utilisateur comme le montre l'étude (Reinhartz-Berger, 2014). Nous

proposons de définir des annotations et des stéréotypes à partir d'un langage comme SysML (le langage du concepteur système) pour exprimer les concepts de variabilité. Ces nouveaux stéréotypes devront cependant être compatibles avec ceux déjà définis dans CVL pour permettre une extension future à ce langage.

Ces stéréotypes seront présentés dans le chapitre 3 et sont adaptés aux choix de composants, à la redondance ainsi qu'à l'allocation. La méthode que nous proposons devra également permettre de modéliser des problèmes combinatoires, continus et mixtes. Pour atteindre cet objectif, plusieurs algorithmes seront proposés pour permettre d'exploiter un solveur à partir d'un modèle SysML. Ces algorithmes seront présentés dans les chapitres 2 et 3 et permettent d'automatiser le processus d'optimisation.

Les méthodes d'optimisation basées sur le langage SysML (ModelCenter/ConsolOptcad/Broodney) sont peu adaptées aux systèmes embarqués, mais plutôt aux systèmes mécaniques. En particulier l'allocation ne figure pas parmi les variables de décision. Selon l'approche, un problème continu (Spyropoulos, 2013) ou discret (Min, 2011) est résolu mais il n'y a pas de possibilité de combiner les deux (problème mixte). Les algorithmes de résolution sont figés et les solveurs utilisés sont intégrés dans des outils commerciaux. Nous proposons de pouvoir intégrer différents solveurs, en utilisant une plate-forme de production de logiciels libre comme Eclipse. Cette possibilité sera explorée au chapitre 4, avec la mise en œuvre de différents solveurs comme Choco, PyOpt ou Labix sur cette plate-forme Eclipse.

Approche	Date publication	Langage/méthode/outil	Variables de décision	Fonctions objectifs	Méthode de résolution	Intégration des résultats dans le modèle
ModelCenter (Min, 2011) (Kerzhner, 2011)	2011	SysML (diag. Paramétrique)/ModelCenter	Choix de composant (discrètes)	Non-linéaire	MINLP (Mono objectif)	Oui
Consol-Optcad (Spyropoulos, 2013)	2013	SysML (diag. Paramétrique)/ConsolOptcad	Variables continues	Linéaire/Quadratique	Algorithme quadratique itératif	
Broodney (Broodney, 2012)	2012	SysML /OOSEM	Choix de composant, variables discrètes		Non précisée	Oui
Van huong (Van Huong, 2012)	2012	DSL (Description architecture HW/SW)	Freq CPU, mémoire,(var. discrètes)	Linéaire/Quadratique	Algorithme génétique	Non
EAST-ADL (Walker, 2013)	2013	EAST ADL (Automobile) + Variabilité	Substitution/Réplication/Allocation	Temps de réponse, SdF	Outil MAST, + Hip-Hop	Oui
Optimisation d'architecture logicielle (synthèse des approches) (Aleti, 2013)	➔ 2013	Modèle d'architecture dans 23% des cas dont UML (3%)	Allocation (33%) puis sélection/duplication SW (40%)	Performance/Coût/Fiabilité en majorité	Solution approchée par méta-heuristique dans 80% des cas	
Approche proposée	**2016**	**SysML/OOSEM et outillage open source**	**Choix de composants, redondance, paramètre continu et allocation**	**Intégrée au modèle SysML, définissable par le concepteur**	**CSP pour les problèmes discrets, framework d'optimisation pour les problèmes continus/Mixte**	**Possible**

Tableau 2 « Comparatif des méthodes d'optimisation MBSE »

2. Optimisation MBSE et transformations associées

Dans ce chapitre nous proposons plusieurs concepts nécessaires à l'intégration de l'optimisation dans l'ingénierie système basée sur les modèles. Dans une première partie, nous proposons d'enrichir un langage d'ingénierie système avec des « points de décision ». Ces points de décision s'inspirent de la notion de variabilité que nous avons vue précédemment, mais avec un objectif différent, à savoir l'optimisation du système. Dans une seconde partie nous proposons d'associer à ces points de décision plusieurs types de variables propres au problème d'optimisation CSMOP. Ces variables vont nous permettre de résoudre efficacement le problème CSMOP. Puis nous formulerons un ensemble de contraintes à intégrer dans le modèle initial ainsi que les fonctions objectifs, et nous réunirons l'ensemble de ces nouvelles notions dans un contexte d'optimisation. Finalement, nous décrirons les transformations permettant de générer automatiquement le problème CSMOP à partir du modèle du système enrichi avec les points de décisions.

2.1. Introduction

Les concepts présentés dans ce chapitre sont définis indépendamment d'un langage de modélisation particulier tel que SysML (Friedenthal, 2014) ou AADL (Feiler, 2006). Nous prenons pour hypothèse que ce langage « MBSE » se compose de classes et de relations de composition entre ces classes. Chaque classe dispose d'un ensemble d'attributs typés. Le système à optimiser *(SuA)* est représenté par un ensemble de classes ainsi que par un ensemble de relations de composition « *CompositeAssociation* ». Une relation de composition dispose d'une multiplicité et permet d'associer une classe « tout » à une ou plusieurs de ses parties. Le langage utilisé intégrera également la notion d'allocation, permettant d'associer des éléments de différents niveaux d'abstraction ou de différents types, ainsi que la notion de contrainte. Le Tableau 3 « correspondance Elément générique/SysML/AADL » présente la correspondance entre ces éléments.

Elément générique	Métamodèle SysML	Métamodèle AADL
Classe	SysML ::Block	Device, memory, processor, bus
Système à analyser (SuA)	SysML ::Block, BDD	Composite, system
Relation de Composition	Composition associations	Subcomponents
Attributs d'une Classe	Property	Properties
Allocation	Allocate for Behavior allocation	Allowed Processor Binding property
Eléments sources pour l'allocation	Single Action/Activity	Single thread or list of threads
Eléments cibles pour l'allocation	Single Block/Part	Single processor or list of processors
instanciation d'un classe	Instance Diagram	Instance model
Contrainte	Constraint Block	Annotations

Tableau 3 « correspondance Elément générique/SysML/AADL »

Lorsque l'on modélise un système et en particulier une plateforme d'exécution formée de composants HW, il est nécessaire de réaliser un modèle au niveau « classe » pour la structure du système ainsi qu'un modèle au niveau « instance ». Une classe étant une généralisation de toutes les instances disposant du même ensemble d'attributs. Les informations contenues dans les classes ne sont pas suffisantes pour pouvoir calculer et donc optimiser les futures caractéristiques du système, comme la performance, la fiabilité ou le coût. Généralement Il manque la valeur des attributs. Si l'on prend l'exemple d'un système dont on veut déterminer le coût à partir de ceux de ses composants, choisis parmi une librairie existante contenant les instances, il faudra tenir compte du coût de ces instances pour estimer le coût du système.

2.2.Les points de décisions pour l'ingénierie système

2.2.1. Le problème du choix de composants et de la redondance

Dans le modèle de niveau classe, nous introduisons des degrés de liberté que nous nommons des points de décision sur les classes mais également sur les instances de ces classes. Ces points de décision « *DecisionPoint* » sont représentés dans la Figure 28 «Points de décision de structure et d'instance». Nous définissons deux types de points de décision : les points de décisions sur les instances « *InstanceDecision* » et les points de décisions sur les multiplicités de classes « StructureDecision ». L'ensemble des points de décisions du modèle est intégré dans une spécification « DecisionSpecification ».

Le point de décision « InstanceDecision » s'applique à une classe du modèle initial par la relation « getClass ». Ce point de décision correspond à la notion de choix de composants HW sur étagère, choisis dans une liste dont les propriétés sont connues. Le choix de composants sur étagère est utilisé en ingénierie système (Min, 2011) (Kerzhner, 2011) (Broodney, 2012), lorsque l'on souhaite réutiliser un composant provenant d'un autre modèle. Cette liste d'instances est définie par une composition entre « InstanceDecision » et « Instance ». Elle est intégrée dans le modèle initial sous forme de table. Chaque instance de la liste dispose de valeurs fixées, correspondant aux propriétés des composants disponibles.

Le point de décision « StructureDecision » s'applique à une relation de composition du modèle initial par la relation « getCompositon », pour représenter le niveau de redondance variable d'une partie d'une classe. Cette relation de composition implique une classe dit contenant (le tout) et un ensemble de « n » classes contenues (les parties). Le point de décision « StructureDecision » modélise la redondance des classes contenues, de la valeur « 1 » (aucune redondance) à « n » pour une redondance de niveau n.

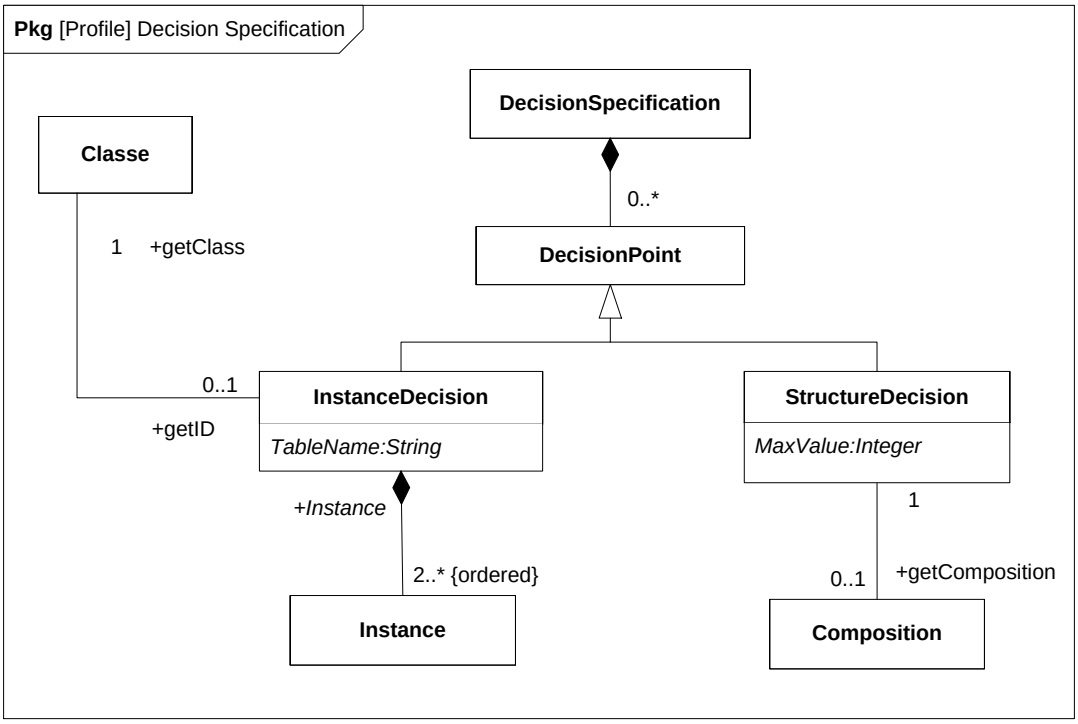


Figure 28 «Points de décision de structure et d'instance»

Dans le premier cas d'étude du chapitre 5, La Figure 61 «Système enrichi avec les points de décisions» de la page 109 présente un système à analyser SuA intégrant les points de décisions « InstanceDecision » et « StructureDecision » que nous venons de décrire, dans le langage SysML.

Formalisation

La formalisation proposée dans ce paragraphe va nous permettre de réaliser des tests de cohérence dans les algorithmes de la sous-section 2.6.

Le système à analyser est représenté par un graphe G formé par les nœuds classes et les relations de composition entre ces classes. Nous pouvons définir le système à analyser SuA enrichi avec les points de décisions comme un tuple composé d'un graphe G, d'un ensemble DP *de* points de décision *(DecisionPoint)* et d'un ensemble DL de liens (*Decision Link*) entre les nœuds classes et les points de décisions :

$$SuA = (G, DP, DL)$$

Définition 4: Système enrichi avec les points de décisions

Le graphe G est un couple $G = (CN, RC)$ où CN est l'ensemble des nœuds classes qui composent le système et RC l'ensemble des relations de composition entre les nœuds classes. Les relations de composition étant d'une part orientées entre un « tout » et des parties et d'autre part unique entre deux nœuds Classes, le graphe G est un graphe simple orienté.

Les points de décisions DP sont catégorisés en deux types : les points de décisions d'instance et les points de décisions de structure :

$$DP = DPi \cup DPs \; where \; DPi \cap DPs = \emptyset$$

$Vtype$ est une fonction de typage : $Vtype : DP \rightarrow Type \; avec \; Type = \{Instance, Structure\}$

Un lien $dl_s \in DL$ est un ensemble composé d'une relation de composition rc et d'un point de décision dp_s :

$$dl_s = \{rc, dp_s\} \; where \; rc \in RC \; and \; dp_s \in DP \; and \; Vtype(dp_s) = Structure$$

Un lien $dl_i \in DL$ est un ensemble composé d'un nœud Classe nc et d'un point de décision dp_i :

$$dl_i = \{nc, dp_i\} \; where \; nc \in Nc \; and \; dp_i \in DP \; and \; Vtype(dp_i) = Instance$$

2.2.2. La variation des attributs du modèle

Les attributs des différentes classes du modèle système ont une influence importante quant aux objectifs que l'on souhaite maximiser ou minimiser. Par exemple si l'on considère une caméra dotée de fonctions de détection d'obstacles présentée dans la Figure 29 «Exemple de point de variation sur les attributs» dans le langage SysML, lorsque nous devons modéliser l'intégration de cette caméra dans un véhicule, nous modélisons la position et l'angle d'inclinaison de cette caméra par des attributs disposant d'un type ainsi que de domaines de variation. Nous souhaitons pouvoir obtenir les valeurs optimales de ces attributs permettant de maximiser les performances tout en minimisant le coût de cette caméra lorsque nous intégrons le choix des composants HW. Les types de données associés à ces attributs doivent nous permettre de traiter les problèmes multi-domaines, incluant une partie matérielle (HW) déterminante pour le coût, ainsi qu'une partie SW et optique pour les performances. Les lois de l'optique doivent être prises en compte pour modéliser les performances du système. C'est pourquoi nous devons permettre au concepteur d'associer des points de décision à des attributs discrets (représentés par des nombres entiers) mais aussi continus (représentés par des réels). Il s'agit alors d'un problème mixte, incluant les deux types de variables.

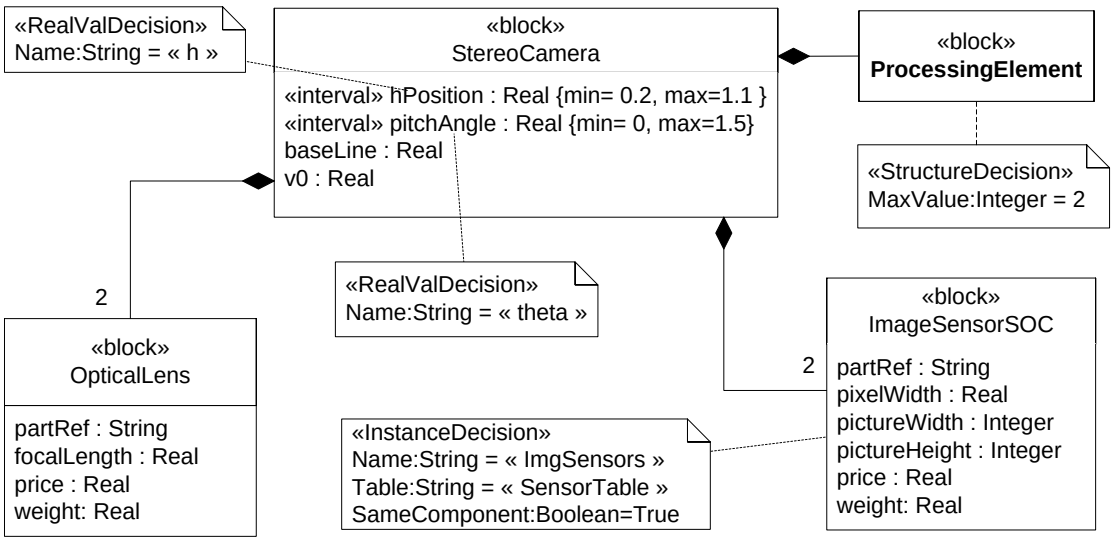


Figure 29 «Exemple de point de variation sur les attributs»

La Figure 30 «Point de décision des attributs» présente les notions que nous proposons pour permettre d'ajouter des variations aux attributs. Il s'agit de points de décision applicables aux attributs «*AttributeValue*» d'une classe du modèle initial.

Exemple

La Figure 29 «Exemple de point de variation sur les attributs» présente un cas d'utilisation du point de décision « ***RealValDecision***» dans le langage SysML. Le bloc « *StereoCamera*» modélise une caméra stéréoscopique embarquée dans un véhicule terrestre. L'attribut « *hPosition* » représente la hauteur de positionnement de la caméra par rapport à la base du véhicule et «*pitchAngle*» l'angle d'inclinaison de la caméra par rapport à la route. Ces deux paramètres influent sur les performances de la caméra, pour obtenir un compromis entre la détection d'obstacle proche et la détection d'objets éloignés. Nous affectons donc deux points de décision « ***RealValDecision***» à ces attributs de type réel et dont les valeurs sont bornées.

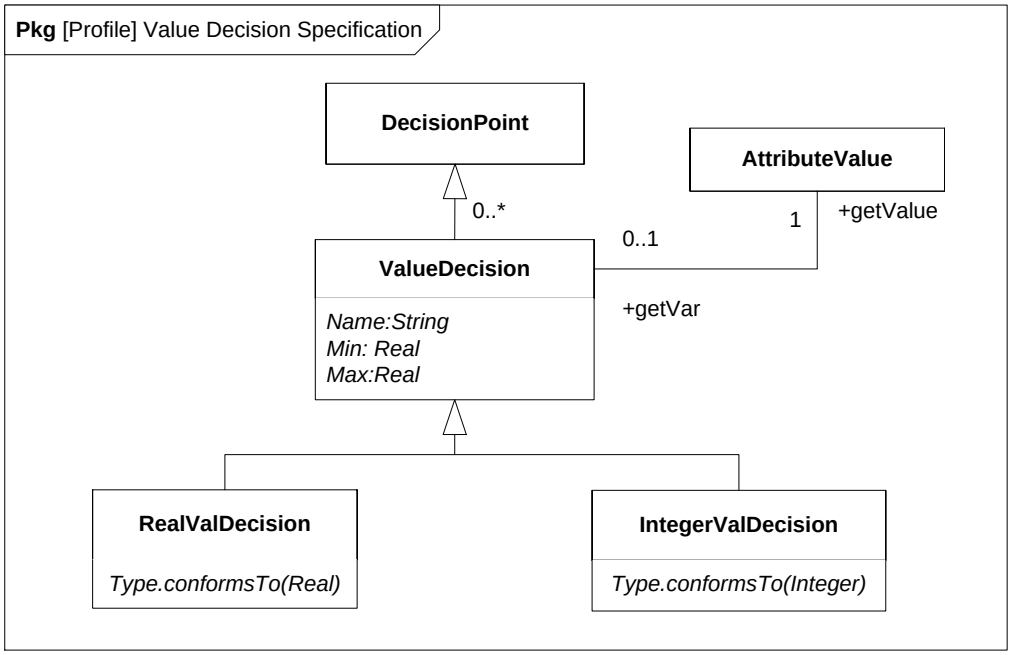


Figure 30 «Point de décision des attributs»

2.2.3. L'allocation variable

L'allocation en ingénierie système est une notion générale permettant d'interconnecter des éléments de différents niveaux d'abstraction ou de différents types. On peut allouer par exemple une activité (de type logiciel) à une ressource (de type matériel) chargée de son exécution. Cette relation permet d'allouer un ensemble d'éléments source à un ensemble d'éléments cibles (Target). Dans un modèle, cette relation n-aire peut se représenter graphiquement par une flèche en pointillée lorsque les éléments source et destination sont représentés dans le même diagramme. Une autre possibilité est de représenter cette allocation par un attribut spécifique d'un élément du modèle ou bien par un commentaire attaché à un élément, précisant la source ou les éléments cibles de l'allocation dans laquelle il est impliqué.

Dans un objectif d'optimisation du système, l'allocation variable représente non pas une situation réelle, mais plutôt un ensemble de possibilités parmi lesquelles le concepteur devra choisir. Ces possibilités sont souvent nombreuses, c'est pourquoi une représentation graphique à l'aide de flèches n'est pas souhaitable. Une autre spécificité de ce contexte d'utilisation est l'existence de contraintes globales entre les différentes allocations. Une contrainte est dite globale lorsqu'elle s'applique à plusieurs allocations variables, par opposition à une contrainte ne s'appliquant qu'à une seule allocation. Par exemple deux activités ne pourront pas (ou devront pas) être allouées à la même ressource.

Nous proposons donc un ensemble de notions permettant de définir l'allocation variable. Dans la Figure 31 « allocation variable et contraintes associées »

le système à analyser « *SystemUnderAnalysis* » est enrichi avec une spécification de déploiement « *DeploymentSpecification* » ainsi qu'avec des contraintes globales « *GlobalConstraint* ». La spécification de déploiement, représentée par une table, est une liste contenant l'ensemble des allocations possible (ou variable) du système. L'allocation variable peut être représentée soit par un commentaire « *VariableAllocComment* » soit par une allocation « *VariableAllocate* » pour un modèle disposant d'un grand nombre d'allocations variables (plusieurs dizaines). Dans les deux cas on retrouve un élément unique « *Source* » comme origine de la relation et plusieurs éléments « *Target* ». Un tableau de paramètres « *AllocationParameter* » permet de spécifier un coût ou une valeur pour chaque instance de la relation d'allocation. Des contraintes globales s'appliquent également aux différentes allocations, avec les contraintes « SameTarget » et « DifferentTarget » pour indiquer que les sources devront (ou pas) être allouées à la même cible.

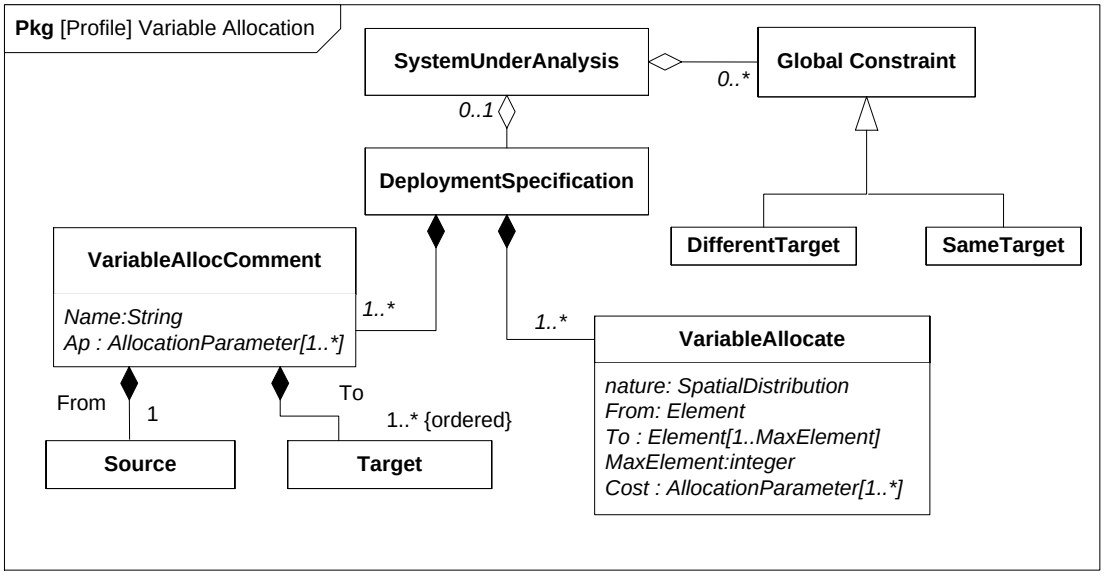


Figure 31 « allocation variable et contraintes associées »

Afin de réaliser des tests de cohérence dans les algorithmes de la section 2.6, nous pouvons définir le système à analyser SuA incluant l'allocation variable comme un tuple :

$$SuA = (T, S, A, K)$$

Définition 5: Allocation variable

Avec :

- $T = \{t_1, t_2, .. t_m\}$, est l'ensemble des éléments cibles (*Target*) pour l'allocation
- $S = \{s_1, s_2, .. s_n\}$, est l'ensemble des éléments sources pour l'allocation
- $A \subseteq (S \times T)$ est une relation d'allocation dans laquelle pour chaque $s \in S$ il existe au moins un élément $t \in T$ tel que $(s, t) \in A$. Cette relation décrit l'ensemble des combinaisons d'allocations possibles.

- $K = \{K_1, K_2, .. K_p\}$ un ensemble de contraintes globales (*Global Constraint*) où K_i est une fonction booléenne impliquant une séquence de variables $X(K_i) = \{a_{i1}, a_{i2}, .. a_{il}\}$ avec $a \in A$

2.3.Les variables associées aux points de décision

Comme nous l'avons vu au chapitre 1.2, le choix des variables d'un problème d'optimisation et leur domaines respectifs sont déterminants dans la formulation d'un problème CSMOP. D'une part ce choix va influer sur le temps de résolution du problème par l'algorithme du solveur, et d'autre part la formulation des fonctions objectifs dans le modèle peut être simplifiée. Nous proposons donc plusieurs choix de domaines pour les variables, adaptés aux problèmes d'ingénierie système que nous souhaitons résoudre.

2.3.1. Variables de décision pour la redondance et le choix d'instances

Dans ce type de problèmes on considère un bloc B disposant d'un niveau de redondance $r \in \{1 \dots n\}$ correspondant à un point de décision de structure DP_s. Ce même bloc peut également être instancié par un ensemble $IB = \{ib_1, ib_2 \dots ib_m\}$ d'instances disponibles « sur étagère » correspondant à un point de décision d'instance DP_i.

Nous distinguons trois cas pour la variable de décision x prenant sa valeur dans un domaine D associées à ce bloc.

- Le bloc B dispose d'un point de décision d'instance DP_i seul. Pour ce seul degré de liberté, nous créons une variable d'optimisation x de type entier énuméré. Chaque valeur de l'énuméré identifie un composant dans la liste des instances utilisables.
- Le bloc B dispose d'un point de décision de structure DP_s seul. Pour ce degré de liberté nous créons une variable d'optimisation x de type entier avec $x \in [1..\mathrm{n}]$ indiquant le niveau de redondance du bloc B.
- Le bloc B dispose de deux points de décision combinés DP_s et DP_i. Nous associons à B une matrice de booléens $x_{ij} \in \{0,1\}$ avec :

$$i \in \{1..\mathrm{n}\}, j \in \{1..\mathrm{m}\}\ x_{ij} = \begin{cases} 1\ si\ l'instance\ ib_j\ du\ bloc\ B\ est\ utilisée\ en\ position\ i \\ 0\ sinon \end{cases}$$

Équation 1: point de décision combinés

La Figure 32«Point de décision combiné instance/structure» illustre cette notion de position d'instance, en combinant un choix de redondance (pas de redondance ou une redondance simple avec deux instances soit $n = 2$) et un choix d'instances de blocs parmi m.

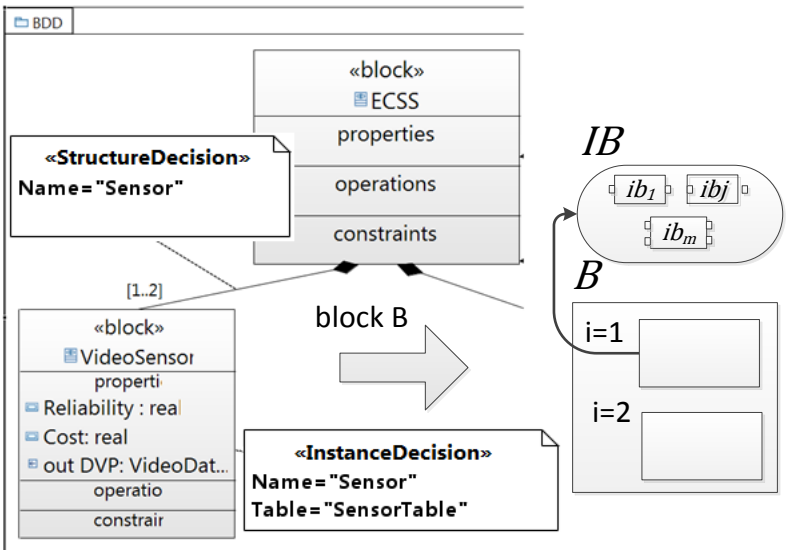


Figure 32«Point de décision combiné instance/structure»

2.3.2. Variables de décision pour les attributs

La création des variables de décision pour les attributs est assez directe, deux cas étant à considérer :

- L'attribut dispose d'un point de décision « RealValDecision ». Pour ce degré de liberté, nous créons une variable d'optimisation x de type réel bornée par les valeurs « min » et « max » de l'attribut.
- L'attribut dispose d'un point de décision « integerValDecision ». Pour ce degré de liberté, nous créons une variable d'optimisation x de type entier bornée par les valeurs « min » et « max » de l'attribut.

2.3.3. Variables de décision pour l'allocation

Nous pouvons représenter les variables de décision d'un problème d'allocation des éléments sources $S = \{s_1, s_2, .. s_n\}$ sur les éléments cibles $T = \{t_1, t_2, .. t_m\}$ de la Définition 5: Allocation variable, avec les éléments suivants :

- $X = \{x_1, x_2, .. x_n\}$, l'ensemble des variables de décision, représentant la spécification d'allocation
- $D = \{D_1, D_2, .. D_n\}$, l'ensemble des domaines avec $x_i \in D_i$,

Deux formulations sont alors possibles et utilisables conjointement ou alternativement. La première est une formulation en nombre entier, avec $D_i \subset \mathbb{N}$ et :

$$\forall s_i \in S, \forall t_j \in T \quad x_i = j \Leftrightarrow s_i \; est \; allou\acute{e} \; \grave{a} \; t_j$$

Équation 2: formulation en nombre entier pour l'allocation

La seconde possibilité est une formulation booléenne, avec :

$$\forall s_i \in S, t_j \in T \quad x_{ij} = \begin{cases} 1 \text{ si } s_i \; est \; allouée \text{ à } t_j \\ 0 \; sinon \end{cases}$$

Équation 3: Formulation booléenne pour l'allocation

2.4.La modélisation des contraintes

Les contraintes applicables au système modélisé permettent d'en limiter le nombre de variantes répondant aux critères d'optimisation. Dans le problème d'optimisation que nous souhaitons résoudre, ces contraintes s'appliquent aux variables de décisions. Dans le premier chapitre, l'étude des problèmes « RAP » ainsi que des problèmes d'allocation (Aleti, 2013) nous ont renseignés sur le type de variables que nous devions traiter : le choix de composants, le niveau de redondance ainsi que l'allocation SW/HW. Des contraintes associées à chaque type de variables sont présentées dans la Figure 33 « Contraintes pour l'optimisation système ». Il s'agit des contraintes de structures, applicables aux blocs et à leurs instances, des contraintes de connections (binding) applicables aux connections entre les blocs ainsi qu'aux contraintes d'allocation.

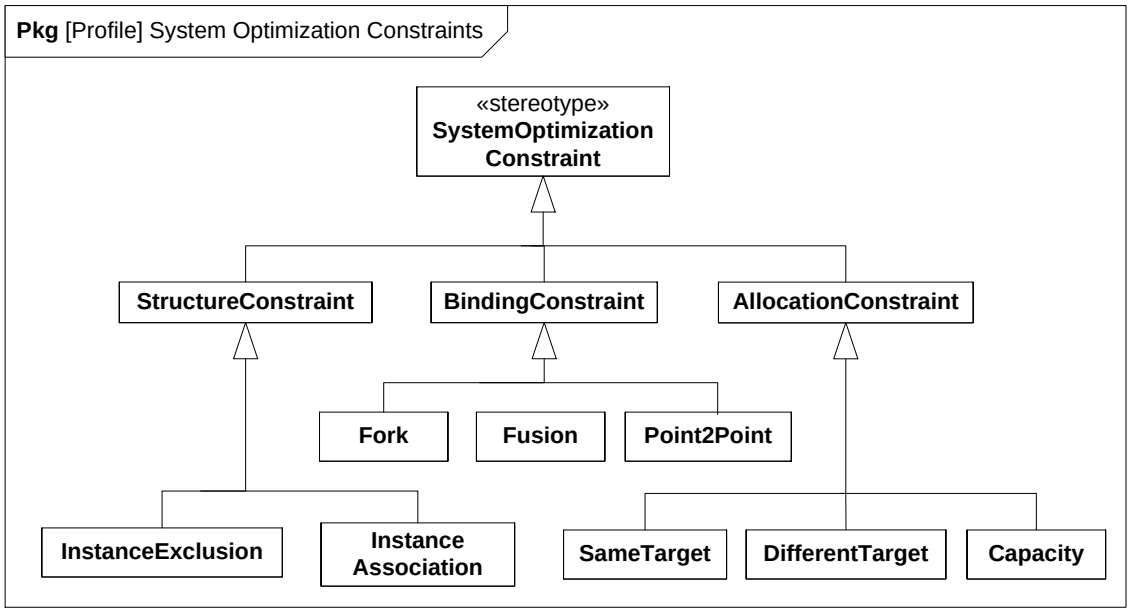


Figure 33 « Contraintes pour l'optimisation système »

Les contraintes de structure

Ces contraintes sont applicables aux blocs et à leur instance, et dépendent des points de décisions appliqués à ces blocs (instance ou structure). Lorsqu'un bloc dispose d'un point de décision de structure et d'un point de décision d'instance, nous utilisons les variables de décisions définis par l'Équation 1: point de décision combinés de la page 63. Dans ce cas la contrainte suivante exprime que, pour le système final et en considérant un bloc B disposant d'une redondance maximum n et d'un choix d'instance parmi m, on ne peut avoir une seule instance de bloc en position i dans le système réel :

$$\forall i \in \{1..\mathrm{n}\} \qquad \sum_{j=1}^{j=m} x_{ij} \leq 1$$

Équation 4 « Contrainte d'instanciation »

Les contraintes de compatibilité (association) ou d'incompatibilité (exclusion) entre instances de blocs peuvent également être exprimées. Par exemple si l'on souhaite associer un bloc B_1 « CPU » à un bloc B_2 « interface réseau », on peut vouloir spécifier qu'une instance j du bloc « CPU » est incompatible avec l'instance k du bloc « interface réseau » car j ne dispose pas du port correspondant. Les blocs B_1 et B_2 sont soumis à des points de décisions d'instance et de structure, exprimés par l'Équation 1: point de décision combinés. Les variables de décision a_{ij} et b_{ij} correspondent respectivement aux blocs B_1 et B_2. On peut exprimer cette contrainte d'exclusion par :

$$\sum_{i=1}^{n} a_{ij} + \sum_{i=1}^{n} b_{ik} \leq 1$$

Équation 5 Contrainte d'exclusion

De même lorsque l'instance j du bloc B_1 doit être associée à l'instance k du bloc B_2:

$$\sum_{i=1}^{n} a_{ij} \leq \sum_{i=1}^{n} b_{ik}$$

Équation 6 Contrainte d'association

Les contraintes de connexion (Binding) entre les blocs

Dans un système complexe, les blocs sont généralement reliés par des connecteurs. L'extrémité d'un connecteur possède une multiplicité décrivant le nombre d'instances connectables par des liens décrits par le connecteur. Le connecteur fournit l'opportunité aux blocs d'interagir ensemble à travers des ports de communication. Dans les langages de modélisation système, on trouve généralement deux types de ports :

- Les ports de données (*FlowPort*), autorisant la circulation de flux physique (fluide, énergie) ou de flux d'information
- Les ports standards (*ControlPort*), autorisant l'échange de services logiques entre les blocs, au moyen d'interfaces regroupant des opérations.

On peut associer également une multiplicité à un port. On peut ainsi exprimer succinctement qu'un calculateur dispose de deux ports (interfaces) CAN identiques.

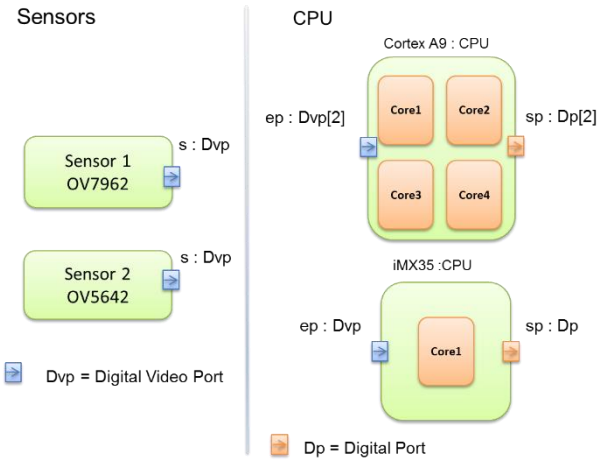
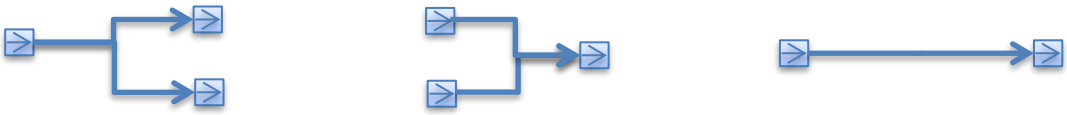


Figure 34: « instances de blocs avec leur ports »

Dans la Figure 34: « instances de blocs avec leur ports », des composants « CPU » et « Sensors » sont représentés par des instances de blocs munis de ports. Chaque port est de type « entrée » ou « sortie » suivant l'orientation de la flèche. Les contraintes de connections expriment le fait qu'une instance de bloc peut être connectée ou non à une autre instance de bloc. On ne cherche pas à établir une matrice de communication entre les instances de blocs, mais plutôt à éliminer les configurations impossibles. Par exemple lorsqu'un capteur est soumis à une redondance optionnelle (point de décision de structure) et qu'il doit être relié à un ou deux CPU, une solution avec deux capteurs et un CPU disposant d'un port unique d'entrée ne sera pas retenue.

On distingue trois types de connections entre les ports d'entrée et de sortie : l'embranchement, la fusion ou le « point à point ».



embranchement fusion point à point

Dans l'exemple de la Figure 34: « instances de blocs avec leur ports », les règles suivantes s'appliquent aux instances de blocs disposant de ports de type DVP (port numérique vidéo) et DP (port numérique).

	embranchement	fusion	Point à point
Digital Video Port	Possible uniquement si les deux ports de sorties appartiennent à deux composants différents	non	oui
Digital Port	non	non	oui

Lorsque l'on considère deux blocs B_1 (source) et B_2 (destination) dont les instances doivent être connectées, on peut définir trois types de contraintes de connexion correspondant à l'embranchement, la fusion ou le point à point. Un bloc $\mathrm{B_i}$ est modélisé par une classe disposant d'un nombre variable de ports, représentée par un couple $\{e_i, s_i\}$ avec e_i le nombre de ports d'entrées du bloc $\mathrm{B_i}$ et s_i le nombre de ports de sorties. Les blocs B_1 et B_2 sont soumis à des points de décisions d'instance et de structure, exprimés par l'Équation 1: point de décision combinés. Les variables de décision a_{ij} et b_{ij} correspondent respectivement aux blocs B_1 et B_2. Nous pouvons alors exprimer une contrainte d'embranchement entre les blocs bloc B_1 et B_2 par l'inégalité suivante :

$$\sum_{i,j} b_{ij} e_2 \geq \sum_{i,j} a_{ij}\, s_1$$

Équation 7 Contrainte d'embranchement ou «fork»

La contrainte de fusion entre les blocs B_1 et B_2 s'exprime avec l'inégalité suivante :

$$\sum_{i,j} b_{ij} e_2 \leq \sum_{i,j} a_{ij}\, s_1$$

Équation 8 Contrainte de fusion

La contrainte « point à point » s'exprime entre les blocs B_1 et B_2 avec l'égalité suivante :

$$\sum_{i,j} b_{ij} e_2 = \sum_{i,j} a_{ij}\, s_1$$

Équation 9 Contrainte point à point

Les contraintes d'allocation

Dans le cas de l'allocation variable, une première contrainte permet d'exprimer qu'un élément source est alloué à un seul élément cible. Lorsque la formulation en nombre entier est utilisée, cette contrainte est intégrée dans la formulation de l'équation (1). Lorsqu'il s'agit de la formulation booléenne, en considérant l'allocation des éléments sources $S = \{s_1, s_2, .. s_n\}$ sur les éléments cibles $T = \{t_1, t_2, .. t_m\}$ de la Définition 5: Allocation variable, nous rajoutons les contraintes suivantes :

$$\forall i \in \{1..n\} \quad \sum_{j=1}^{j=m} x_{ij} \leq 1$$

Équation 10 «contrainte d'unicité pour l'allocation»

Une seconde contrainte permet de spécifier l'ensemble des éléments cibles α « allouables » à un élément source s_i. Cette contrainte est exprimée par un couple (s_i, α) avec $\alpha \subseteq T$

Avec la formulation en nombre entier, cette contrainte est incluse dans le domaine de définition $D_i = \alpha$ de chaque variable x_i. Avec la formulation booléenne, nous ajoutons la contrainte suivante pour une ressource s_i:

$$\forall i \in \{1..n\} \;\; x_{ij} = 0 \; avec \;\; c_j \in \; \bar{\alpha}$$

Équation 11 « éléments allouables »

La contrainte « SameTarget » permet d'exprimer qu'un ensemble d'éléments sources $\beta \subseteq S \; avec \; S = \{s_1, s_2, .. s_n\}$ est alloué à une même destination $t_j \in T \; avec \; T = \{t_1, t_2, .. t_m\}$. Avec la formulation en nombre entier, nous rajoutons une contrainte globale au problème :

$$AllEqual(x_i) \; with \; x_i \in \; \beta$$

Avec la formulation booléenne nous rajoutons un ensemble de m contraintes :

$$\forall \, j \in \{1..m\} \;\; AllEqual(x_{ij}) \;\; with \; x_i \in \; \beta$$

De même la contrainte « DifferentTarget » permet d'exprimer qu'un ensemble d'éléments sources $\beta \subseteq S$ ne doit pas être alloué à une même destination $t \in T$. L'expression des contraintes devient, pour la formulation en nombre entier :

$$AllDifferent(x_i) \; with \; x_i \in \; \beta$$

Ainsi que pour la formulation booléenne :

$$\forall \, j \in \{1..m\} \;\; AllDifferent(x_{ij}) \;\; with \; x_i \in \; \beta$$

D'autres contraintes sont utilisées dans le cas de l'allocation. La contrainte de ressource ou « **Capacity** » exprime le fait qu'un élément source $s_i \in S \; avec \; S = \{s_1, s_2, .. s_n\}$ consomme une ressource m_i et que la somme des ressources allouée à une destination $t_j$est limité par m_j :

$$\forall \, j \in \{1..m\} \quad \sum_i x_{ij} m_i \leq \; m_j$$

Équation 12 « contrainte de ressource pour l'allocation »

Dans le cas d'un système où s_i représente une tâche de période T_i et de WCET C_{ij}et $t_j \in T \; avec \; T = \{t_1, t_2, .. t_m\}$ un processeur, la contrainte suivante exprime le fait que le taux d'utilisation d'un processeur ne peut être supérieur à 1 :

$$\forall \, j \in \{1..m\} \quad \sum_i x_{ij} \frac{C_{ij}}{T_i} \leq \; 1$$

Équation 13 « contrainte d'utilisation pour l'allocation »

2.5.Le contexte d'optimisation

Le contexte d'optimisation permet de rassembler les différents éléments nécessaires à l'optimisation du système. Il fait partie du modèle du système à optimiser. Le contexte d'optimisation est représenté par un contexte « $MdOC$ » (Multi-domain Optimisation Context) composé des éléments présentés dans la Figure 35: « Contexte d'optimisation » . On y retrouve les éléments suivants :

- Une référence « SystUnderAnalysis » vers le système à analyser, contenant les différents points de décision (Instance, composant, allocation variable) définis précédemment.
- Un bloc « Opt Model » représentant le modèle d'optimisation. Les paramètres de ce modèle se composent d'une représentation des variables d'optimisation, en utilisant des types « booleen », « entier » ou « réel ». Chacune des contraintes de ce bloc correspond à une expression mathématique, intégrée sous forme de code source dans le langage du solveur sélectionné.
- Une ou plusieurs fonctions objectifs « ObjectiveFunction ». Une fonction objectif permet de calculer la valeur d'un objectif à minimiser ou maximiser. Cette valeur est fonction du modèle d'optimisation mais aussi de valeurs contenues dans le système à analyser et dans l'environnement.
- Une fonction dite de Pareto « ParetoFront ». Cette fonction permet de générer les solutions optimales du problème, selon les différents objectifs.
- L'environnement représente le contexte d'utilisation dans lequel on souhaite optimiser les objectifs. Il peut intégrer des éléments statiques comme des données d'entrées du système ou également des éléments dynamiques comme un scénario décrit par des diagrammes de séquences.

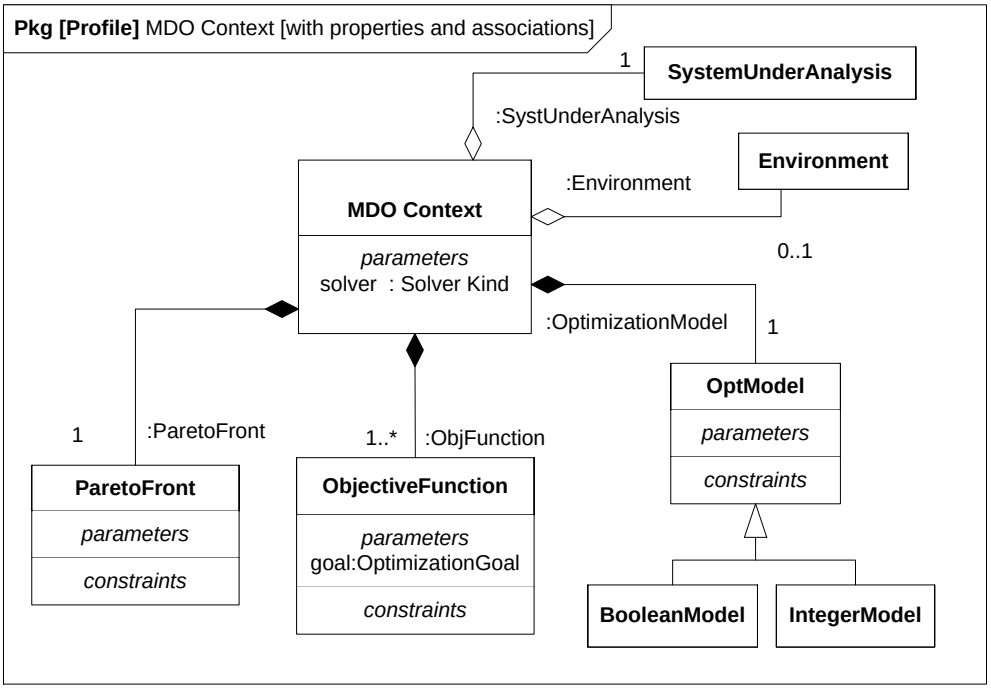


Figure 35: « Contexte d'optimisation »

Ce contexte « $MdOC$ » sera utilisé par la suite dans les algorithmes de transformation. La définition suivante du tuple « $MdOC$ » permettra de réaliser des tests de cohérence :

$$MdOC = (OCB, OFN, Om, Pfn, Su, Ra)$$

Avec :

- *OCB (Optimisation Context Bloc)* un nœud bloc représentant le contexte d'analyse
- *OFN (Objective Function Nodes)* un ensemble de fonctions « ObjectiveFunction »
- *OM (Optimization Model)* un nœud bloc nommé « OptModel »
- *PF (Pareto Front)* un nœud bloc nommé « ParetoFront »
- *SUA (System Under Analysis)* un nœud bloc représentant le système à analyser
- *AR (Association Relathionship)* un ensemble de relations d'association entre les nœuds bloc de $MdOC$

Les relations d'association entre nœuds sont catégorisées en composition ($Rcomp$), référence ($RShared$), et généralisation ($RGen$):

$$AR = \{Rcomp \cup RShared \cup RGen\}$$

$Rtype$ est une fonction de typage :

$$Rtype : AR \rightarrow Type\ avec\ Type = \{Composite, Shared, Generalization\}$$

L'ensemble des relations AR est constitué :

- D'une relation $Pf \in AR$. C'est un couple constitué du contexte OCB et du nœud bloc PF:

$$Pf = (OCB, PF)\ where\ Rtype(Pf) = Composite$$

- De relations $Of \in AR$. Il s'agit de couples constitués du contexte OCB et d'une fonction objectif Ofn :

$$Of = (\mathrm{OCB}, Ofn)\ where\ Ofn \in OFN\ and\ type(Of) = Composite$$

- D'une relation $Omd \in AR$. C'est un couple constitué du contexte OCB et du nœud bloc Om :

$$Omd = (\mathrm{OCB}, Om)\ where\ type(Omd) = Composite$$

- D'une relation $Sua \in AR$ C'est un couple constitué du contexte OCB et du nœud bloc Su :

$$Sua = (\mathrm{OCB}, Su)\ where\ type(Sua) = Shared$$

2.6.Algorithmes de transformations

Pour faciliter l'exploitation d'un modèle enrichi pour l'optimisation, nous proposons plusieurs algorithmes de transformation nécessaires aux solveurs pour obtenir les solutions optimales du système. Ces algorithmes sont définis indépendamment du langage de modélisation système et permettent de générer une description du problème CSMOP associé au modèle. Nous prenons pour hypothèse que la sémantique de ce langage de modélisation repose sur des blocs, des attributs, des relations de composition, sur la notion d'allocation et dispose de commentaires.

Ces algorithmes sont indépendants et répartis sur les trois étapes de la méthodologie d'optimisation du système définies ci-après. Ces trois étapes peuvent être conduites successivement ou indépendamment dans le cas où le concepteur ne souhaite optimiser qu'un seul aspect du système.

Etape 1 : Identifier les architectures optimales pour la plateforme matérielle du système, en prenant en compte le niveau de redondance des composants (les blocs) ainsi que les choix d'instances des blocs.

Etape 2 : Déterminer les valeurs optimales pour les attributs des instances de bloc déterminées après la première étape, ou d'un modèle équivalent.

Etape 3 : Obtenir l'allocation optimale des composants logiciels sur les blocs de la plateforme matérielle, à partir du résultat de l'étape 2 ou d'un modèle équivalent.

2.6.1. Etape 1-Transformation pour le choix de composant et la redondance

Cette transformation va permettre, à partir du modèle d'un système enrichi avec les points de décisions (Définition 4: Système enrichi avec les points de décisions de la page 58) , de créer les variables du problème CSP. Le choix de ces variables est déterminant pour résoudre efficacement le problème d'optimisation, et nous proposons de différencier trois cas selon les types de points de décision affectés à un bloc B :

1. Le bloc B dispose de deux points de décision combinés DP_s et DP_i. Nous associons à B une matrice de booléens $x_{ij} \in \{0,1\}$
2. Le bloc B dispose d'un point de décision de structure DP_s seul. Pour ce degré de liberté nous créons une variable d'optimisation x de type entier avec $x \in [1..\mathrm{m}]$ indiquant le niveau de redondance du bloc B
3. Le bloc B dispose d'un point de décision d'instance DP_i seul. Pour ce seul degré de liberté, nous créons une variable d'optimisation x de type entier énuméré. Chaque valeur de l'énuméré identifiant un composant dans la liste des instances utilisables.

L'Algorithme 2 a pour objectif d'analyser le modèle et de créer les variables correspondant aux points de décision insérés par le concepteur.

A partir de l'ensemble des points de décisions DP du modèle d'entrée nous commençons par traiter les points de décisions de structure dp_s (ligne 4-20) et vérifier la cohérence de l'ensemble.

Lorsque que le test de cohérence est positif, nous vérifions si le nœud fils comporte un point de décision d'instance dp_i. Si c'est le cas nous sommes dans le cas d'un point de décision combiné (ligne 11). Nous créons alors la variable correspondante et nous enlevons le point dp_i de l'ensemble DP. Dans l'autre cas nous sommes en présence d'un point de décision de structure simple.

Une fois que les points de décisions dp_s sont traités, il reste à analyser les points de décisions dp_i.

Création des variables pour le choix de composant et la redondance de bloc

```
01: Input: DecisionPoint set DP = {DP_s ∪ DP_i}
02: boolean ErrorFound ← False
03: optimization problem P ← ∅
04: For dp_s in DP_s :
05:     el ← dp_s.GetAnnotatedElmt( )
06:     If not (el.GetElmtType( ) == Composion) : ErrorFound =True
07:     childNode ← el.GetMemberEnd( )
08:     If not (childNode.GetElmtType( ) == Block) : ErrorFound =True
09:     card ← el.GetPartMultiplicity( )
10:     If (card < 2 || card > NMax) : ErrorFound =True
11:     If (dp_i ← childNode.GetDecisionPoint( ) ∈ DP_i) :
12:         CompSize ← dp_i.GetListSize( )
13:         boolean IS_Array[Card][CompSize]
14:         P.AddVariable( IS_Array)
15:         DP_i.remove(dp_i)
16:     Else:
17:         integer Redundancy = {1,2..card}
18:         P.AddVariable( Redundancy)
19:     Endif
20: Endfor
21: For dp_i in DP_i:
22:     CompSize ← dp_i.GetListSize( )
23:     enum CompList = {1,2..Compsize}
24:     P.AddVariable( CompList)
25: Endfor
26: Return ErrorFound, P
```

Algorithme 2 Création des variables pour le choix de composants

Le tableau suivant présente les variables utilisées par l'Algorithme 2:

Nom	Type	Description
dp_s	StructureDecision	Point de décision de structure correspondant à une redondance
dp_i	InstanceDecision	Point de décision d'instance correspondant à un choix de composants
el	Element	Elément du modèle
P	Problem	Instance du problème d'optimisation utilisée par le solveur
IS_Array	Boolean [][]	Variable du problème d'optimisation pour la redondance et le choix de composants combinés
Redundancy	enum	Variable du problème d'optimisation pour la redondance simple
CompList	enum	Variable du problème d'optimisation pour le choix de composants

Tableau 4: Variables utilisées

Le tableau suivant présente les primitives utilisées par l'Algorithme 2:

Nom	S'applique à	Description
$\boldsymbol{GetAnnotatedElmt}()$	el : Element	Renvoie l'élément du modèle associé au point de décision (commentaire).
$\boldsymbol{GetElmtType}()$	el : Element	Renvoie le type de l'élément : bloc ou relation de composition
$\boldsymbol{GetMemberEnd}()$	Rc : Composite relation	Renvoie le nœud bloc impliqué dans une relation de composition, du côté « partie ».
$\boldsymbol{GetPartMultiplicity}($	Rc : Composite relation	Renvoie la cardinalité de la relation de composition, du côté « partie »
$\boldsymbol{GetDecisionPoint}()$	el : Element	Renvoie le point de décision associé à un élément, sous forme de commentaire
$\boldsymbol{GetListSize}()$	dp_i : Instance Decision Point	Renvoie le nombre d'instances associées à un dp_i.
$\boldsymbol{AddVariable}()$	P : Optimization problem	Rajoute une variable au problème d'optimisation

Tableau 5: Primitives de l'algorithme

Test de cohérence

Les tests de cohérence permettent de vérifier que le modèle est conforme à la Définition 4: Système enrichi avec les points de décisions. Pour chaque point de décision dp_s nous vérifions qu'il s'applique bien à une relation de composition unique (ligne 6). Puis nous vérifions que cette relation de composition associe elle-même un nœud père unique de type bloc à

un nœud fils unique (childNode) de type bloc également (ligne 8). Pour chaque point de décision dp_i nous vérifions qu'il s'applique bien à un nœud unique de type bloc et que la liste des instances est disponible avec au moins deux éléments.

Analyse de la complexité

Dans cet algorithme, nous avons choisi d'effectuer une recherche linéaire des points de décision. Avec cette méthode, le pire cas est un modèle disposant d'un grand nombre de points de décisions combinés, avec un grand nombre d'instances associées à chaque bloc. Dans ce cas la complexité de l'algorithme est de $O(mn)$, où m est le nombre de points de décision de structure DP_s du modèle et n est le nombre maximum d'instances associées à un point de décision d'instance DP_i.

2.6.2. Etape 2-Transformation des attributs

A partir de l'ensemble des points de décision DP du modèle d'entrée nous obtenons les points de décisions de valeur « *ValueDecision* ». Pour chacun de ces points de décision, nous vérifions qu'il est bien rattaché soit à un attribut de type entier pour un « *IntegerValDecision* » soit à un attribut de type réel pour un « *RealValDecision* ». Nous récupérons les intervalles de chacun de ces points de décision, puis nous créons une variable correspondant du problème CSMOP, en précisant son type et ses bornes.

Le tableau suivant présente les primitives utilisées par cette transformation :

Nom	S'applique à	Description
$\boldsymbol{GetAnnotatedElmt}()$	el : Element	Renvoie l'élément du modèle associée au point de décision.
$\boldsymbol{GetElmtType}()$	el : Element	Renvoie le type de l'élément : attribut
$\boldsymbol{GetAttribute}()$	el : Element	Renvoie une propriété d'un bloc
$\boldsymbol{AddVariable}()$	P : Optimization problem	Rajoute une variable au problème d'optimisation

2.6.3. Etape 3-Transformation pour l'allocation

Cette transformation va permettre, à partir du modèle d'un système disposant d'une spécification de déploiement, de créer les variables du problème CSMOP. Le système (Définition 5: Allocation variable) est constitué d'un ensemble de ressources à allouer et d'un ensemble de cibles pour l'allocation. Deux algorithmes sont utilisés selon le type de variables que l'on souhaite générer. Dans les deux cas, la spécification de déploiement (Figure 31 « allocation variable et contraintes de la page 62) est analysée. Cette spécification est une liste d'allocations variables «v». Chaque allocation variable contient un élément source «s» ainsi qu'un ensemble d'éléments cibles : «$Target\_List$».

Le traitement des cas incohérents concerne l'analyse d'une d'allocation variable. Pour chaque allocation variable, on vérifie qu'un unique élément source est présent, et que l'ensemble des éléments de destination n'est pas vide.

Le premier algorithme (Algorithme 3 Création des variables de type énuméré) permet de générer des variables conformes à l'Équation 2: formulation en nombre entier pour l'allocation de la page 63. Pour chaque allocation variable v, une variable de type énuméré est créée, avec des valeurs correspondant aux identifiants des éléments cibles.

Création des variables de décision de type énuméré pour l'allocation variable

01:**Input**: $VA = \{Deployment\ Specification\ of\ the\ model\}$
02: $boolean\ ErrorFound \leftarrow False$
03: $optimization\ problem\ P \leftarrow \emptyset$
04: $EnumList \leftarrow \emptyset$
05: $variableAllocate\ v$
06: $Source\ s$
07: $Target\_List\ TA$
08: $Target\ t$
09: **For** v **in** VA :
10: $TA \leftarrow \emptyset$
10: $s \leftarrow v.getFrom()$
11: $TA \leftarrow v.getTo()$
12: **If** $(card(s) \neq 1\ or\ TA = \emptyset)$
13: $ErrorFound \leftarrow True$
14: **Else**
15: **For** t **in** TA :
16: $EnumList.Add(t.id)$
17: **Endfor**
18: $P.CreateEnumVariable(s, EnumList)$
19: **Endif**
20: Endfor
21:**Return** $P, ErrorFound$

Algorithme 3 Création des variables de type énuméré

Le second algorithme (Algorithme 4 Création des variables de type booléen) permet de générer des variables conformes à l'Équation 3: Formulation booléenne pour l'allocation de la page 65. Dans cette configuration, une matrice de variables booléennes correspondant à la spécification de déploiement tout entière est créée. Un premier ensemble de contraintes booléennes est ajoutée au problème (ligne 8), pour chaque élément source à allouer, donné par l'Équation 10 «contrainte d'unicité pour l'allocation».

Ensuite, les contraintes booléennes correspondant à l' Équation 11 « éléments allouables» de la page 69 sont rajoutées au problème CSMOP (ligne 14).

Création des variables de décision de type booléen pour l'allocation variable

01:**Input**: $VA = \{Deployment\ Specification\ of\ the\ SysML\ model\}$
02: $CSP\_Problem\ P \leftarrow \emptyset$
03: Target$List\ TA_1 \leftarrow Model.getTargetElement()$
04: $int\ P_{max} = TA_1.Card()$
05: $int\ S_{max} = VA.Card()$
06: $P.CreateBoolMatrixVariable(s, S_{max}, P_{max})$
07: **For** i **in** $[0..S_{max}[$:
08: $P.AddGlobalConstraint$("AtMostOne(s[i])")
09: **For** v **in** VA :
10: $TargetList\ TA_2 \leftarrow v.getTo()$
11: **For** t **in** TA_1 :
12: **If** $(t \notin TA_2)$
13: $P.AddBoolConstraint("s[i][j] = False")$
14: $j++;$
15: **Endfor**
16: $i++$
17: Endfor
18:Return P

Algorithme 4 Création des variables de type booléen

Le tableau suivant présente les variables utilisées par les algorithmes d'allocation :

Nom	Type	Description
VA	DeploymentSpecification	L'ensemble des allocations variables contenu dans le modèle d'entrée.
v	VariableAllocate	L'élément courant d'allocation variable du modèle
s	Source	L'élément source courant à allouer
TA	TargetList	L'ensemble des éléments cibles (Target) correspondant à l'élément source courant
t	Target	L'élément cible courant (Target), avec un identifiant "id"
P	CSP_Problem	Le problème CSP à construire

Tableau 6:Variables pour l'algorithmes d'allocation

Le tableau suivant présente les primitives utilisées par les algorithmes d'allocation :

Nom	Description
GetFrom(), GetTo()	Récupère la liste des éléments «Source» et «Target» de la relation n-aire «*variableAllocate*».
CreateEnumVariable(), CreateBoolMatrixVariable()	Création d'un variable du problème CSP
AddBoolConstraint()	Rajoute une contrainte booléenne au problème CSP
AddGlobalConstraint()	Rajoute une contrainte globale au problème CSP

Tableau 7: Primitives pour les algorithmes d'allocation

Test de cohérence

Les tests de cohérence permettent de vérifier que la spécification d'allocation est conforme à la Définition 5: Allocation variable de la page 62.Pour chaque allocation variable, nous vérifions que cette relation dispose d'un unique élément source ainsi que d'un ensemble non vide d'éléments de destination.

Analyse de la complexité

Dans ces deux algorithmes, nous avons choisi d'effectuer une recherche linéaire des allocations variables. Le pire cas est alors un modèle disposant d'un grand nombre d'allocations variables, avec un grand nombre d'éléments de destination pour chaque allocation. Dans ce cas l'ordre de grandeur est $O(mn)$, où m est le nombre de points d'allocations du modèle et n est le nombre maximum d'éléments de destinations associés à un point d'allocation.

2.7.Conclusion

Dans ce chapitre nous avons proposé plusieurs concepts nécessaires à l'intégration de l'optimisation dans l'ingénierie MBSE, indépendamment d'un langage particulier. Ce langage peut ainsi être enrichi avec des « points de décision » permettant de modéliser la redondance variable, un choix de composants ou l'allocation variable. A partir de ces points de décision s'appliquant à une classe du système, il est possible de générer les variables du problème d'optimisation CSMOP, avec plusieurs algorithmes que nous avons décrits. Selon la combinaison de point de décision présente dans le modèle, un type de variable particulier est créé. Des contraintes spécifiques s'appliquant à ces variables viennent également compléter le problème CSMOP, et peuvent être créées automatiquement à partir des points de décision et des contraintes. Les points de décisions, les contraintes ainsi que les fonctions objectifs sont réunis dans un contexte d'optimisation, permettant au concepteur de manipuler l'ensemble de ces notions de façon cohérente. Dans le chapitre suivant nous allons montrer comment ces concepts et algorithmes peuvent s'intégrer dans le langage SysML (Friedenthal, 2014) d'ingénierie système.

3. Implémentation dans SysML/UML et intégration des solveurs

Dans ce chapitre nous présentons l'intégration dans le langage SysML des concepts développés précédemment. En effet comme nous l'avons vu dans la section 1.1, le langage SysML peut être adapté à des besoins d'analyse par la création de stéréotypes. Une première partie définit les concepts présentés dans le chapitre 2 par extension des classes de base UML (méta-classe) ou par spécialisation d'un stéréotype. Une nouvelle utilisation des diagrammes BDD et paramétrique de SysML est également proposée pour l'optimisation. Une seconde partie adapte les algorithmes de transformation proposés dans le paragraphe 2.6 pour prendre en entrée le métamodèle UML/SysML ainsi que les nouveaux stéréotypes proposés. Le résultat de cette transformation est une définition d'un problème d'optimisation CSMOP, pouvant être résolu par un solveur tel que ceux présentés dans la section 1.2.

Enfin, pour permettre la mise en œuvre des différents solveurs, nous proposons une description du problème CSMOP au format EBNF, ce qui permet de rester indépendant d'un solveur particulier.

3.1. Stéréotypes « points de décision »

3.1.1. Stéréotypes pour le choix de composants et la redondance

Dans le langage SysML et en particulier dans les digrammes BDD et IDB, un composant est décrit par un bloc (block) et constitue la brique de base du système. Un bloc est décomposable et peut également contenir un comportement. La structure des composants du système est décrite dans un diagramme de définition de blocs « BDD ». Les points de décisions d'instance et de redondance proposés dans la section 2.2 sont définis par des commentaires spécifiques associés à certains éléments du modèle SysML. La Figure 36 «Points de décision dans SysML» présente ces différents éléments.

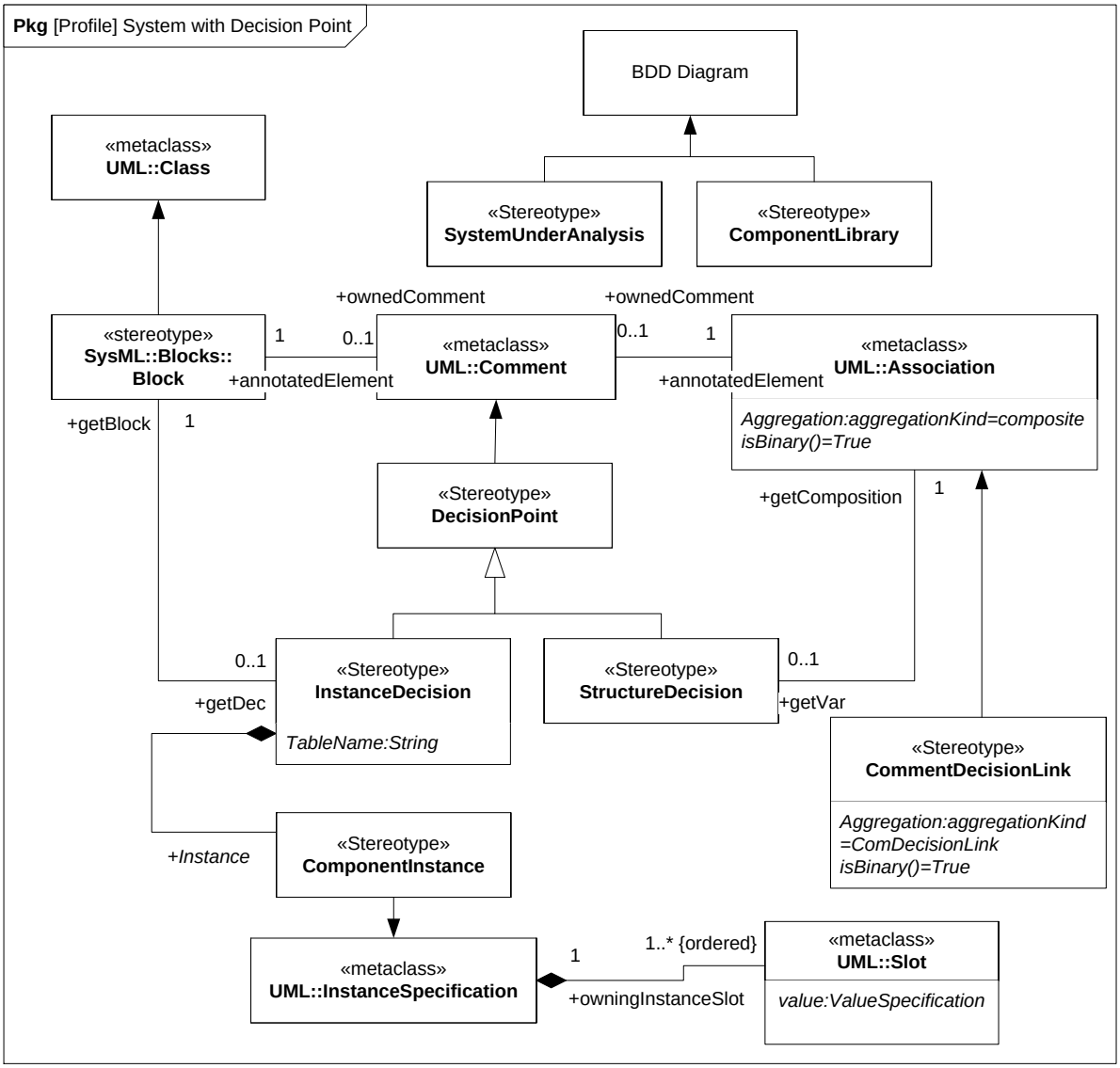


Figure 36 «Points de décision dans SysML»

Le système à analyser « *SystemUnderAnalysis* » est un stéréotype défini par extension du diagramme de définition de bloc BDD SysML. Les éléments suivants du diagramme BDD sont considérés : les blocs « *SysML ::Block* », les commentaires « *UML ::Comment* » et les relations de compositions binaires « *UML ::Association* ». La librairie de composants « *ComponentLibrary* » est également une extension du diagramme BDD. Ce diagramme contient les instances de blocs du système à analyser « *UML ::InstanceSpecification* ».

La notion de point de décision est introduite sous forme d'un stéréotype « *DecisionPoint* » par extension de la classe « *UML ::Comment* ».

Le stéréotype « *InstanceDecision* » permet de représenter un choix d'instance pour un bloc donné. Le stéréotype « *StrutureDecision* » représente quant à lui la redondance d'un bloc et s'applique à une relation de composition « *UML ::Association* ». Deux relations « *getBlock* » et « *getCompositon* » permettent d'obtenir les objets reliés à ces commentaires, soit respectivement un bloc et une relation de composition.

Le choix d'instance « *InstanceDecision* » se compose d'une liste ordonnée d'instances « *ComponentInstance* ». Cette liste d'instance dispose d'au moins deux éléments, pour proposer un choix entre deux instances de composant lors du processus d'optimisation.

Une instance « *ComponentInstance* » est une extension de la classe « *UML ::InstanceSpecification* » et correspond à une instance du bloc associé au point de décision d'instance par la relation « *getBlock* ». Cette instance contient une liste d'au moins une valeur de type « ***ValueSpecification*** » contenue dans un emplacement «***UML ::Slot »***. Chaque slot correspond à un attribut et à sa valeur imposée, pour le block correspondant (relation *GetBlock*). Le stéréotype « ***CommentDecisionLink*** » permet d'associer un point de décision à un bloc ou à une relation de composition.

Un exemple de modèle enrichi avec ces stéréotypes est présenté Figure 61 «Système enrichi avec les points de décisions» de la page 109.

3.1.2. Stéréotypes pour la variation des attributs

La variation des attributs est indiquée sous forme de commentaires spécifiques rattachés aux attributs du modèle. Dans la Figure 37 «Variation d'attribut dans SysML» les stéréotypes présentés permettent d'analyser le modèle pour générer la description du problème d'optimisation.

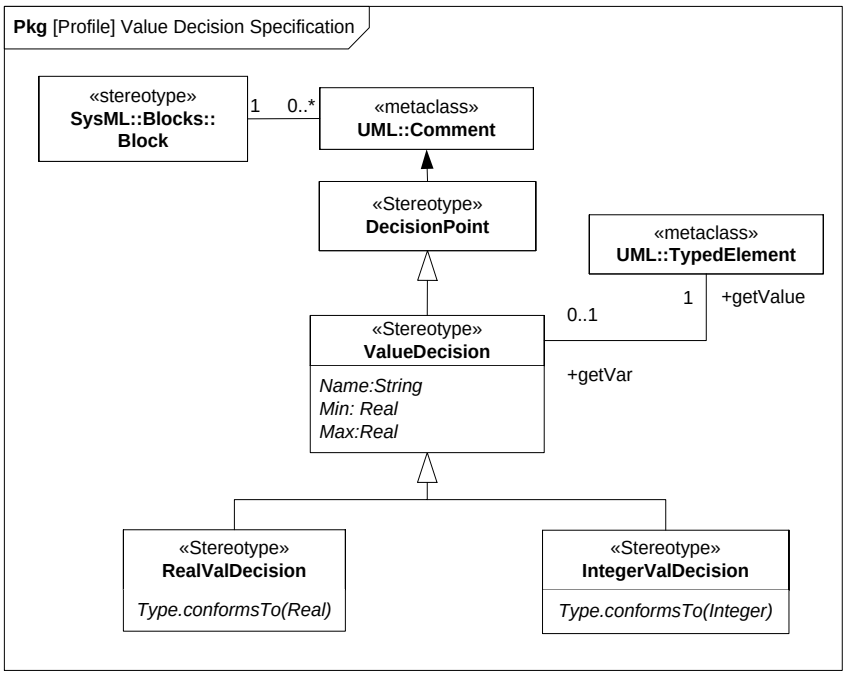


Figure 37 «Variation d'attribut dans SysML»

Exemple d'utilisation

La Figure 38 «Exemple de point de variation sur les attributs» présente un cas d'utilisation du stéréotype « ***RealValDecision***». Le bloc « *StereoCamera*» modélise une caméra stéréos-

copique embarquée dans un véhicule terrestre. L'attribut « *hPosition* » représente la hauteur de positionnement de la caméra par rapport à la base du véhicule et «*pitchAngle*» l'angle d'inclinaison de la caméra par rapport à la route. Ces deux paramètres influent sur les performances de la caméra, pour obtenir un compromis entre la détection d'obstacle proche et la détection d'objets éloignés. Nous affectons donc deux points de décision « ***RealValDecision***» à ces attributs de type réel et dont les valeurs sont bornées.

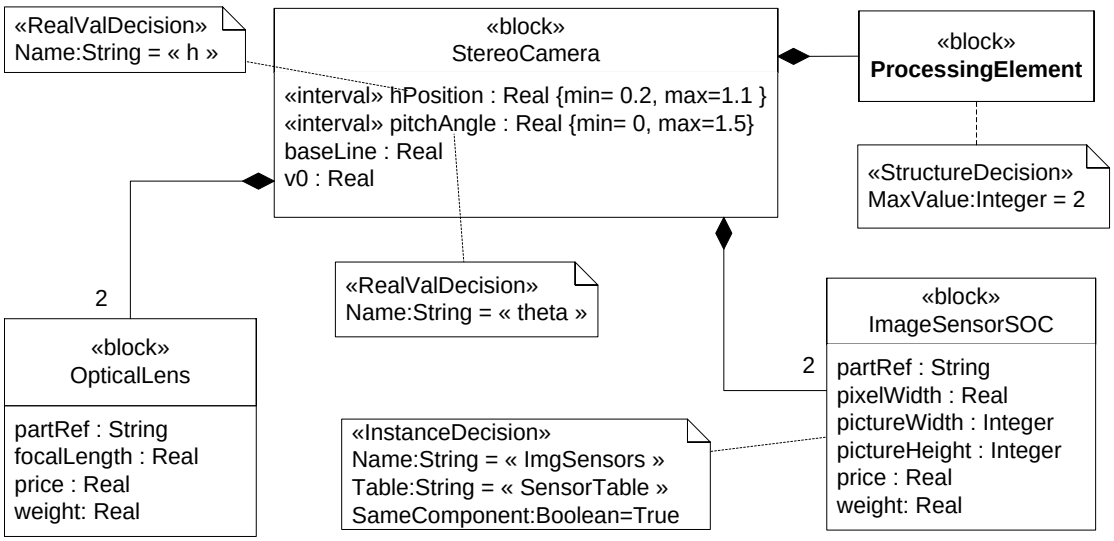


Figure 38 «Exemple de point de variation sur les attributs»

3.1.3. Stéréotypes pour l'allocation variable

SysML (pour les systèmes complexes) et MARTE (pour les systèmes embarqués HW/SW) proposent des mécanismes pour spécifier une relation d'allocation entre des éléments « sources » et des éléments « destinations ». Cependant les mécanismes existant dans ces deux langages ne permettent pas d'exprimer le concept d'allocation variable dont nous avons besoin.

Dans MARTE, le stéréotype *Allocate* dérive de la classe *UML ::Abstraction* dans la Figure 39 : «Allocate dans MARTE». Ce stéréotype représente une association n-aire entre un ensemble d'éléments source à allouer sur un ensemble d'éléments destination. La contrainte « *impliedConstraint* » permet de spécifier un coût pour chaque allocation d'un élément source vers un élément de destination. L'inconvénient de ce stéréotype est qu'il ne permet pas de différencier une allocation réelle d'une allocation possible exploitable dans un contexte d'optimisation. De plus la spécification d'un coût pour chaque allocation est contraignante dans un système avec un grand nombre d'éléments à allouer. Ce stéréotype « Allocate » est représenté par une flèche en pointillé, restreignant son utilisation à une situation où les sources et les destinations sont représentées sur le même diagramme.

Un second stéréotype « Assign » est également présent dans le profile MARTE. Il dérive du commentaire « UML ::Comment ». L'avantage de « Assign » est que l'élément source n'est pas modifié, ce qui n'est pas le cas de « Allocate », spécialisation de l'association UML.

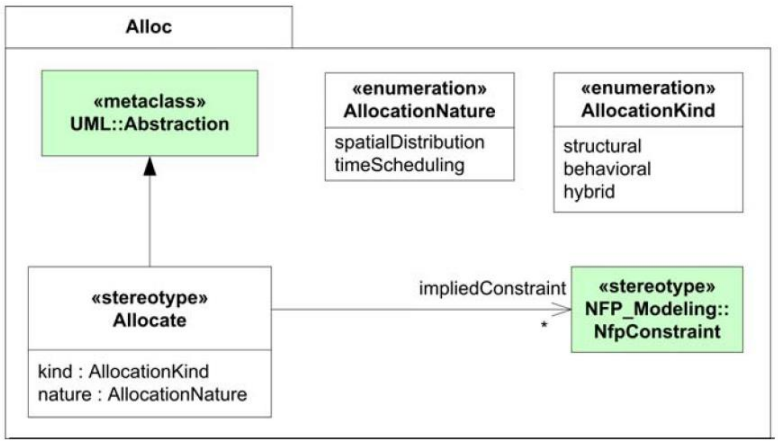


Figure 39 : «Allocate dans MARTE»

Dans SysML, un stéréotype « Allocate » est également proposé. Il permet de représenter une allocation réelle, comme dans SysML, mais avec un seul élément source et un seul élément destination (relation binaire orientée). Ce stéréotype est utilisé pour allouer des *blocks*, *parts* ou des *connectors* entre eux. On peut également l'utiliser pour allouer une exigence à un élément du modèle. A titre d'exemple, la Figure 40 «Allocate dans SysML » tirée de (SysML, 2015) Alloue des éléments abstraits à des éléments concrets, en utilisant de multiples associations « *allocate* » :

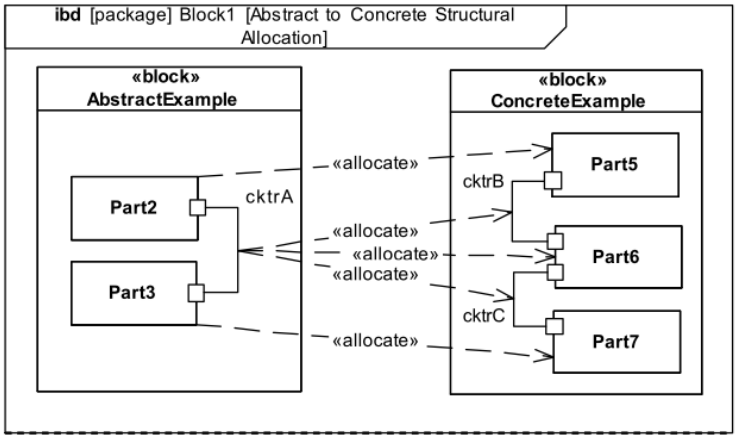


Figure 40 «Allocate dans SysML »

Pour intégrer la notion d'allocation variable dans MARTE et SysML, nous proposons les stéréotypes suivants dans la Figure 41:«Stéréotypes pour l'allocation variable» et leurs relations avec les classes de base SysML ou UML. Nous conservons les deux représentations possibles, l'une basée sur le commentaire « UML ::Comment » et l'autre sur le stéréotype « MARTE ::Allocate ». Par contre un seul élément source « *From* » est présent dans chacun des nouveaux stéréotypes « *VariableAllocComment* » et « *VariableAllocate* ». Ceci nous permet d'une part d'associer chaque élément du modèle portant l'un de ces stéréotypes à une variable de décision et, d'autre part, de contraindre plusieurs allocations entre elles. De même le coût de l'allocation d'un élément source sur un élément destination n'est plus

porté nécessairement par la relation d'allocation mais peut l'être par les éléments sources et destination.

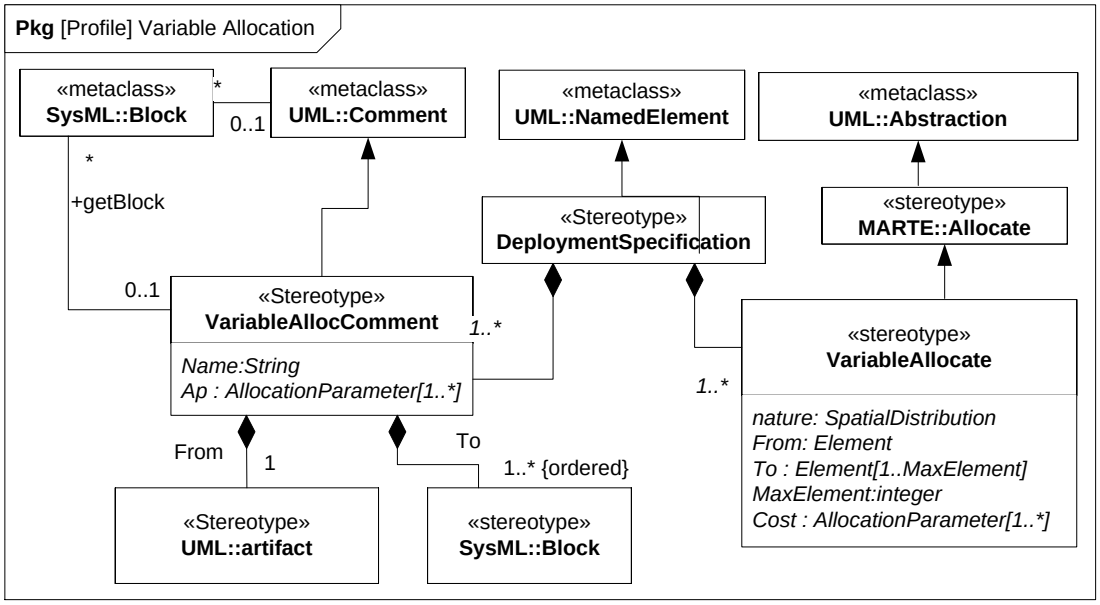


Figure 41:«Stéréotypes pour l'allocation variable»

Un ensemble d'allocations peut être constitué, en utilisant le stéréotype « DeploymentSpecification ». Cet ensemble est utilisé pour générer un ensemble cohérent de variables du problème d'optimisation.

Exemple d'utilisation

La Figure 42«Exemple d'allocation variable» présente un cas d'utilisation du stéréotype « ***VariableAllocComment***» pour un modèle de taille restreinte. Le diagramme représente la modélisation d'un système embarqué multitâches, avec une plate-forme d'exécution de type processeur multi-cœurs. La plate-forme dispose de quatre cœurs à haute performance « BigCore » et de quatre cœurs moins consommateurs d'énergie « LittleCore ». L'allocation variable d'une tâche «GetFrame» sur un ensemble de deux cœurs BigCore[0] et LittleCore[0] est représentée avec le nouveau stéréotype « ***VariableAllocComment***».

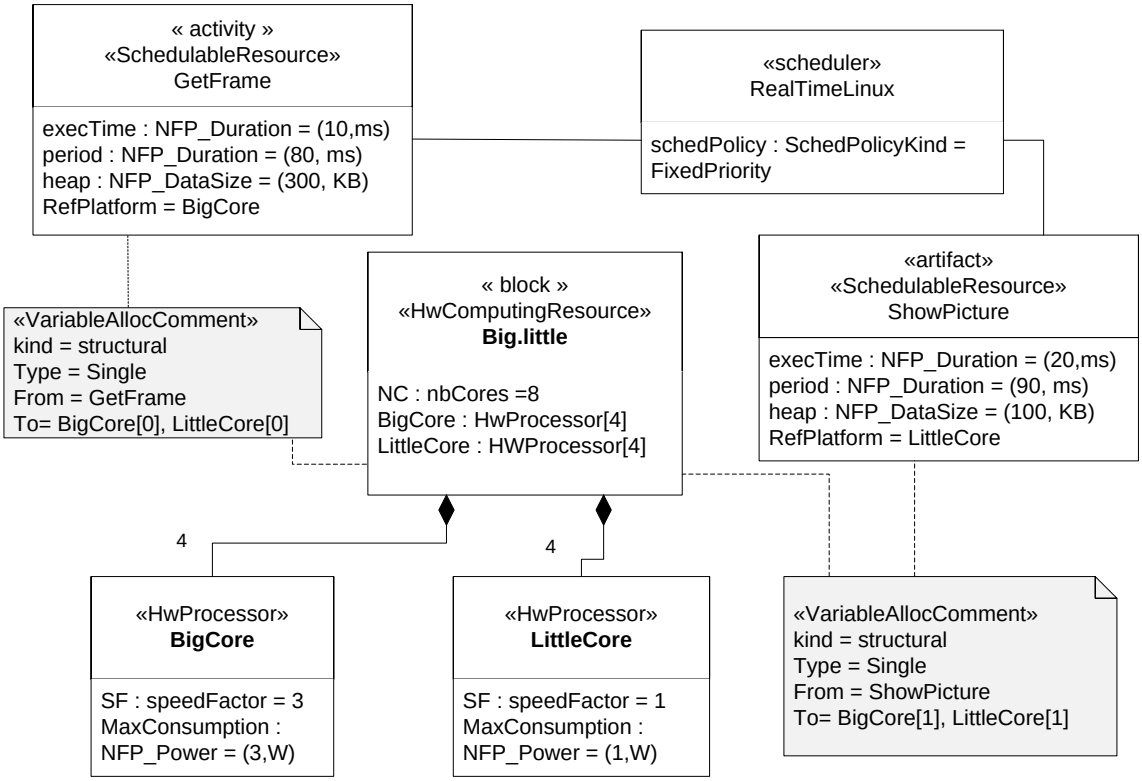


Figure 42«Exemple d'allocation variable»

Dans le cas d'un modèle plus étendu, le Tableau 8 «forme tabulaire de l'Allocation variable » présente l'utilisation du stéréotype « ***VariableAllocate***» sous forme de tableau avec l'outil Papyrus.

		A	B	C	D	E
		name : String [0..1]	client : NamedEleme...	supplier : NamedEle...	/owner : Element [0....	ownedComment : Co..
0	<Abstraction> VideoApp...	VideoApplication	[OV7962, OV5642]	[]	SysMLmodel	[]

Tableau 8 «forme tabulaire de l'Allocation variable »

3.2.Nouvelle utilisation de diagrammes existants

3.2.1. Le diagramme d'instance

Le diagramme d'instance de SysML peut être utilisé lorsque l'on rajoute des points de décision « InstanceDecision » à des blocs du modèle. La Figure 43 «Diagramme d'instance» présente des instances de blocs, avec le stéréotype « componentInstance ». Ce stéréotype permet aux algorithmes de génération de récupérer les informations des blocs et leurs attributs. Les blocs de type « VideoSensor », « ProcessingElement » et « NetworkTransceiver » sont instanciés. Les valeurs des attributs sont utilisées par les fonctions objectifs, pour calculer la fiabilité globale du système ou bien son coût. Ces valeurs sont contenues dans des « Slots ».

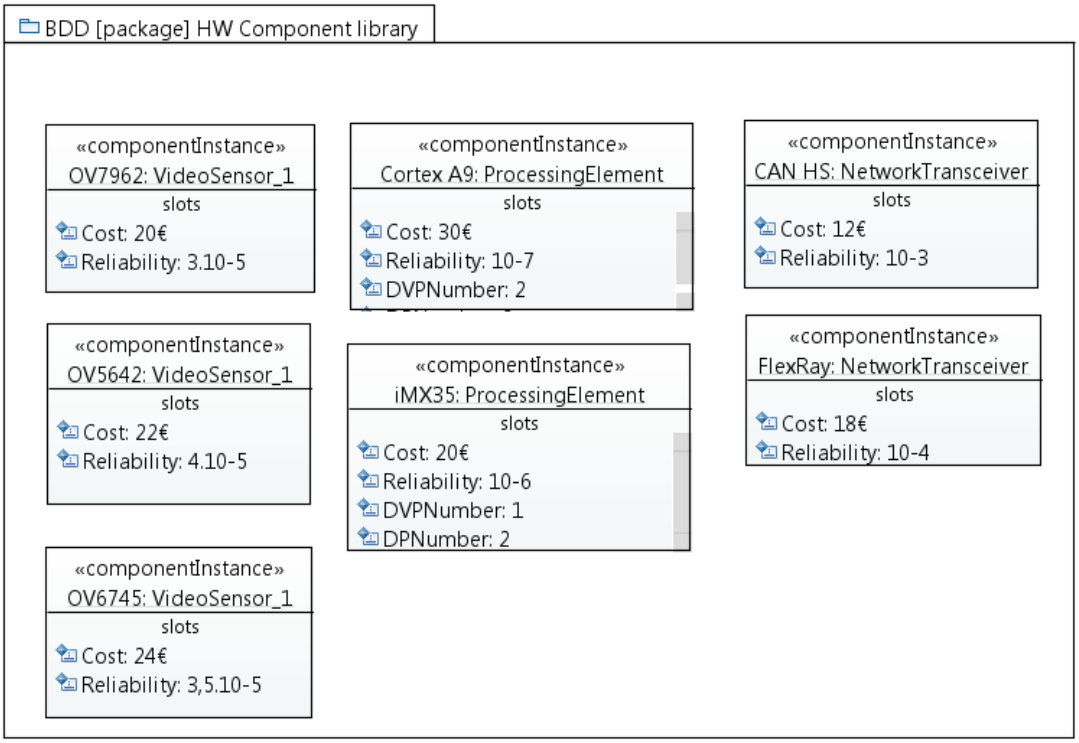


Figure 43 «Diagramme d'instance»

3.2.2. Les contraintes

Les contraintes nécessaires au problème d'optimisation (2.4) peuvent être représentées dans le langage SysML en utilisant blocs de contrainte et diagrammes paramétriques. En SysML un mécanisme générique exprime les contraintes s'appliquant à un système, sous forme d'une expression textuelle s'appliquant à un élément du modèle. SysML ne fournit pas de langage d'expression de contrainte, car selon le domaine d'utilisation on peut utiliser OCL (Object Constraint Language), Java ou une expression mathématique avec MathML. La définition de la contrainte peut inclure le langage utilisé pour la définir. La contrainte peut être représentée par une note, semblable à un commentaire, et lié à l'élément concerné. Elle peut également être intégrée dans le compartiment d'un bloc, représentant une propriété du bloc.

Lorsque l'on souhaite ré-utiliser une contrainte, il est préférable d'employer un bloc de contrainte « *Constraint Block* ». Un bloc de contrainte peut alors être intégré dans un réseau de contraintes, pour représenter une contrainte globale plus complexe. Le bloc de contrainte SysML se compose d'un ensemble de paramètres typés ainsi que d'une expression décrivant l'évolution des paramètres entre eux. Une fois défini, le bloc de contrainte peut être instancié dans un diagramme paramétrique. Les paramètres de chaque bloc sont alors reliés aux paramètres des autres blocs ou bien à des propriétés d'autres blocs du modèle. Dans la Figure 44 «Contraintes dans SysML» tirée de (Friedenthal, 2014), plusieurs blocs de contraintes sont utilisés pour représenter la dynamique d'un véhicule, avec des paramètres

comme la puissance, la vitesse et l'accélération. Un premier diagramme BDD présente la hiérarchie des contraintes. Une contrainte de haut niveau « VehicleDynamics » est représentée par un bloc de contrainte « *VehicleDynamics* » disposant uniquement de paramètres. Ce bloc se compose lui-même de quatre blocs de contrainte dont les blocs « *PowerEquation* », « *PositionEquation* ». Ces blocs de contrainte disposent chacun de paramètres et d'une expression mathématique. Un second diagramme, de type paramétrique, représente les connections entre les paramètres des contraintes précédentes.

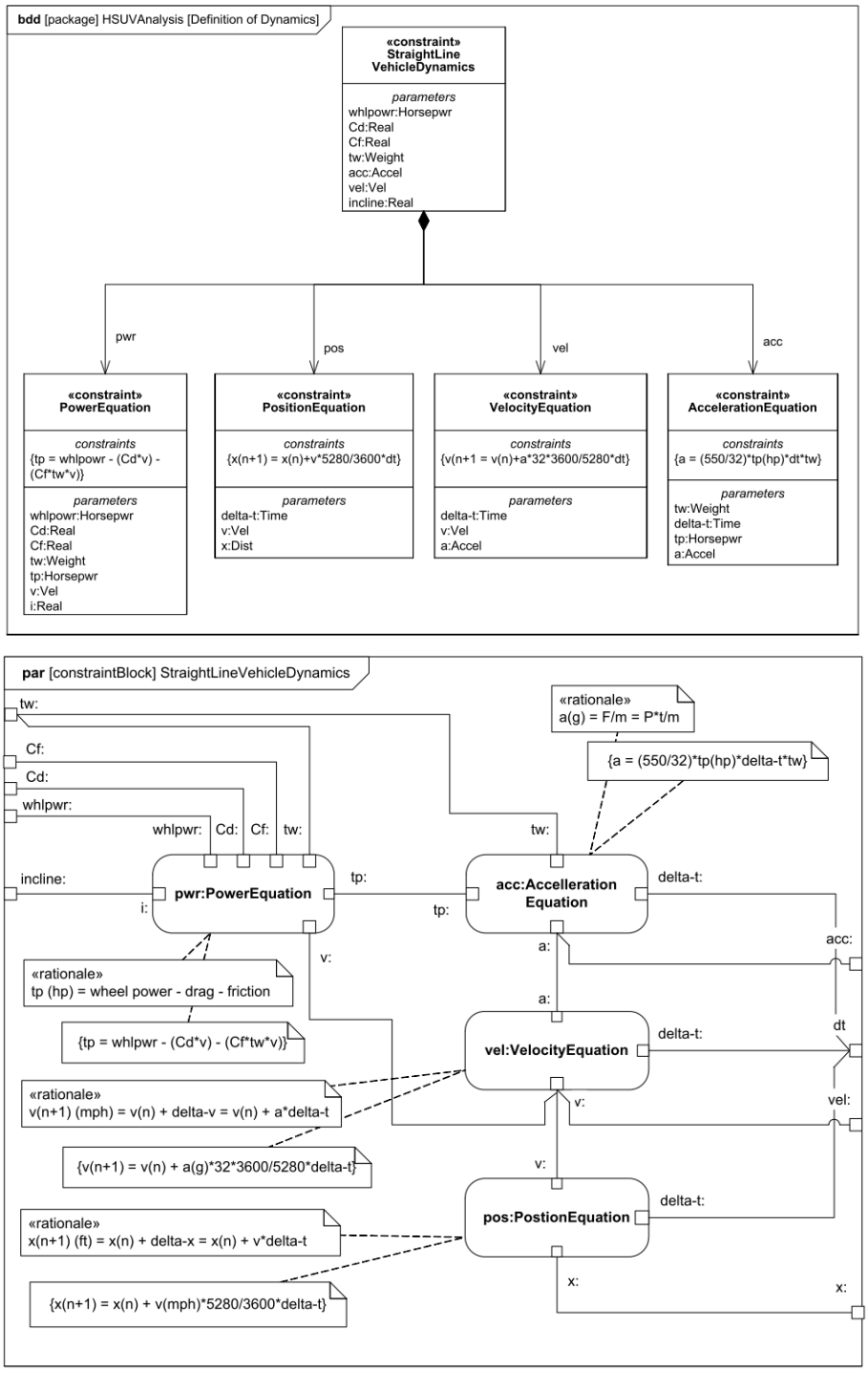


Figure 44 «Contraintes dans SysML» de (Friedenthal, 2014)

3.2.3. Le contexte d'optimisation

Pour permettre l'optimisation d'un système modélisé avec SysML ou MARTE, nous proposons une nouvelle extension du diagramme paramétrique « MDO Context Diagram » (Multi-Domain Optimization Context) ainsi qu'un ensemble de stéréotypes associés. La Figure 45 «MDO Context dans SysML» présente les relations entre les nouveaux stéréotypes pour l'optimisation et les classes de base UML ou SysML. Le diagramme paramétrique est utilisé pour représenter le contexte d'optimisation. Il s'agit bien d'une nouvelle utilisation et non d'un nouveau diagramme. Ce diagramme est associé à un BDD représentant le bloc de haut niveau « MDO Context Block », ainsi que sa hiérarchie de blocs. Les blocs « ParetoFront-Block », « ObjectiveFunction » et « OptimizationModel » sont également des spécialisations de blocs de contrainte SysML. Le système à analyser « SystemBlock » ainsi que l'environnement sont représenté par des blocs SysML.

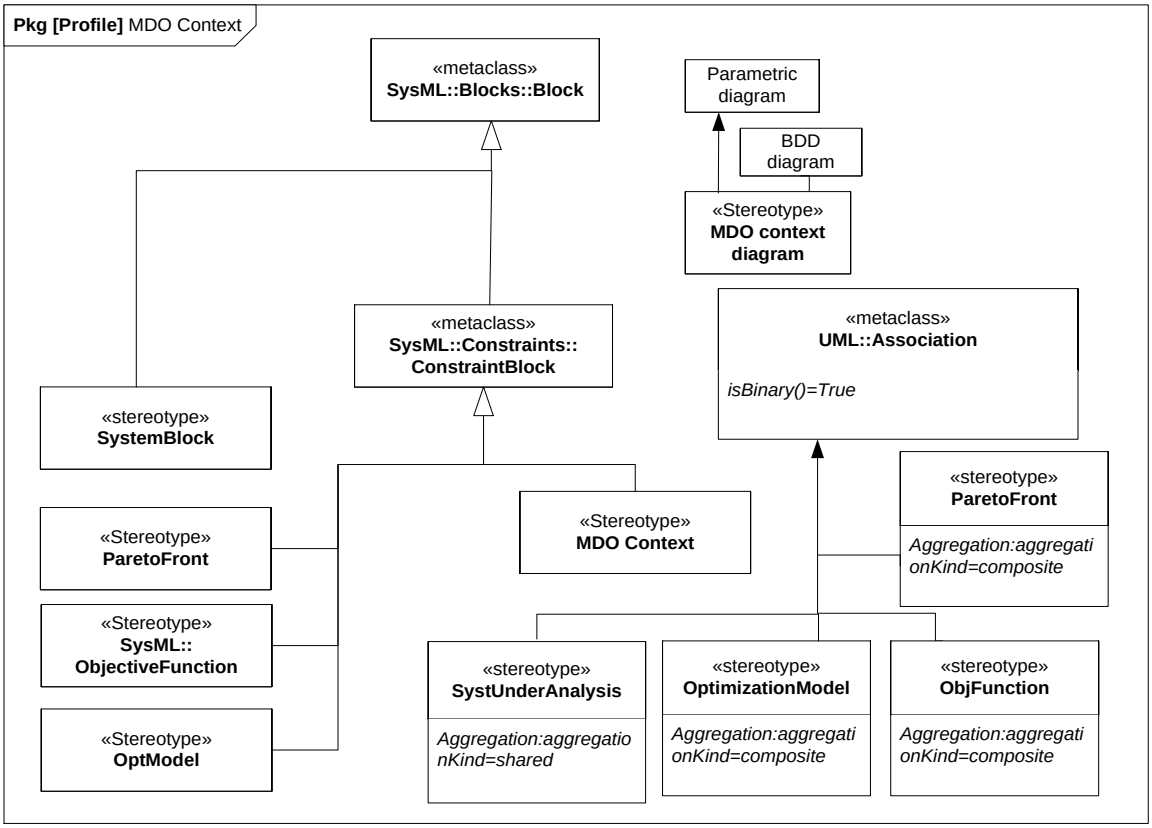


Figure 45 «MDO Context dans SysML»

Exemple de contexte d'optimisation SysML

La Figure 46 «contexte d'optimisation (BDD) »présente le diagramme de définition de bloc d'un contexte d'optimisation multi-domaine « ***MDO context*** ». Ce diagramme réalisé avec l'outil Papyrus intègre deux objectifs d'optimisation : le coût des éléments matériels et la fiabilité du système. Deux fonctions objectifs « ***SystemReliability*** » et « ***HWCostEvaluation*** » sont associées à chaque objectif.

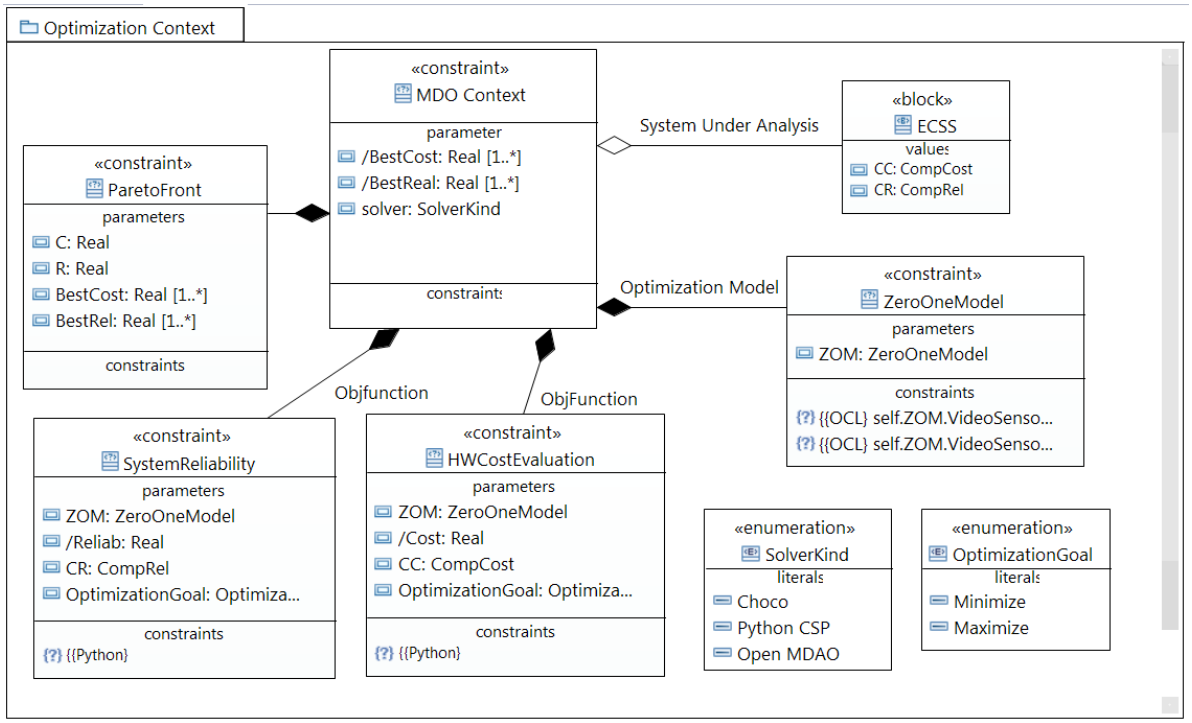


Figure 46 «contexte d'optimisation (BDD) »

La Figure 47 «Contexte d'optimisation (Diag. Paramétrique)» représente le diagramme paramétrique associé au diagramme précédent. Dans la partie droite de ce diagramme, le modèle d'optimisation est représenté par un bloc de contrainte, incluant dans ses paramètres une formulation booléenne du problème. Ces paramètres sont utilisés par les deux fonctions objectifs pour évaluer la fiabilité et le coût du système. Ces deux fonctions ont également besoin de valeurs de coût et de fiabilité des composants du modèle. Ces valeurs sont obtenues par référence sur les valeurs du système à analyser de type « *CompRel* » et « *CompCost* ». A partir de ces valeurs et des variables du modèle d'optimisation, les deux fonctions objectifs fournissent la fiabilité et le coût de chaque solution. Le bloc de contrainte ParetoFront peut alors générer l'ensemble des solutions optimales en utilisant le solveur et les fonctions objectifs.

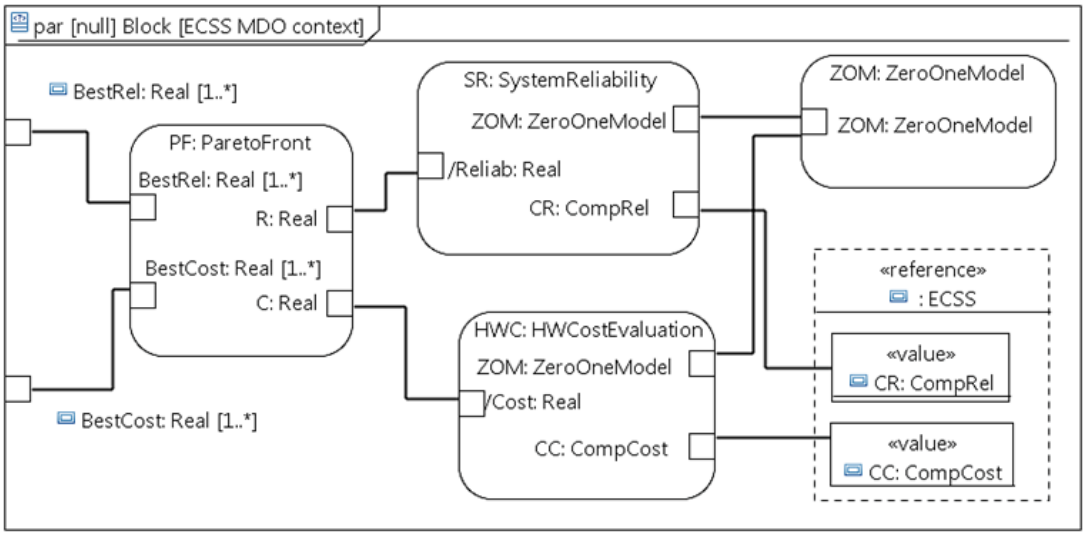


Figure 47 «Contexte d'optimisation (Diag. Paramétrique)»

3.3.Adaptation des algorithmes de transformation pour SysML

Les algorithmes de transformation proposés dans le paragraphe 2.6 sont indépendants du langage de modélisation système utilisé, nous les qualifions de « générique ». Nous proposons une adaptation de ces transformations en prenant en entrée le méta-modèle UML/SysML. Le résultat de ces transformations est une description d'un problème CSMOP, indépendante du solveur utilisé et de son méta-modèle.

3.3.1. Adaptation des algorithmes de transformations pour un modèle SysML

Etape 1-Transformation pour le choix de composant et la redondance

Dans cette transformation, le point d'entrée est le « *Package* » UML. Dans ce package, nous récupérons l'ensemble des commentaires disposant du stéréotype « *DecisionPoint* » que nous avons introduit. Pour chacun de ces commentaires nous récupérons l'ensemble des éléments qui lui sont reliés. Pour cette première étape, les variables sont présentées dans le Tableau 9 «Eléments SysML et variables génériques pour le choix de composants».

Elément UML/SysML	Variable «générique»	Commentaire
$\boldsymbol{UML}$ :: $\boldsymbol{comment}$	StructureDecision	Un commentaire UML du Package sélectionné, contenant le stéréotype « StructureDecision »
$\boldsymbol{UML}$ :: $\boldsymbol{comment}$	InstanceDecision	Un commentaire UML du Package sélectionné, contenant le stéréotype « InstanceDecision »
$\boldsymbol{UML}$ :: $\boldsymbol{Element}$	Element	Element du package UML
$\boldsymbol{UML}$ :: $\boldsymbol{Association}$		Element annoté par le commentaire, de type UML ::Classe correspondant à un bloc SysML
$\boldsymbol{UML}$ :: $\boldsymbol{Class}$		Element annoté par le commentaire, de type UML ::Association correspondant à une relation de composition SysML

Tableau 9 «Eléments SysML et variables génériques pour le choix de composants»

Le Tableau 10 «Primitives SysML pour le choix de composants» présente les primitives utilisées pour cette transformation.

Primitive UML/SysML	Primitives génériques	Description
$\boldsymbol{GetAnnotatedElmt}()$	$GetAnnotatedElmt()$	Renvoie l'élément du modèle associé au point de décision.
$\boldsymbol{instanceof}$	$GetElmtType()$	Renvoie le type de l'élément : une Classe ou une association
$\boldsymbol{GetMemberEnd}()$	$GetMemberEnd()$	Renvoie la liste des éléments impliqués dans l'association
$\boldsymbol{getLower}(), \boldsymbol{getUppe}$ $\boldsymbol{isCompostite}()$	$GetPartMultiplicity($	Renvoie la cardinalité de l'association par rapport à un élément donné.

Tableau 10 «Primitives SysML pour le choix de composants»

Etape 2-Transformation des attributs

Dans cette étape, nous analysons les propriétés d'un bloc. La relation entre les primitives génériques et les primitives SysML est directe. Il suffit pour cela de récupérer les propriétés d'un bloc et de vérifier qu'elles sont du type « *Attribute* ».

Etape 3-Transformation pour l'allocation

Dans cette transformation, le point d'entrée est un contexte d'optimisation représenté par un bloc SysML disposant du stéréotype «*MDO Context*». A partir de ce bloc nous récupérons les éléments source « SW », les éléments destination « HW », et les allocations variables. Les allocations variables sont identifiées avec le stéréotype «SysML::Allocations::Allocated » appliqué à un bloc. Les éléments SysML utilisés sont présentés dans le Tableau 11 «Eléments SysML et variables génériques pour l'allocation».

Elément UML/SysML	Variable «générique»	Commentaire
$\boldsymbol{UML :: Class}$	Source	L'ensemble des éléments source à allouer correspond à un ensemble de bloc avec le stéréotype « SW »
$\boldsymbol{UML :: Class}$	Destination	L'ensemble des éléments de destination pour l'allocation correspond à un ensemble de bloc avec le stéréotype « HW »
$\boldsymbol{SysML :: Allocations :: Allocated}$	VariableAllocate	Ce stéréotype appliqué à un bloc « SW » permet de définir l'allocation variable

Tableau 11 «Eléments SysML et variables génériques pour l'allocation»

Le Tableau 12 «Primitives SysML pour l'allocation» présente les primitives utilisées dans cette étape. Pour chaque allocation variable, la liste des éléments de destination est récupérée avec la primitive UML « *GetAllocatedTo()* ». Leur correspondance avec les éléments de l'algorithme « générique » décrit dans la section 2.6 est également indiquée.

Primitive UML/SysML	Primitives génériques	Description
$\boldsymbol{getAllocatedTo}()$	$GetTo()$	Renvoie la liste des éléments alloués à un bloc.
$\boldsymbol{isAllocatedApplied}()$	GetFrom()	Permet de savoir si un bloc dispose du stéréotype « Allocated »

Tableau 12 «Primitives SysML pour l'allocation»

3.4.Intégration des solveurs

Dans ce paragraphe nous proposons une formalisation du problème CSMOP sous forme de méta-modèle. Ce méta-modèle nous permet d'être indépendant du solveur utilisé pour résoudre le problème CSMOP. Nous décrivons ce méta-modèle en utilisant un langage dont la syntaxe est proche du langage C. La grammaire de ce langage adapté au problème CSMOP est décrite en utilisant le métalangage EBNF pour Extended Backus-Naur Form (Wirth, 1996) .

3.4.1. Problème CSMOP

La Figure 48 «méta-modèle d'un problème CSMOP» présente la structure du problème CSMOP. Le problème CSMOP se compose d'un ensemble de déclarations, chacune étant ordonnée par un attribut « order ». Parmi les déclarations qui le composent, on trouve les déclarations des variables et des constantes, la définition des contraintes, des fonctions objectifs ainsi que de la stratégie de résolution du problème.

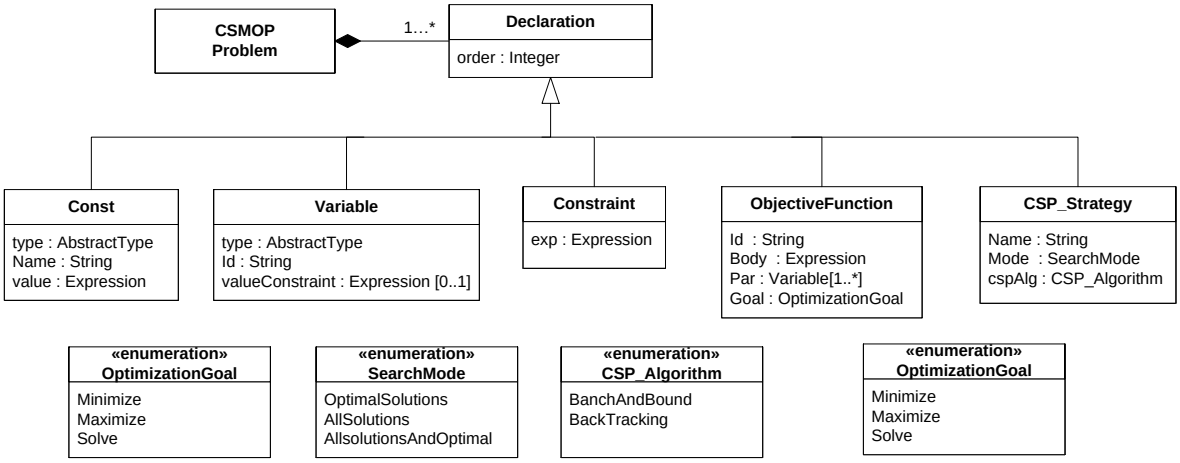


Figure 48 «méta-modèle d'un problème CSMOP»

Syntaxe EBNF

La description du problème CSMOP dispose de la syntaxe suivante :

CSMOP ::= (def-type|decl-var|decl-Constraint|def-ObjFunc|def-CSPStrat)*

Un identifiant « id » est utilisé pour nommer une variable ou une fonction objectif.

id ::= (letter)*

letter ::= a|…|z|A..|Z|

Une constante peut être utilisée pour initialiser une variable ou intervenir dans une expression :

cst ::= cst-int |cst-real

cst-int ::= [-] $(digit)^+$

cst-real ::= [-] $(digit)^+$. $(digit)^+$[cst-int]

digit ::= 0|1|…|9

3.4.2. Variables et Types

Les variables du problème CSMOP disposent d'un type abstrait « AbstractType » présenté dans la Figure 49 «type de données pour le problème CSMOP». Ce type abstrait peut être concrétisé par un type scalaire « ScalarType » ou par un type défini « DefinedType ». Pour les types scalaires, nous utilisons pour un problème CSMOP le type booléen, les types énumérés et les types numériques (entiers et réels) bornés. Pour les types définis, les matrices « MatrixType » ou les tableaux « ArrayType » sont utilisés.

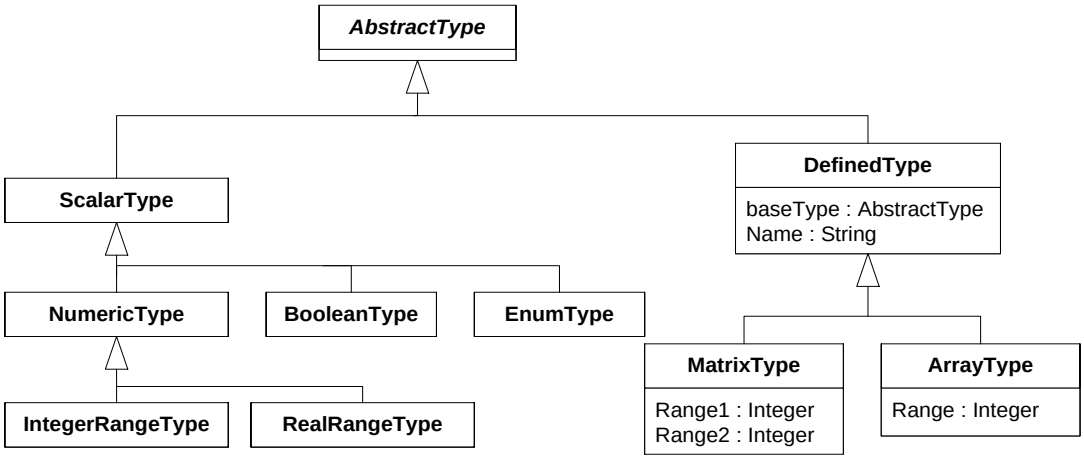


Figure 49 «type de données pour le problème CSMOP»

Syntaxe EBNF

La description d'un type est faite en utilisant la syntaxe suivante :

id-type ::= int | real | boolean |enum

il est possible de déclarer un tableau à une ou plusieurs dimensions avec le suffixe suivant :

arr-suff ::= ([[cst-entier]]) *

ainsi qu'un intervalle pour la variable avec le suffixe suivant :

range-suff ::= [WithRange [cst-entier , cst-entier]]

La syntaxe de déclaration d'une variable de décision est la suivante :

decl-var ::= decisVar id-type id arr-suff range-suff ;

exemple de déclarations de variables de décision :

decisVar int a WithRange [-2, 5] ; // une variable de décision de type entier de -2 à 5

decisVar boolean a[2][2] ; // une variable de décision de type matrice de booléens

3.4.3. Les expressions pour les contraintes et les fonctions objectifs

Une contrainte ou une fonction objectif intègrent une expression prenant en paramètre un ensemble de variables. Une expression peut être considérée comme une instruction exécutable représentant une valeur. Les expressions sont présentées Figure 50 «Les expressions du problème CSMOP». Une expression basique est une chaîne de caractères sans opérateur, les autres expressions étant une expression binaire « BinaryExpression », ou bien une expression applicable à un ensemble de variables « SetExpression » ou « AggregExpression ».

Les expressions non basiques font intervenir un opérateur de type relationnel « RelationOperator » ou applicable à un ensemble de variables « SetOperator » ou « AggregOperator ».

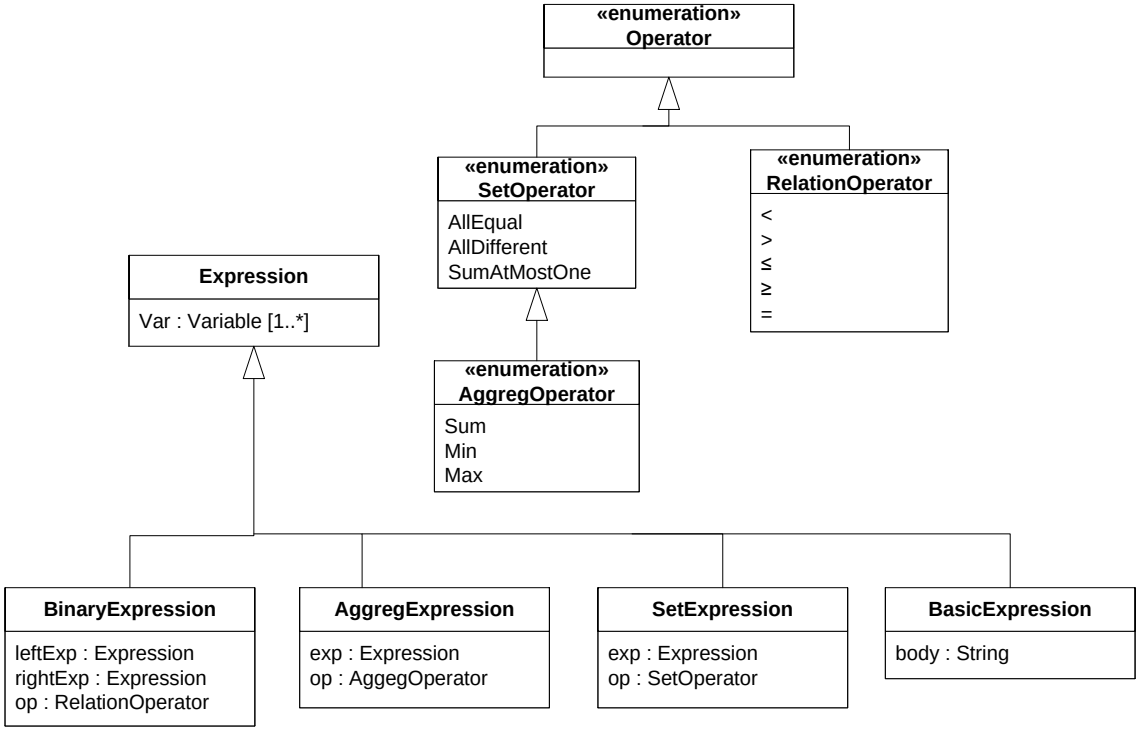


Figure 50 «Les expressions du problème CSMOP»

La description d'une expression est faite en utilisant la syntaxe suivante :

expr ::= cst |id| constr-expr| arith-expr

expr-list ::= expr (, expr)*

constr-expr ::= bin-expr | aggreg-expr | set-expr

arith-expr ::= expr arithm-op expr

bin-expr ::= arith-expr | rel-op | arith-expr

aggreg-expr ::= aggeg-op (arith-expr)

set-expr ::= set-op ([expr-list])

rel-op ::= <|>|≤|≥

arithm-op ::= +|-|*|/

set-op ::= AllEqual | AllDifferent | SumAtMostOne

aggeg-op ::= Sum | Min | Max

Exemples de déclaration d'expressions de contraintes impliquant des variables de décision :

int b[4] ;

int c[4] ;

b[1] + b[2] < b[1] + b[2]

Sum(b) <2

SumAtMostOne(b)

AllEqual (b)

La définition d'une fonction objectif et de la stratégie de résolution du problème est faite en utilisant la syntaxe suivante :

def-objFunc ::= optim-goal id-type id (args-list) { cst-string }

args-list ::= id-type id arr-suff (, id-type id arr-suff)*

cst-string ::= " (char)* "

optim-goal ::= [min]|[max]|[solve]

def-CSPStrat ::= ResStrat res-algo search-mode

res-algo ::= Branch&Bound | BackTrack

search-mode ::= optimal | allSolution | optimalWithAllSolution

3.4.4. Le modèle CSMOP et sa représentation par Labix/PyOpt/CHOCO

Dans le Tableau 13 : «Exemple d'instanciation d'expressions CSMOP» différentes expressions sont présentées. La première est une déclaration de variable de type matrice booléenne, utilisée pour résoudre un problème de choix de composant ou bien un problème d'allocation. La seconde expression est une contrainte d'unicité, exprimant par exemple dans le cas de l'allocation qu'un élément source est alloué à un seul élément de destination.

CSMOP (syntaxe EBNF)	Solveur Labix (Python)	Solveur CHOCO (JAVA)
Declaration de Variable Name = « a » Type = MatrixType baseType = Boolean Range1 = 2 Range2 = 2 DecisVar Boolean a[2][2];	problem.addVariables(["***a***"+str(i)+str(j) for j in range(**1,3**) for i in range(**1,3**)], [0,1])	BoolVar[][] **a** = VariableFactory.*boolMatrix*("s", ***2,2***, solver);
Constraint Name = "Unicity constraint" Var = a[1] Exp : BinaryExpression SetOperator= SumAtMostOne SumAtMostOne(a[1]);	problem.addConstraint(lambda a11,a12, : a11+a12 <= 1, (*"a11"*,*"a12"*))	SatFactory.*addAtMostOne*(a[1]);

ObjectiveFunction
Name = "TotalCost"
Goal = Minimize
Par = Boolean a[][], real compCost[], real TotalCost
BodySolveur1 = "…."
	for l in range(len(lisR)):
	X=1
	for j in range(len(lisR[l])):
		for(**int** i=0;i<***C_MAX***;j++) {
		for(**int** j=0;i<***S_MAX***;j++) {
		TotalCost= TotalCost + a[i][j]*Compcost[i];
		}
		}

Tableau 13 : «Exemple d'instanciation d'expressions CSMOP»

3.5.Conclusion

Dans ce chapitre nous avons présenté l'intégration dans le langage SysML des concepts MBSE développés dans le chapitre 2. Les points de décisions sont définis avec des stéréotypes, par extension des classes de base UML (méta-classe) ou par spécialisation d'un stéréotype proposé. Une nouvelle utilisation des diagrammes BDD et paramétrique de SysML est utilisée pour le contexte d'optimisation. Les algorithmes de transformation proposés dans le paragraphe 2.6 ont été adaptés au métamodèle UML/SysML et permettent d'analyser les points de décisions et les contraintes. Le résultat de cette transformation est une définition d'un problème d'optimisation CSMOP, pouvant être résolu par un solveur tel que ceux présentés dans la section 1.2. Enfin, pour permettre la mise en œuvre des différents solveurs, nous avons proposé une description du problème CSMOP au format EBNF, ce qui permet de rester indépendant d'un solveur particulier. Les stéréotypes et les algorithmes proposés dans ce chapitre peuvent être mise en œuvre dans un outillage SysML. Nous proposons de le faire dans le chapitre suivant avec l'outil Papyrus ainsi qu'avec trois cas d'étude. Ces cas d'étude correspondent aux trois types de points de décisions que nous avons définis dans le chapitre 2.

4. Méthodologie, Outillage et application

Dans ce chapitre, nous proposons dans une première section une méthodologie permettant d'intégrer l'optimisation dans les méthodes MBSE. Nous prenons ensuite pour exemple l'intégration dans la méthode OOSEM. Un outillage associé à la méthodologie a été développé, sous forme d'un plug-in Eclipse intégré dans l'outil Papyrus avec le profil SysML. Dans une seconde section, trois cas d'études sont présentés. Ces cas d'étude correspondent aux trois types de points de décisions que nous avons définis. Le premier concerne un choix de composant ainsi qu'un niveau redondance pour une caméra embarquée dans un véhicule. Il s'agit d'optimiser le coût et la fiabilité du système, en utilisant le premier type de points de décisions que nous avons définis. Le second cas d'étude intègre des variables continues dans le problème d'optimisation et le troisième cas d'étude consiste à résoudre un problème d'allocation optimale pour un système multi-cœurs.

4.1. Méthodologie et outillage de la méthode d'optimisation

4.1.1. Méthodologie proposée

La Figure 51 « Méthodologie proposée pour l'optimisation » présente les différentes étapes permettant d'obtenir un modèle optimisé du système. Une première étape consiste à enrichir un modèle SysML initial avec un contexte d'optimisation « MDO Context ». Ce contexte a été défini au paragraphe 2.5. Le modèle initial décrit le système sans variation ni degré de liberté.

D'autre profils comme MARTE peuvent être également être utilisés dans le modèle initial lorsqu'il s'agit de modéliser des aspects plus spécifiques comme une tâche pour un logiciel embarqué. Dans ce cas, le profil MARTE permet de modéliser les temps d'exécution et les périodicités dont nous avons besoin pour optimiser les performances du système. Le contexte d'optimisation représente une situation dans laquelle le système est placé, et pour laquelle un ensemble de fonctions objectifs doivent être maximisées ou minimisées. Les points de décisions définis au paragraphe 2.2 sont intégrés dans le contexte d'optimisation. En fonction du besoin, l'un des trois types de point de décision peut être appliqué (choix de composant et redondance, configuration ou allocation).

Dans une seconde étape, le modèle et son « *MDO Context »* sont transformés en un problème *CSMOP* défini dans la section 3.4, pouvant être résolu par un solveur. Le choix du solveur est à l'initiative de l'utilisateur, en fonction du type de problème à résoudre (problème combinatoire ou continu) et de la nature des points de décision. Même s'il est possible d'intégrer d'autre solveurs, nous avons pour le moment utilisés trois d'entre eux : Labix, Choco et PyOpt (1.2)

Dans une troisième étape, les solutions optimales sont générées par le solveur et sont soumises au choix du décideur. Lorsque ce choix est effectué, le modèle initial peut être mis à jour, et d'autre optimisations plus fines peuvent être conduites.

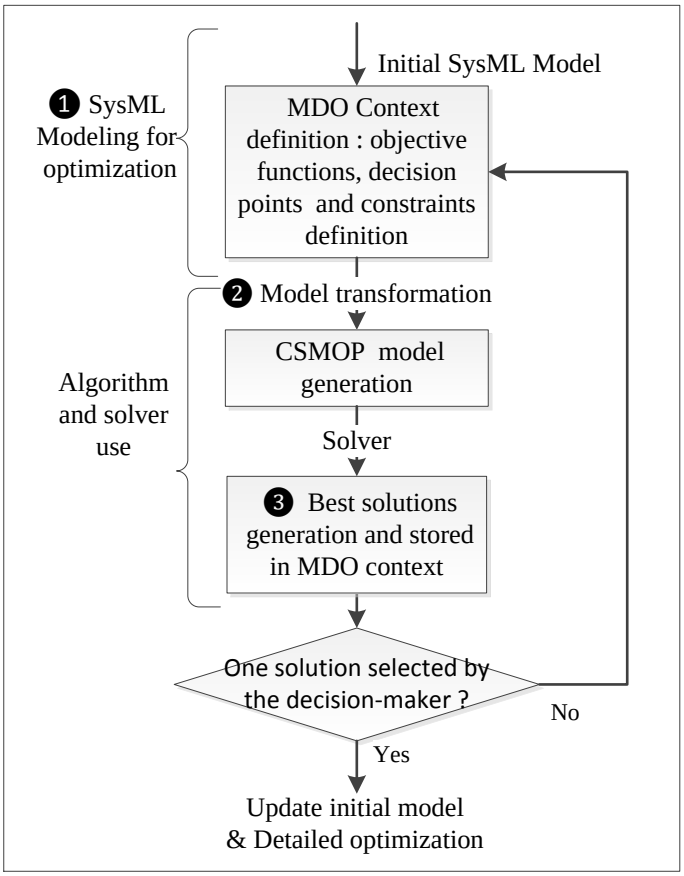


Figure 51 « Méthodologie proposée pour l'optimisation »

4.1.2. Intégration dans OOSEM

La méthodologie que nous venons de proposer peut s'adapter aux différentes méthodes d'ingénierie système MBSE que nous avons présenté dans la section 1.1. La Figure 52 « Méthode OOSEM modifiée » montre les activités impactées par notre approche : l'activité d'analyse des besoins des parties prenantes ainsi que l'activité d'évaluation des variantes. Cette dernière est réalisée en parallèle d'autres activités, et en particulier de la synthèse des architectures répondantes aux contraintes et aux objectifs. C'est lors de cette synthèse que les optimisations que nous proposons vont s'avérer utiles pour le concepteur.

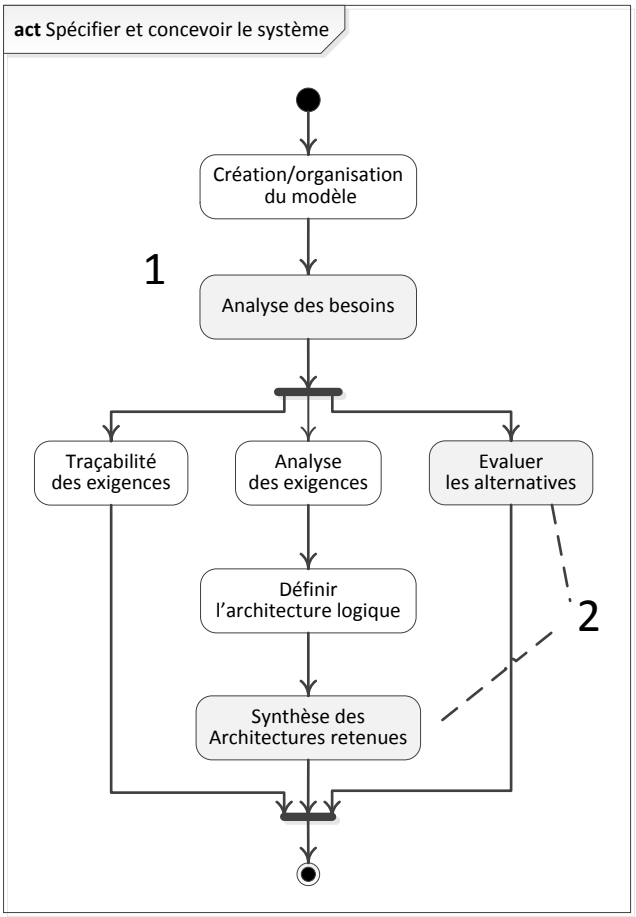


Figure 52 « Méthode OOSEM modifiée »

Dans l'analyse des besoins présentée dans la Figure 53 « Analyse des besoins» les fonctions objectifs à optimiser sont ajoutées au modèle, sous forme de diagrammes paramétriques. Les objectifs à atteindre pour ces fonctions sont également définis, sous forme de contraintes dans un diagramme d'exigences. Une première version du contexte d'optimisation « MDO context » est créé, incluant les fonctions objectifs et leurs valeurs respectives à atteindre.

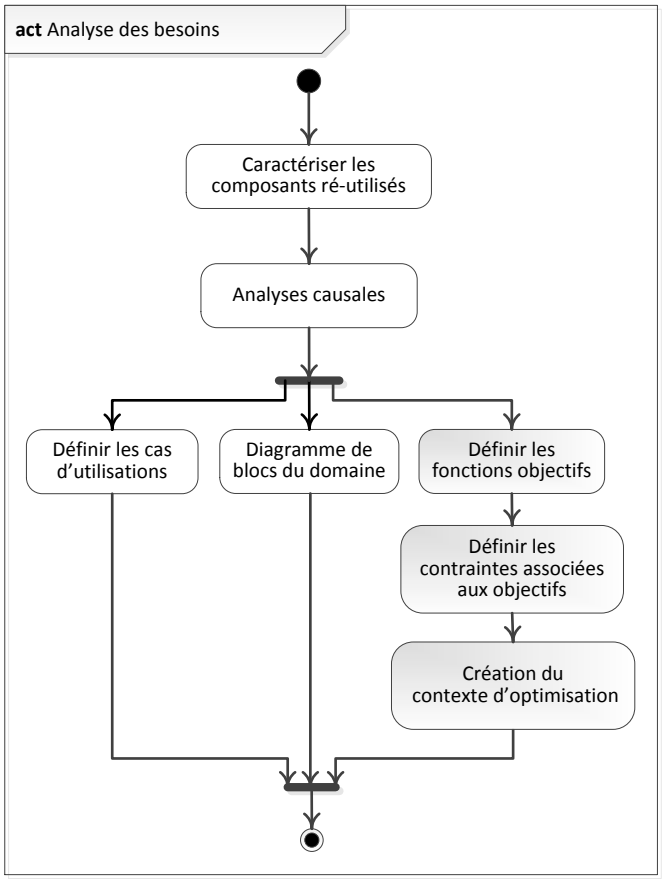


Figure 53 « Analyse des besoins»

Puis l'étape qui consiste à trouver et à évaluer les différentes solutions se déroule dans l'activité de synthèse des architectures, comme présenté dans la Figure 54 « Synthèse des architectures modifiée» le la méthode OOSEM. Les activités en gris complètent les activités de la méthode. Dans les premières étapes de définition (partitionnement, architecture logique et physique), le contexte d'optimisation « MDO context » est enrichi avec des points de décisions pour la redondance (structure) et le choix d'instances. Des contraintes globales sont associées au partitionnement. Dans une deuxième étape, ce premier problème d'optimisation CSMOP est résolu, et la solution optimale choisie par le concepteur est utilisée pour définir l'architecture Hardware. Lorsqu'il s'agit de définir l'architecture logicielle « Software architecture » (troisième étape), des nouvelles variables de décisions propres à l'allocation sont rajoutées au contexte d'optimisation. On obtient alors un second problème CSMOP à résoudre. La meilleure configuration choisie est alors utilisée pour définir l'architecture logicielle, sous la forme d'une table d'allocation.

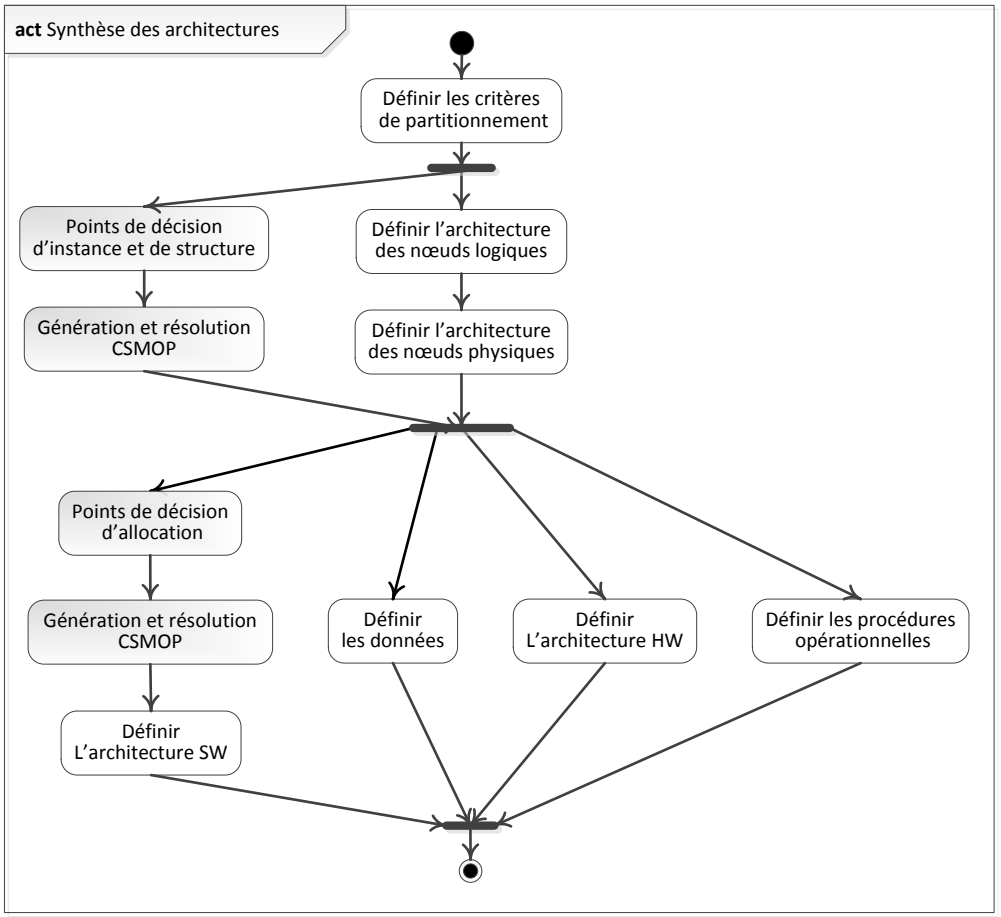


Figure 54 « Synthèse des architectures modifiée»

4.1.3. Intégration dans l'outil Papyrus

La méthode proposée a été implémentée dans l'outil Papyrus et avec la plateforme Eclipse « open source » (Eclipse, 2016) permettant d'intégrer les différents solveurs que nous avons utilisés. Papyrus (Papyrus, 2015) est un outil de modélisation UML2 multiplateformes développé par le CEA, sous licence logicielle libre. Papyrus supporte également les langages de modélisation dérivés de UML, comme MARTE et SysML. Papyrus permet à l'utilisateur de définir un éditeur de son propre DSL (Domain Specific Language) sous forme d'un profil contenant un ensemble de stéréotypes. L'outil graphique Papyrus inclut notamment un éditeur pour les différents diagrammes, une vue hiérarchique permettant de parcourir le modèle ainsi qu'une vue permettant d'éditer les propriétés d'un élément du modèle comme présenté dans la Figure 55 « Outil Papyrus ».

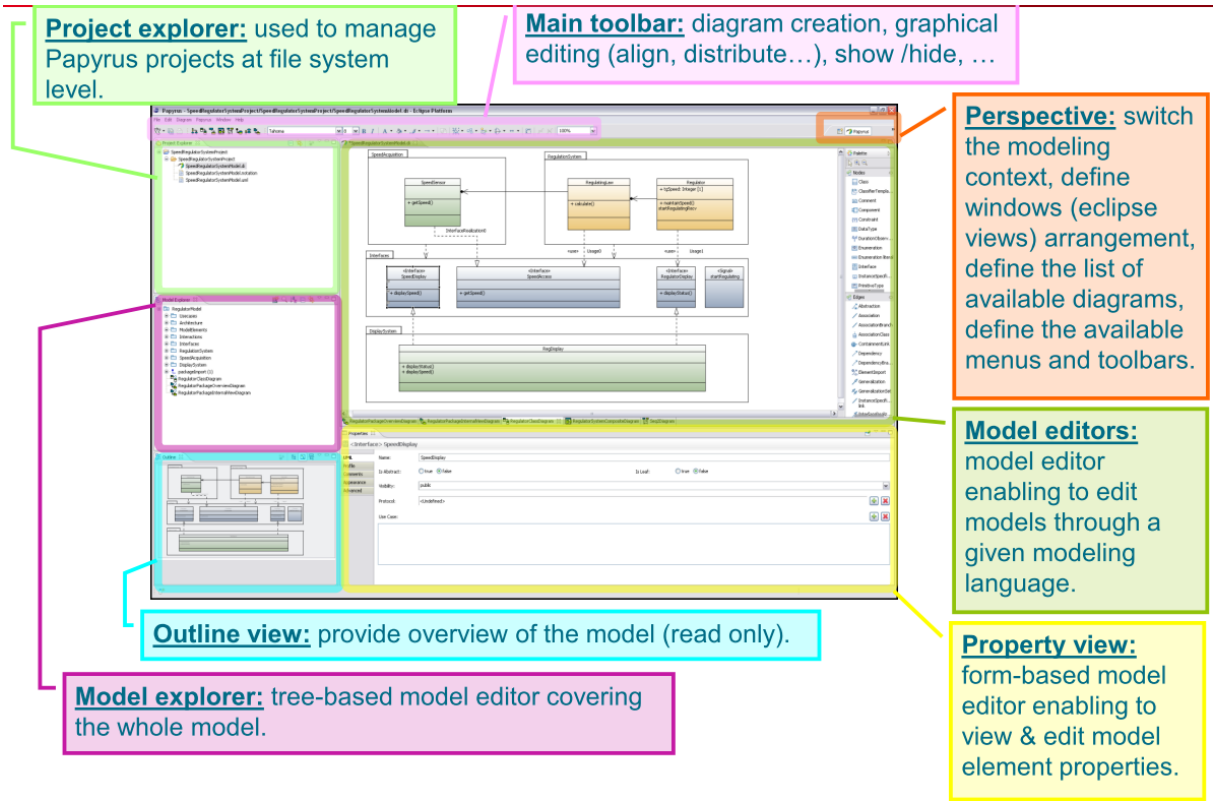


Figure 55 « Outil Papyrus » de (Papyrus, 2015)

Pour réaliser le démonstrateur, nous avons utilisés plusieurs composants présentés dans la Figure 56 « Briques logicielles pour le demonstrateur ». Le démonstrateur se compose des éléments suivants :

Un plug-in permettant d'optimiser un modèle SysML, selon la méthode que nous proposons. Ce plug-in a été développé avec les langages Java pour l'analyse du modèle et XTEND (Bettini, 2013) pour la génération de code interprétable par les solveurs. Le langage XTEND permet de générer le code du solveur à partir d'expressions « template » incluant des paramètres provenant du modèle analysé :

```
'''
«IF VarKind==Allocation»
# // Schedulable Resource number
private static final int S_MAX = «ResourceList.getNum()»
private static final int C_MAX = «CoreList.getNum()»
Solver solver = new  Solver("Allocation problem"); '''
«ENDIF»
```

Pour ce plug-in, nous utilisons d'abord l'infrastructure de génération de code intégrée à l'EMF (Eclipse Modeling Framework) pour générer le code du solveur. Le code du plug-in est intégré dans un projet Eclipse ce qui permet à l'utilisateur de rester dans le même environnement de travail, depuis la modélisation jusqu'à l'optimisation. Le Plug-in utilise également le framework Eclipse RCP (Rich Client Platform), comme une application cliente classique,

pour afficher les résultats sous forme de tableaux ou de graphiques comme le front de Pareto. Un modèle intermédiaire entre le plug-in et le solveur permet de réduire le travail d'intégration d'un nouveau solveur. En particulier le plug-in est capable de générer un code Java pour le solveur Choco mais également Python pour les solveurs Labix et PyOpt. Dans ce dernier cas, le code Python est exécuté dans l'environnement Eclipse et avec un projet associé, en utilisant le module PyDev (Liu, 2011).

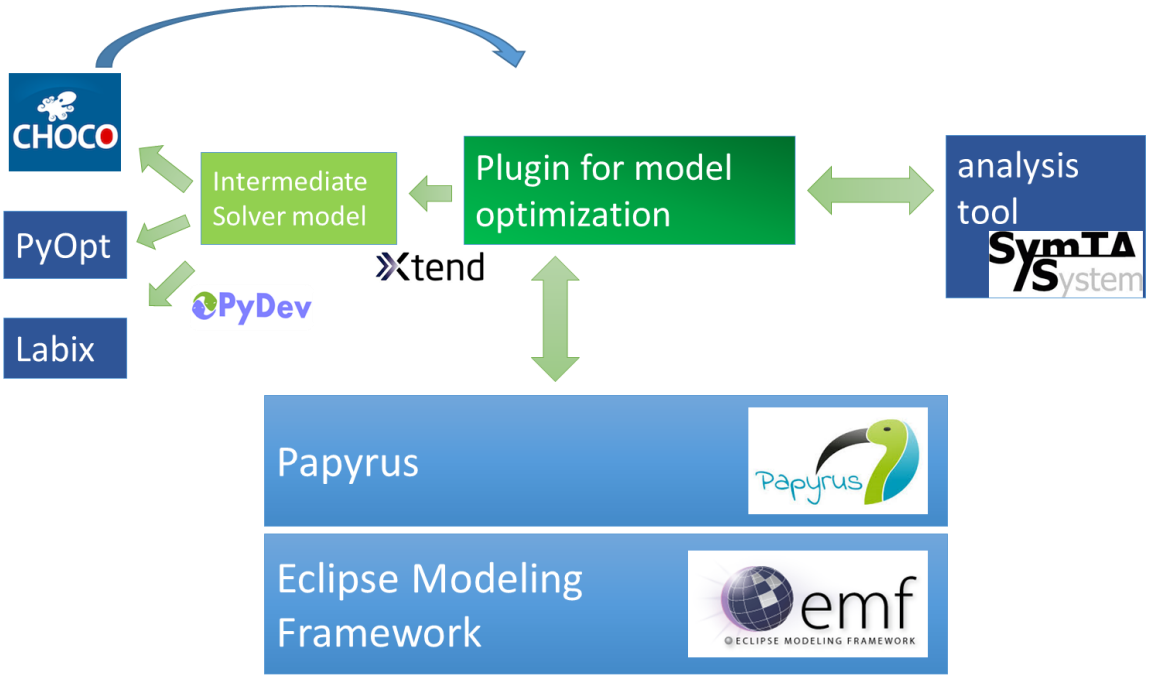


Figure 56 « Briques logicielles pour le demonstrateur »

Le démonstrateur intègre également une interface de type client-serveur avec l'outil d'analyse SymTA/S (Henia, 2005). Cette fonctionnalité est utilisée pour la résolution d'un problème d'allocation de tâches sur des processeurs, afin de déterminer quelles sont les solutions ordonnançables parmi les solutions optimales calculées par le solveur. La partie cliente du démonstrateur permet un accès à distance au serveur SymTA/S et va réaliser les actions suivantes :

- Ouverture de la connexion client/serveur SymTA/S
- La création d'une configuration initiale SymTA/S à partir du modèle SysML/MARTE, avec les tâches, les processeurs, les cœurs et les allocations correspondant aux solutions optimales calculées par le solveur.
- Pour chaque solution, l'analyse du système est demandée au serveur.
- Sauvegarde des résultats d'analyse incluant la charge de chaque cœur pour affichage tabulaire
- Fermeture de la liaison client/serveur SymTA/S

4.1.4. Mise en œuvre du démonstrateur

La mise en œuvre du démonstrateur est présentée dans la Figure 57 « Génération des solutions optimales avec Papyrus ». Dans la première étape, l'utilisateur complète un modèle initial avec les différents points de variabilités pour l'optimisation, ainsi que les fonctions objectifs. Le choix d'un solveur est également effectué. Puis l'utilisateur crée un contexte d'utilisation, dont le point d'entrée est un diagramme BDD contenu dans un package. En sélectionnant ce package et avec un clic droit, l'utilisateur affiche un menu de génération de code, dans lequel plusieurs options lui sont proposées. La première option, présentée dans la figure, permet de générer les solutions optimales en prenant en compte les choix de composant et de redondance. La seconde option va prendre en compte les choix de configuration (variation sur les attributs) et la troisième option les choix d'allocation, avec une analyse possible SymTA/S dans le cas d'un problème d'ordonnancement. Une fois l'option choisie, un nouveau projet est créé automatiquement (deuxième étape de la figure), dont le type est fonction du solveur choisi, comme par exemple un projet de type « Python ». L'utilisateur peut alors « exécuter » ce projet et afficher les résultats sous forme d'un diagramme de Pareto permettant d'afficher le niveau d'atteinte des objectifs par les différentes solutions. Ce diagramme est complété par une table permettant de connaitre les valeurs de chacune des variables de décision. C'est à partir de cette table qu'il sera possible (dans une prochaine version du démonstrateur) de générer le modèle correspondant, avec des valeurs figées pour les différents points de décision.

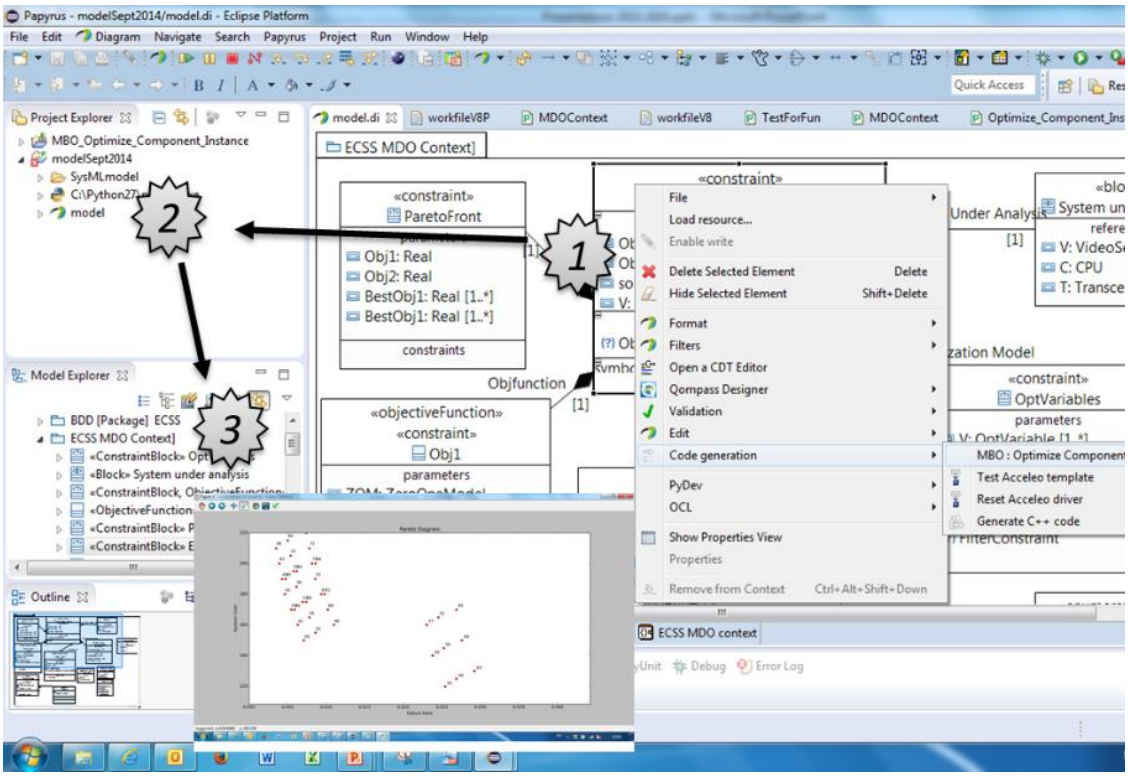


Figure 57 « Génération des solutions optimales avec Papyrus »

4.2.Les cas d'étude

4.2.1. Choix de composants et niveaux de redondance pour une caméra stéréoscopique (ADAS)

Ce premier cas d'étude a été présenté dans (Leserf, Déc. 2015) et (Leserf, Fev. 2015). Dans le secteur des transports en général et de l'automobile en particulier, les ADAS ou « Systèmes Avancés d'Aide à la Conduite » apportent de nouvelles réponses aux problématiques de sécurité depuis plusieurs années. Les ADAS regroupent l'ensemble des technologies permettant :

1. D'éviter l'apparition d'une situation dangereuse pouvant aboutir à un accident. On parle alors de système de sécurité active, chargé de prévenir l'accident ;
2. De libérer le conducteur de certaines tâches qui atténuent sa vigilance ;
3. D'assister le conducteur dans sa perception de l'environnement.

Dans l'automobile, parmi les accidents que l'on souhaite éviter, ceux impliquant des piétons sont malheureusement en augmentation malgré une tendance générale à la baisse comme le montre la Figure 58 « Nombre d'accidents impliquants des piétons en Europe » issue de (Eckert, 2013).

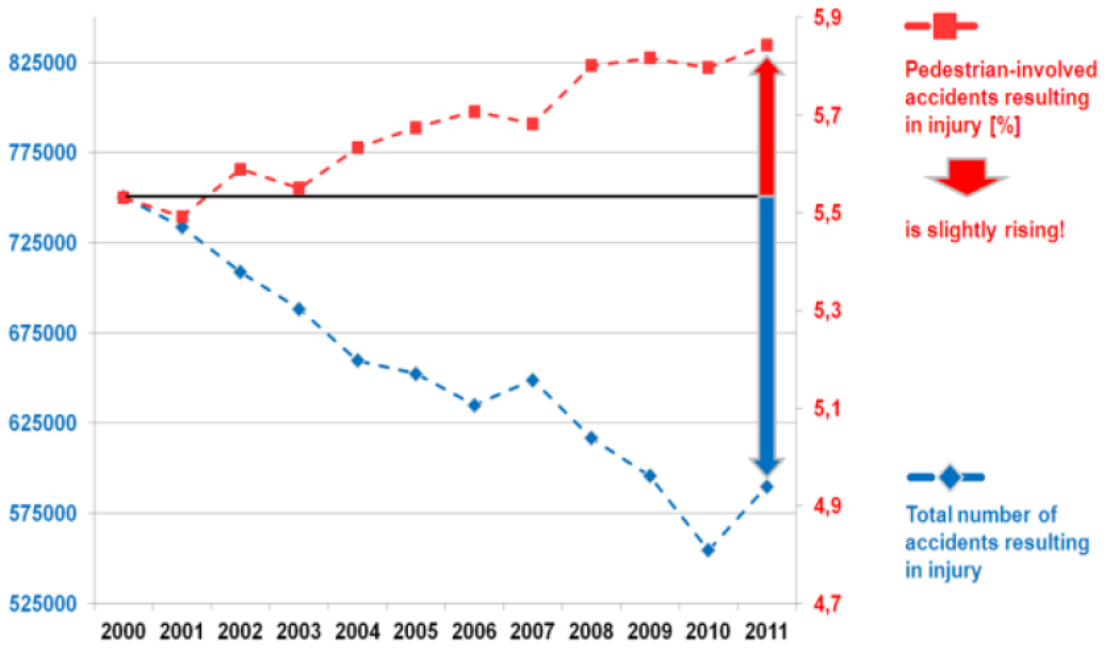


Figure 58 « Nombre d'accidents impliquants des piétons en Europe »

Les ADAS sont une réponse à cette problématique de sécurité active, notamment avec les systèmes optiques composés d'une ou de plusieurs caméras, associées à un traitement d'image s'exécutant sur une plateforme embarquée dans le véhicule. On désignera un tel système par ECSS pour « Embedded Cognitive Safety System ». Les fonctions typiques d'un système ECSS sont la détection de piétons ou d'obstacle (Eckert, 2013), mais également le

franchissement de ligne sur la chaussée par le véhicule. La Figure 59 « Système ECSS » présente la structure d'un tel système. La détection d'obstacle et en particulier la mesure de la distance séparant le véhicule de l'obstacle est rendue possible grâce à l'utilisation de deux caméras, en vision stéréoscopique. Les caméras sont un choix plus économique par rapport à l'utilisation de radars ou lidars. Pour le concepteur du système ECCS, **le coût et la fiabilité** sont les deux paramètres principaux à optimiser lors de la conception du système.

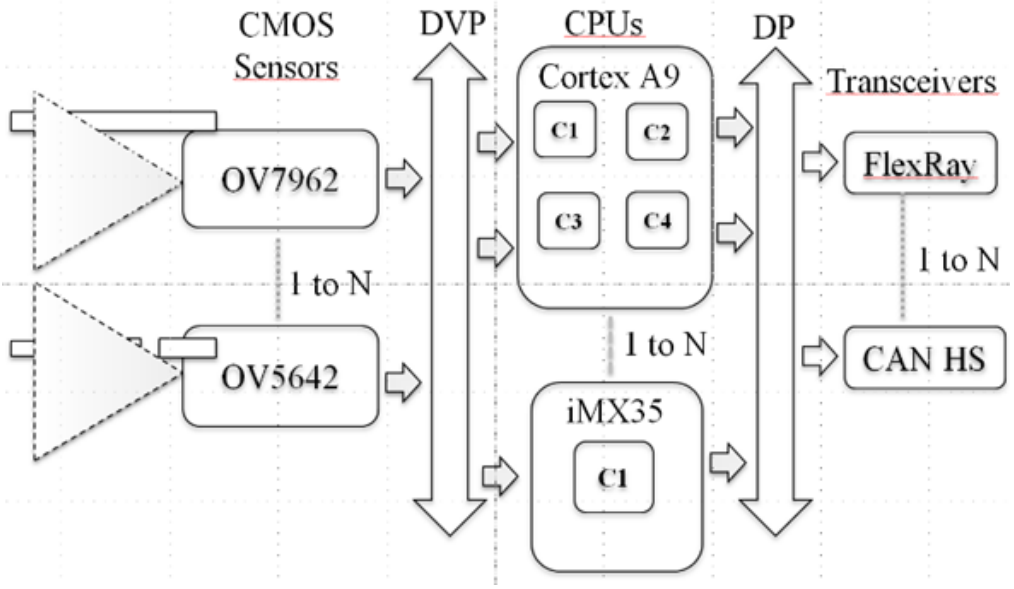


Figure 59 « Système ECSS »

Le système se compose de capteurs CMOS « CMOS Sensors » disposant de fonction de stabilisation d'image et de mise au point automatique. Ces capteurs sont connectés à des processeurs « CPUs » par l'intermédiaire d'un bus numérique vidéo DVP (Khan, 2011) disposant de vingt signaux parallèles. Les processeurs comme le Cortex A9 ou l'iMX35 disposent de fonctions de compression et de traitement d'image. Le traitement d'image va permettre de détecter un obstacle et d'envoyer une information au véhicule permettant une prise de décision automatique. Cette information est envoyée par le CPU par l'intermédiaires de « Transceivers » compatibles avec les réseaux embarqués présents dans un véhicule, à savoir le bus CAN à haute vitesse CAN HS (Paret, 2007) ou FlexRay (Paret, 2012). Les Transceivers sont spécifique à un réseau embarqué, mais il est possible de les relier à un CPU non spécifique en utilisant un port numérique parallèle « DP » disposant d'une dizaine de signaux. Un port de type « DP » est spécifique à une connexion CPU/Transceiver tandis que le port « DVP » est spécifique à la connexion capteur CMOS/CPU. Bien que l'on trouve d'autres réseaux embarqués dans l'automobile, seuls le CAN HS et le FlexRay dispose des débits bruts suffisants pour ce type d'application, à savoir 1 Mbps pour le CAN HS et 10 Mbps pour le FlexRay. La protection face aux perturbations électromagnétiques est également un avantage de ces réseaux par rapport au réseau Ethernet notamment.

Modélisation des exigences SysML

La modélisation SysML du système ECSS intègre la représentation des exigences. La Figure 60 « Diagramme d'exigences du système ECSS » représente les exigences liées aux coût et à la fiabilité que l'on souhaite optimiser pour ce système. Une exigence se compose d'un identifiant ainsi que d'un texte permettant de la décrire. Dans notre cas une exigence indique que le coût complet du système ne devra pas dépasser 200€. Cette exigence est évaluée (Relation « Evaluate ») à l'aide d'une fonction objectif « HardwareCostEvaluation ». Les relations « Satisfy » et « Verify » permettent de relier une exigence aux éléments d'architecture du système ainsi qu'à des éléments liés au test du système, comme des cas de test. L'autre exigence concerne la fiabilité à atteindre pour le système. Dans notre contexte d'optimisation, des exigences spécifiques liées à l'architecture du système sont dérivées des exigences standard. Il s'agit du stéréotype « ArRequirement » utilisé pour spécifier que la redondance maximum d'un capteur doit être de deux, car on ne souhaite pas multiplier le nombre de capteurs pour des raisons de coût.

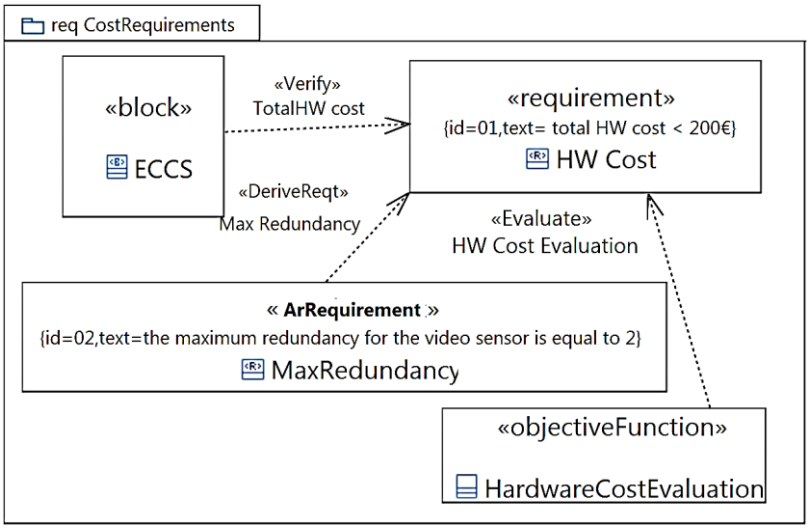


Figure 60 « Diagramme d'exigences du système ECSS »

Contexte d'optimisation et points de décisions

La modélisation du contexte d'optimisation de ce système a été présentée dans la sous-section 3.2 au paragraphe « Exemple de contexte d'optimisation SysML ». Le contexte d'optimisation du système ECSS se compose de deux fonctions objectifs, l'une permettant d'évaluer le coût du système et l'autre la fiabilité du système. Ce contexte d'optimisation est représenté par un diagramme paramétrique ainsi qu'un *BDD* associé.

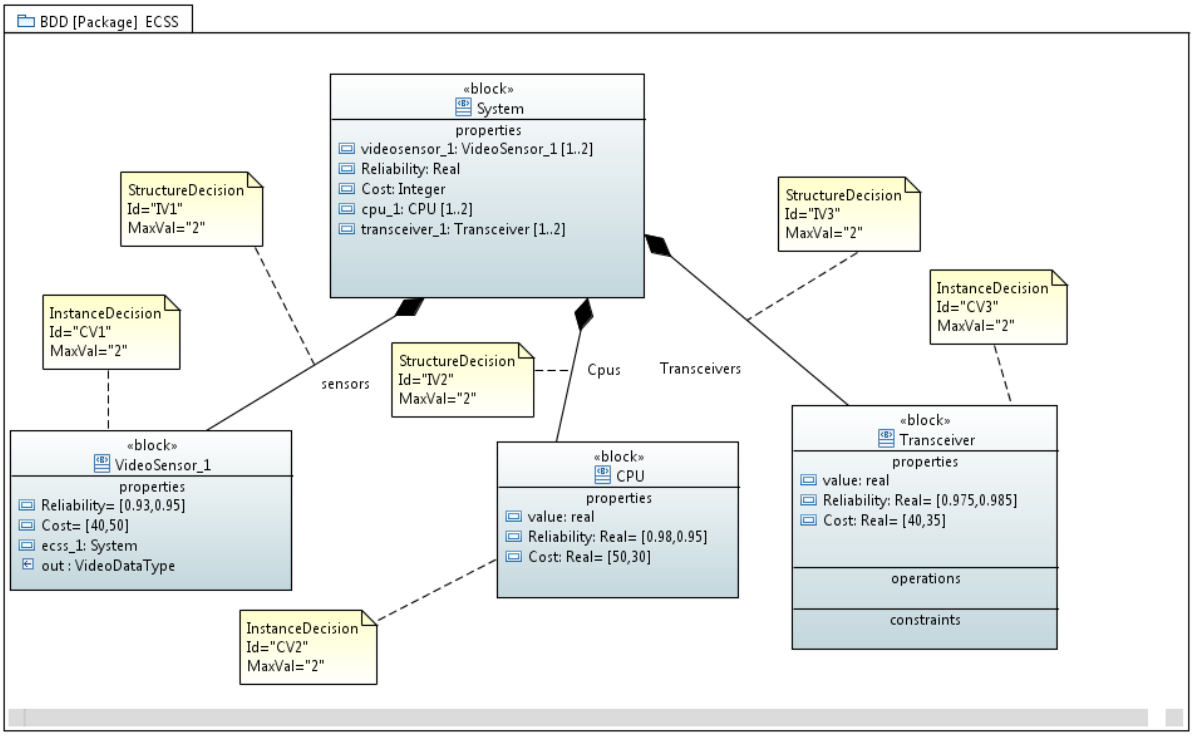


Figure 61 «Système enrichi avec les points de décisions»

La modélisation des points de décisions du système ECSS est présentée dans la Figure 61 «Système enrichi avec les points de décisions». Chacun des blocs capteurs, CPU, Transceivers est soumis à un point de décision de type « StructureDecision » pour modéliser la redondance variable ainsi qu'à un point de décision « InstanceDecision » pour modéliser le choix de composant parmi une liste de composants sur étagère. Selon la présence de l'un ou l'autre de ces points de décision, nous sommes en présence de décision de structure, d'instance ou combiné comme nous l'avons décrit en 2.3.1.

La fiabilité et le coût de ces composants sont intégrés dans le modèle, en associant une liste de valeurs aux attributs. Ces valeurs sont présentées Figure 62 « Caractéristiques des Capteurs/CPUs/Transceivers ».

Component	Reliability min-max	Cost min-max (€)
Sens. 1-3	0.99997-0.99998	16.9-20.2
Sens. 4-6	0.999976-.999985	21.5-25.7
CPU 1-3	0.99996-0.99998	12.6-21.2
CPU 4-6	0.99997-0.999985	28.4-34.5
Trans. 1-3	0.9934-0.9969	12.8-13.1
Trans. 4-6	0.9971-0.9995	13.8-15.4

Figure 62 « Caractéristiques des Capteurs/CPUs/Transceivers »

Création des variables de décision

A partir du modèle précédent, nous pouvons créer le problème CSMOP associé au système ECSS. Pour chaque bloc (Capteur/CPU/Transceiver) l'algorithme présenté dans la section 2.6

va créer un ensemble de variables de décision. Lorsqu'un point de décision combiné (redondance et choix de composant) est associé à ce bloc, cet ensemble de variables est constitué d'un tableau de booléen. La déclaration de cet ensemble, associée au bloc capteurs et pour le solveur Labix (Niemeyer, 2016) , est donnée par :

```
problem.addVariables(["a"+str(i)+str(j) for j in range(1,3) for i in
range(1,3)], [0,1])
```

Les contraintes associées aux variables de décision

Les contraintes de structure, données par l'Équation 4 « Contrainte d'instanciation » de la page 66 , permettent d'instancier un seul composant dans la solution. Elles sont données par la séquence de code suivante pour un bloc donné :

```
problem.addConstraint(lambda a11,a12, : a11+a12 <= 1, ("a11","a12"))
problem.addConstraint(lambda a21,a22, : a21+a22 <= 1, ("a21","a22"))
```

Les contraintes de connexion entre les blocs présentées dans la sous-section 2.4 concernent les liaisons capteurs-CPU et CPU-Tranceiver. Ces contraintes sont insérées dans un digramme de bloc interne IBD du modèle de l'ECSS. Un capteur doit être connecté à un CPU disposant d'un ou de deux ports d'entrée de type DVP. Il s'agit d'une contrainte de type « embranchement » dont le code associé est le suivant pour le solveur Labix :

```
problem.addConstraint(lambda a11,a12,a21,a22,b11,b12,b21,b22 :
2*b11+b12+2*b21+b22 >= a11+a12+a21+a22,
["a11","a12","a21","a22","b11","b12","b21","b22"])
```

Les fonctions objectifs

Les deux fonctions objectifs de ce cas d'étude sont le coût C (à minimiser) et la fiabilité R (à maximiser). En pratique c'est le taux de défaut $(1 - R)$ que l'on cherchera à minimiser. Ces deux fonctions peuvent s'exprimer en fonction des caractéristiques des composants du système ainsi que des variables de décision, par les équations suivantes. L'indice i correspondant au niveau de redondance d'un bloc B_k, et l'indice j correspond au choix d'instance pour ce même bloc B_k. Le coefficient θ_i représente un coût d'interconnexion entre deux blocs, c_{jk} et r_{jk} le coût et la fiabilité de l'instance j du bloc B_k. Les variables de décision x_{ijk} correspondent au choix d'instance et de redondance pour le bloc B_k, défini par l' Équation 1: point de décision combinés de la page 63 .

$$\min C = \sum_{i,j,k} c_{jk} \left[x_{ijk} + exp\left(\theta_i \sum_{k} x_{ijk} \right) \right]$$

$$\max R = \prod_{k} \left[1 - \prod_{j,k} [1 - x_{ijk} r_{jk}] \right]$$

Les fonctions objectifs C et R sont insérées dans le modèle, dans les blocs de contrainte du contexte d'optimisation portant le stéréotype « ObjectiveFunction ». Le code de ces fonctions objectifs est écrit dans le langage Python (Van Rossum, 2011).

Configurations testées

Les caractéristiques des configurations testée sont présentées dans Tableau 14 « Configuration redondance/choix de composants »

Configuration	Config 1	Config 2	Config 3
Nb de blocs avec points de décision	3	3	3
Points de décisions combinés	3	3	3
Instances / bloc	4	6	6
Redondance Max	2	2	2
Variables de décision	24	36	36
Contraintes de structure	24	36	36
Contraintes de connexion	3	3	5

Tableau 14 « Configuration redondance/choix de composants »

Pour une configuration donnée, le problème CSP associé est résolu avec le solveur Labix. Le Tableau 15 « Solutions et temps de calculs obtenus » présente les résultats obtenus avec un PC de type Intel i5 3GHz disposant d'une mémoire RAM de 4 GB. La Figure 63 « Solutions optimales pour le coût et le taux de défauts » est obtenue avec l'outil MATLAB (Mathews, 1999) pour la première configuration. Les solutions optimales ou front de Pareto sont obtenue par un parcours des solutions, en retenant les solutions non-dominées selon le critère présenté au chapitre 1 page 31. Parmi ces solutions optimales, le concepteur retiendra les trois solutions du Tableau 16 « Solutions optimales choisies par le concepteur » en retenant celles dont le coût est inférieur à 90€ et un taux de défauts inférieur à 2.10^{-5}. La configuration 2 correspond à une limite d'utilisation pour le solveur Labix, avec trois variables de décisions combinées, pour une redondance double et six composants pas blocs. Ceci est vrai pour trois contraintes de connexion, mais si l'on rajoute des contraintes de ce type on peut résoudre plus rapidement le problème (config 3).

Configuration	Config 1	Config 2	Config 3
Solveur	Labix	Labix	Labix
Algorithme CSP	Back-Tracking	Back-Tracking	Back-Tracking
Nb de solutions du problème CSP	1824	13600	8859
Nb de solution optimales	10	70	40
Solutions retenues par le concepteur	3	3	3
Temps de calcul (min.)	0,3 min	36 min	11 min

Tableau 15 « Solutions et temps de calculs obtenus »

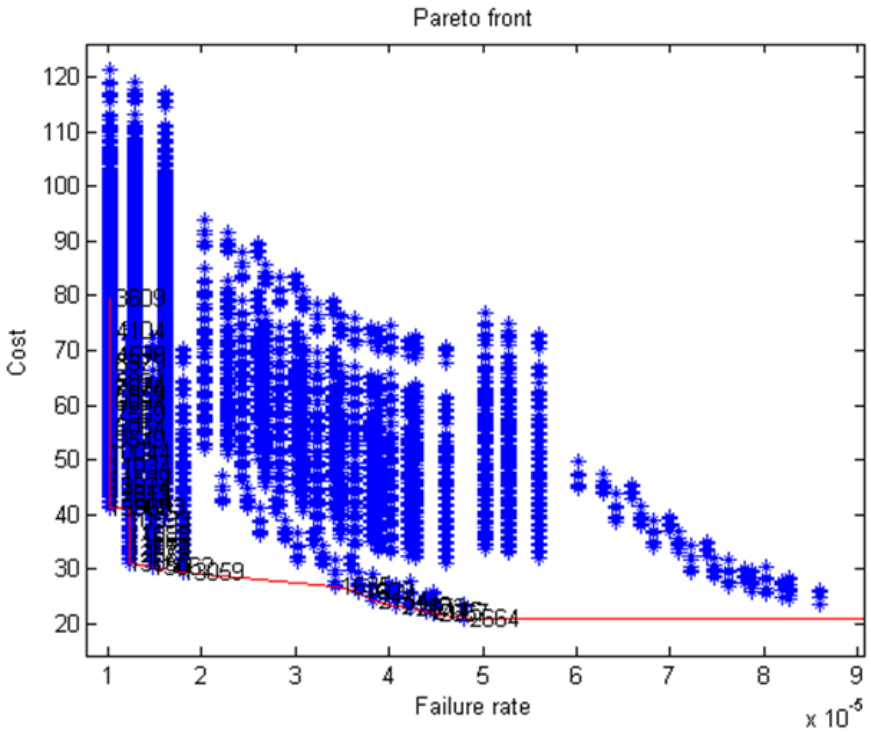


Figure 63 « Solutions optimales pour le coût et le taux de défauts »

Solution #	Sensors	CPUs	Transceivers	Cost (€)	FR (10-5)
1	S1+S1	CPU1	T4+T1	73	1.48
2	S1+S3	CPU1	T1+T1	75.3	1.22
3	S1+S3	CPU1+CPU1	T1+T1	87.9	1.02

Tableau 16 « Solutions optimales choisies par le concepteur »

4.2.2. Positionnement d'une caméra stéréoscopique dans un véhicule

Cette seconde application de la méthode proposée a été présenté dans (Leserf, Sept. 2015).

Dans les systèmes ADAS destinés à déclencher un freinage d'urgence en cas de détection d'un piéton étudiés dans (Eckert, 2013), le concepteur va également chercher à maximiser la performance du système, en plus du coût et de la fiabilité. Ce paramètre de performance dépend largement de la qualité des composants optiques utilisés, à savoir les lentilles et capteurs CMOS ainsi que leur association. Un autre paramètre important concerne le positionnement initial de la caméra dans le véhicule. Les performances globales du système traduisent sa capacité à détecter un obstacle le plus éloigné possible mais également un objet proche qui doit rester dans le champ de vision du capteur. Cette configuration optimale est obtenue en combinant des paramètres optiques (distance focale des lentille, position de la caméra) ainsi que le choix des capteur CMOS. La Figure 64 « Configuration géométrique de l'ECSS » présente la configuration du système stéréoscopique dans son environnement présentée dans (Charbonnier, 2008). Le principe d'optique stéréoscopique permettant de calculer la distance qui nous sépare d'un objet à partir de la comparaison d'image dans les deux vues des caméras se nomme la disparité. Les capteurs CMOS gauches et droits sont représentés par les deux plans inclinés S_l et S_r dénommés les vues. Le plan horizontal représente

la route et les deux plans verticaux les obstacles. Le ballon constitue l'obstacle proche et le piéton l'obstacle éloigné. Les capteurs et lentilles sont identiques entre les deux vues. La projection du point *P(X,Y,Z)* est à la même hauteur h dans les deux vues, sur une ligne horizontale l, et nous avons $v_r = v_l = v$ et nous avons :

$$v(Y,Z) = \frac{(v_0 \cos\theta + \alpha \sin\theta)Y + (v_0 \sin\theta - \alpha \cos\theta)(Z-h)}{Y\cos\theta + (Z-h)\sin\theta}$$

Équation 14 « équation de disparité »

Avec :

- v la position de la ligne l dans le plan S_l ou S_r, exprimé en pixel
- θ l'angle d'inclinaison de la caméra par rapport à la verticale,
- h la hauteur de la caméra,
- $v_0 = H_c/2$ est la projection du centre optique de la caméra, avec Hc la hauteur des capteur identiques en pixel,
- $\alpha = f/t_u$ où f est la distance focale des deux lentilles identiques et t_u la taille d'un pixel du capteur CMOS.

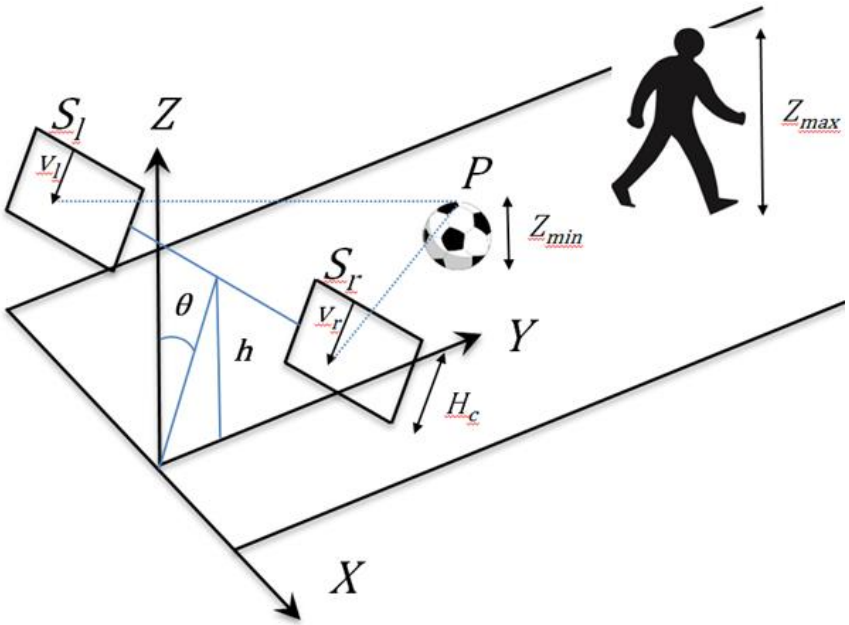


Figure 64 « Configuration géométrique de l'ECSS »

Dans une configuration optimale, le système ECSS doit détecter à la fois un obstacle distant et un obstacle proche pour un coût limité. L'image d'un obstacle distant sur les capteurs doit avoir une taille minimale (fixée à 50 pixels) pour permettre aux algorithmes de traitement d'image de détecter l'obstacle. L'image de l'obstacle proche doit pouvoir être contenue par les capteurs. Ce problème d'optimisation peut donc être formalisé ainsi :

- Minimiser le coût du système optique constitué de capteurs et lentilles

- Maximiser $v(Y,Z)$ pour l'obstacle proche.

Avec les contraintes :

- $v(obstacle\ distant) > 50\ pixels$,
- $v(obstacle\ proche) < H_c$

Et les variables:

- θ, h Nombres réels et bornés,
- Un choix de lentilles et de capteurs parmi une liste de composants aux caractéristiques connues

Il s'agit d'un problème d'optimisation mixte (variables discrètes et continues) et non-linéaire en raison de l'Équation 14 « équation de disparité »

Contexte d'optimisation et points de décisions avec SysML

La modélisation SysML des points de décisions de ce problème d'optique a été présentée dans la Figure 38 «Exemple de point de variation sur les attributs» de la page 82. Deux points de décision de type « RealValDecision » sont associés aux attributs réels représentant la position de la caméra et deux points de décision de type « InstanceDecision » sont associés aux blocs capteurs et lentilles.

La modélisation du contexte d'optimisation associée à ce problème d'optique est présentée dans la Figure 65 « Contexte d'optimisation paramètres optiques ». Le diagramme paramétrique comporte deux fonctions objectifs, l'une « HWCostEvaluation » permettant d'évaluer le coût du système et l'autre « NearObjectPicture » la taille de l'objet proche. Le bloc de contrainte « FarObjectPicture » représente la contrainte de taille de l'image pour l'objet éloigné. L'environnement est pris en compte avec la référence « Environment » sur les objets proches et éloignés. Les variables de décision sont intégrées dans le bloc « OptModel », il s'agit de « LensId » et « SensorId » pour le choix de composants (type entier) et « theta » et « hPosition », de type réel, pour le positionnement de la caméra. Une référence vers le bloc « StCam » permet aux deux fonctions objectifs d'accéder aux paramètres des composants optiques (distance focale, coût, résolution des capteurs).

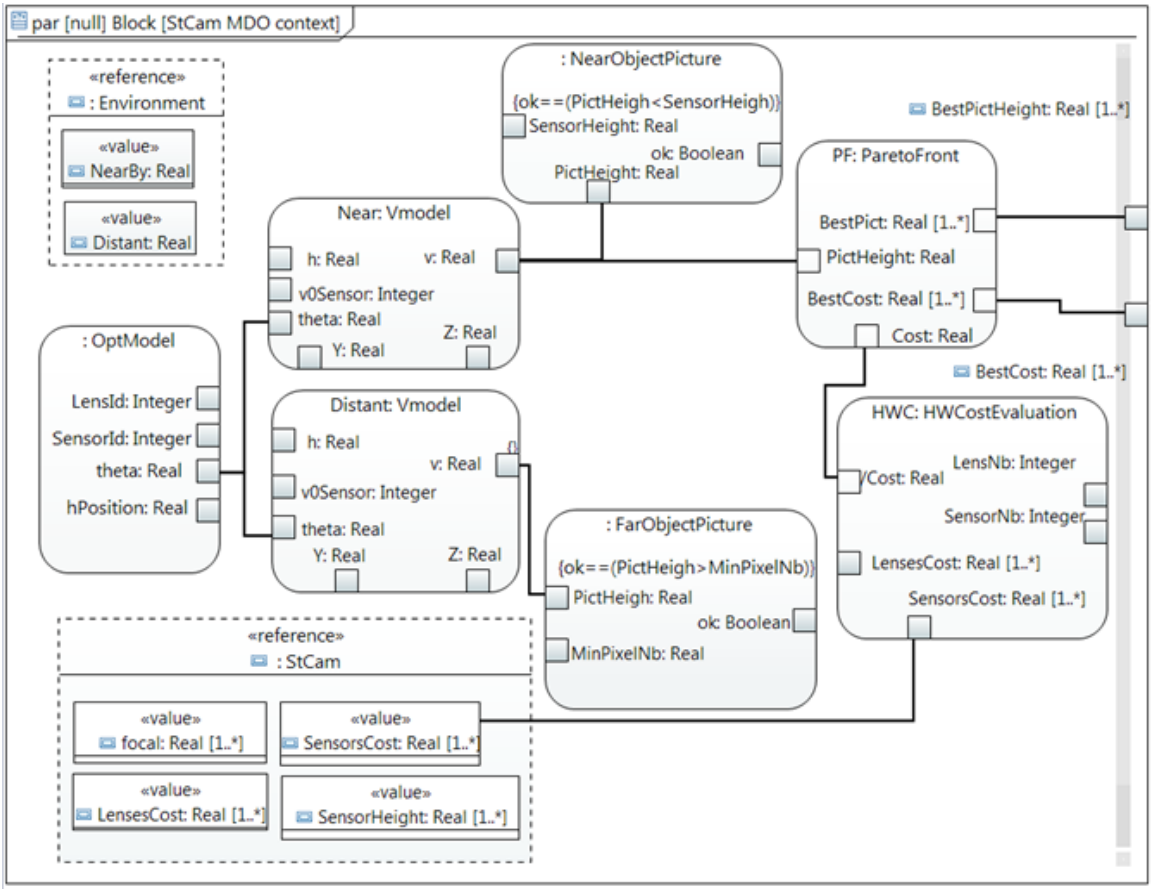


Figure 65 « Contexte d'optimisation paramètres optiques »

Génération et résolution du problème CSMOP

Pour générer le problème CSMOP associé à ce modèle SysML, nous utilisons l'algorithme proposé dans la sous-section 2.6 pour la transformation des attributs. Le choix du solveur est présent dans le modèle, ce choix est fait par le concepteur. Comme il s'agit d'un problème mixte avec des fonctions objectifs non-linéaires, nous avons choisi le solveur PyOpt (Perez, 2012), par rapport aux solveur CHOCO ou Labix qui ne peuvent résoudre cette classe de problèmes.

Nous obtenons les définitions suivantes associées aux points de décision du modèle :

```
opt_prob.addVar('x1','d',choices=[0,1,2])
opt_prob.addVar('x2','d',choices=[0,1,2,3])
opt_prob.addVar('x3','c',lower=0,upper=1)
opt_prob.addVar('x4','c',lower=0.2,upper=1)
```

Les variables discrètes x_1 and x_2 représentent les attributs *LensId* et *SensorId* du bloc caméra ECSS. Les variables continues x_3 and x_4 représentent les attributs ϑ and h.

Le Tableau 17 présente les solutions optimales obtenues avec le solveur PyOpt. Comme nous l'avons vu dans la section 1.2 , PyOpt est un Framework regroupant plusieurs algorithmes d'optimisation. Nous avons utilisé un algorithme génétique NSGA-2, le seul disponible dans

PyOpt pour résoudre un problème multi-objectif. Le paramètre v en pixel représente la taille de l'objet proche à maximiser. Le coût à minimiser est la somme des coûts des composants capteurs et lentilles. Les solutions 1,2,3 sont les plus performantes mais pour un coût élevé. A contrario les solutions 10 et 11, plus économiques, sont les moins performantes.

Variables de décision					*Objectifs*	
SolId	LensId	SensId	θ(rd)	h(m)	$\boldsymbol{v}$ (pix)	Cost(€)
1	3	1	0.40	0.2	2600	110
2	3	1	0.35	0.3	2600	110
3	3	1	0.32	0.4	2600	110
4	1	1	0.45	0.2	2450	85
5	1	1	0.38	0.2	2450	85
6	1	1	0.38	0.3	2350	85
7	1	1	0.40	0.3	2350	85
8	2	2	0.55	0.2	2300	80
9	2	2	0.55	0.2	2300	80
10	2	2	0.45	0.3	2100	80
11	2	2	0.45	0.3	2100	80

Tableau 17 « Solutions optimales obtenues avec les solveur PyOpt »

4.2.3. Allocation optimale d'applications sur un processeur pour un système robotique volant (drone autonome)

Les systèmes robotiques volants, ou drone autonomes, sont utilisés pour des missions de surveillances, de reconnaissance et de renseignement (How, 2009) et (Weatherington, 2005). Ces systèmes autonomes évoluent progressivement vers une autonomie complète, et des scénarios de complexité croissante sont envisagés pour leur missions dans (Weatherington, 2005). Ces scénarios s'étendent depuis le pilotage manuel du drone depuis une station au sol jusqu'à son autonomie complète, en passant par un scénario intermédiaire de suivi de trajectoire. Dans le cas du suivi de trajectoire, le drone doit être capable de planifier à nouveau sa trajectoire lorsqu'un obstacle est détecté, pour le contourner. Compte de tenu des temps de réponse et de l'autonomie limitée du drone, cette fonction de re-planification est réalisée par le drone lui-même, sans intervention de la station au sol. Cette fonction composée d'une détection d'obstacle, d'une prise de décision et d'un calcul de nouvelle trajectoire nécessite une grande capacité de traitement de la part de la plateforme HW, alors que les ressources énergétiques sont limitées. Cette dépense énergétique est généralement bien plus faible que celle des moteurs, cependant sa réduction peut permettre de gagner les quelques watts nécessaire à la réalisation d'une mission. Dans le cas des nano-drones dont la masse ne dépasse pas 30 grammes, ces dépenses énergétiques deviennent comparables.

Dans la suite de cette étude nous allons nous intéresser à la partie traitement d'image et détection d'obstacle. Cette fonction est associée à une caméra haute définition unique et non stéréoscopique car le drone ne peut embarquer une charge utile de masse limitée. L'application SW embarquée réalise un traitement d'image afin de reconnaitre la couleur d'un objet,

et de déterminer la distance de celui-ci par rapport au drone. Une série de filtres permet d'obtenir l'image présentée Figure 66 « image de la caméra avant et après filtrage ». La distance est fortement approximée en raison de la vision 2D de la caméra.

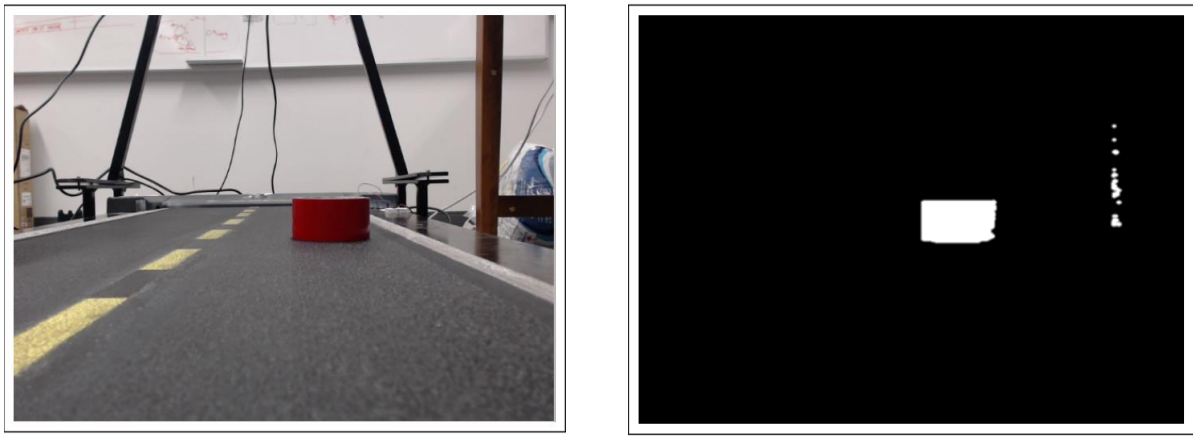


Figure 66 « image de la caméra avant et après filtrage »

L'application SW dispose d'une première tâche réalisant la capture d'image. Les images capturées sont stockées dans une file d'attente FIFO. Puis deux tâches de filtrages et une tâche de restitution permettent de compléter le traitement des images stockées. Les tâches de filtrages utilisent la librairie OpenCV (Laganière, 2011) permettant de réaliser les opérations nécessaires.

Pour la plateforme HW, une architecture multi-cœur hétérogène (SHMC) permet d'obtenir un compromis entre une grande puissance de calcul et une consommation énergétique moindre. Un processeur SHMC permet de réduire la consommation énergétique de 39% tout en diminuant de 3% les performances, par rapport à un processeur mono-cœur ou même par rapport à un processeur multi-cœur homogène (cœur identiques) selon (KUMAR, 2003). En effet le processeur SHMC dispose de cœurs différents par leur puissance de calcul, mais utilisant le même jeu d'instruction. Une tâche logicielle peut donc être allouée indifféremment d'un cœur à l'autre, sans besoin de recompilation. Les cœurs peuvent également être stoppés lorsqu'aucune tâche ne leur est allouée, sans avoir besoin d'être maintenu dans un mode de veille ou « Idle ». Sur ce type de processeur on trouve généralement un ensemble de cœur plus puissant (Big core) et un ensemble de cœur moins puissants (little core), cette architecture étant appelée « Big.Little » (Greenhalgh, 2011) (Padoin, 2015). La plateforme utilisée pour le cas d'étude est un processeur Exynos 5422, incluant quatre cœurs Cortex A15 « BigCore » et quatre cœurs Cortex A7 « LittleCore ». La puissance maximum dissipée est de 4 Watt pour un « BigCore » et de 1 Watt pour un « LittleCore ».

Modélisation de la plateforme HW avec SysML

La modélisation SysML de la plateforme HW Exynos 5422 est présentée dans la Figure 67 « Modélisation SysML de la plateforme HW du drone ». Le profil MARTE est utilisé pour modéliser le processeur SHMC, conjointement avec SysML. Le processeur 5422 dispose de deux mémoires caches L2, pour chaque ensemble de quatre cœurs, ainsi que d'une mémoire DRAM. Chacun des cœurs dispose également d'une mémoire cache L1.

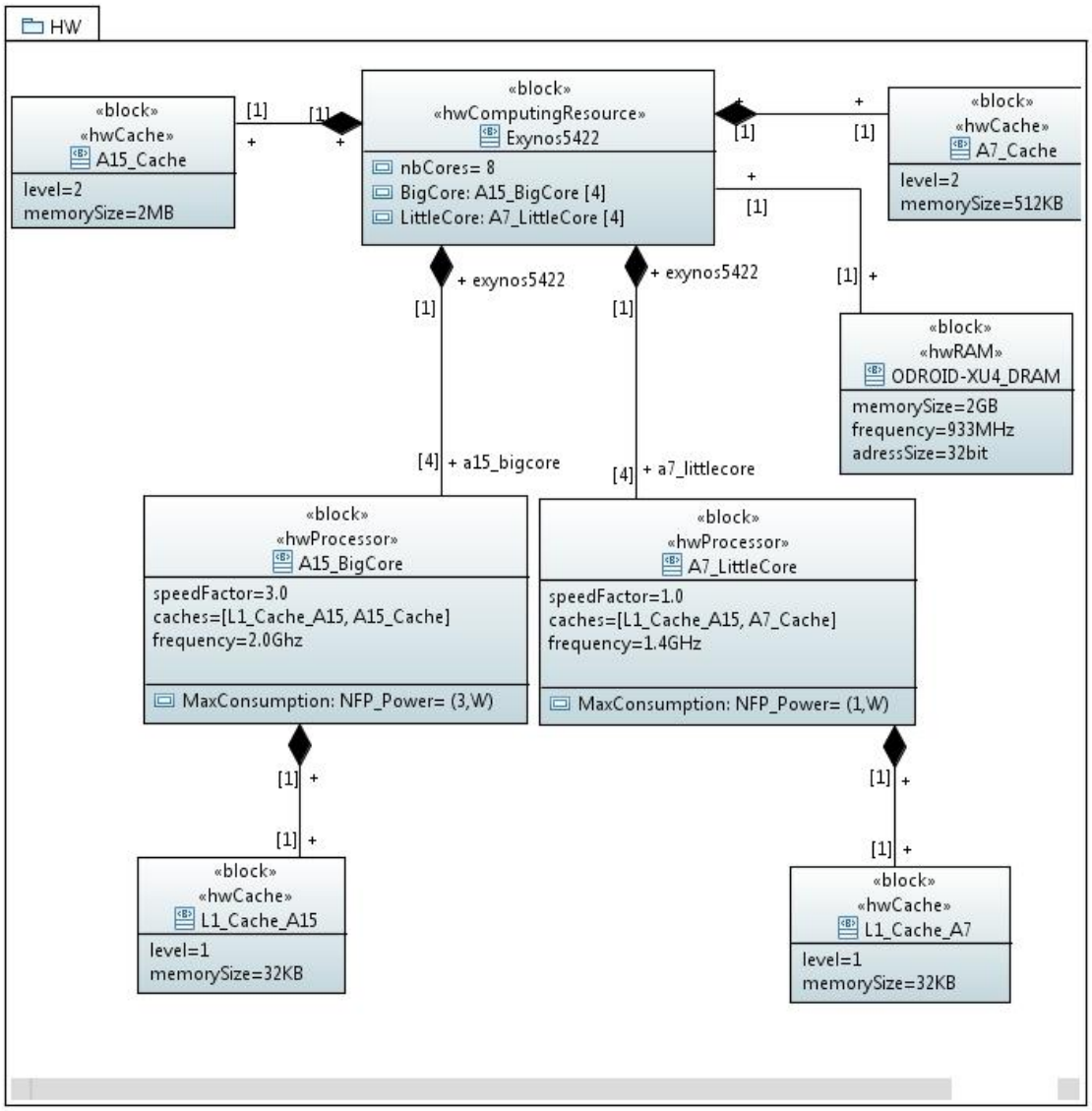


Figure 67 « Modélisation SysML de la plateforme HW du drone »

Modélisation SW, de l'allocation et des contraintes avec SysML

L'application que nous souhaitons modéliser est composée de neuf tâches présentées dans la Tableau 18 « taches de l'application de détection d'obstacle ». La première tâche « GetFrame » va capturer les images de la caméra et va les placer en mémoire dans une file d'attente de type « FIFO ». La seconde tâche « ShowPicture » va répartir ces images afin qu'elles puissent être filtrées par les deux tâches de filtrages. Les tâches « Filtering 1 » et

« Filtering 2 » vont filtrer les images stockées en file d'attente, afin d'extraire les contours des objets présents dans l'image. Ces deux tâches disposent d'un WCET important, mais peuvent être allouées à deux cœurs différents pour une exécution en parallèle, un sémaphore de type « Mutex » permettant de se prémunir d'un accès concurrent à la même image. La tâche « Obstacle detection » permet de déterminer si l'image contient un objet susceptible d'être considéré comme un obstacle. Dans cette version du logiciel, le critère retenu est basé sur la couleur de l'objet. La dernière tâche « Trajectory replanning » va effectuer une replanification de la trajectoire du drone, en fonction des paramètres de l'objet détecté (taille et distance). Les temps WCET ont été mesurés sur la carte ODROID en instrumentant le code embarqué avec un timer HW, sur une série de 1000 images différentes, et avec une consigne d'acquisition de 20 images par seconde.

Id	**Name**	**WCET for BigCore(ms)**	*Period (ms)*	*Heap + Stack Memory size(KB)*
s_0	Get Frame	83	100	600
s_1	Show Picture	1	90	100
s_2	Filtering 1	137	200	200
s_3	Filtering 2	111	200	200
….				
s_8	Obstacle detection	90	20	300
s_9	Trajectory Replanning	110	300	400

Tableau 18 « taches de l'application de détection d'obstacle »

La modélisation des tâches, de leurs allocations et des contraintes associées est présentée Figure 68 « Modélisation des tâches et des allocations ». Nous utilisons un diagramme SysML « IBD » dans lequel chaque tâche est modélisée par un bloc SysML. Chaque bloc est également complété avec les stéréotypes « schedulableResource » du profil MARTE. Le stéréotype « AllocatedTo » représente les différentes possibilités d'allocation de cœur pour chaque tâche. Lorsqu'il n'est pas présent sur une tâche, cela signifie que cette tâche peut être allouée à n'importe quel cœur. Des contraintes globales d'allocation « SameCore » et « DifferentCore » sont également utilisées. Ces contraintes permettent de rassembler les tâches partageant un grand nombre de donnée (images stockées après acquisition) ou d'en séparer d'autre pour le parallélisme ou la sûreté de fonctionnement (en cas défaillance d'un cœur).

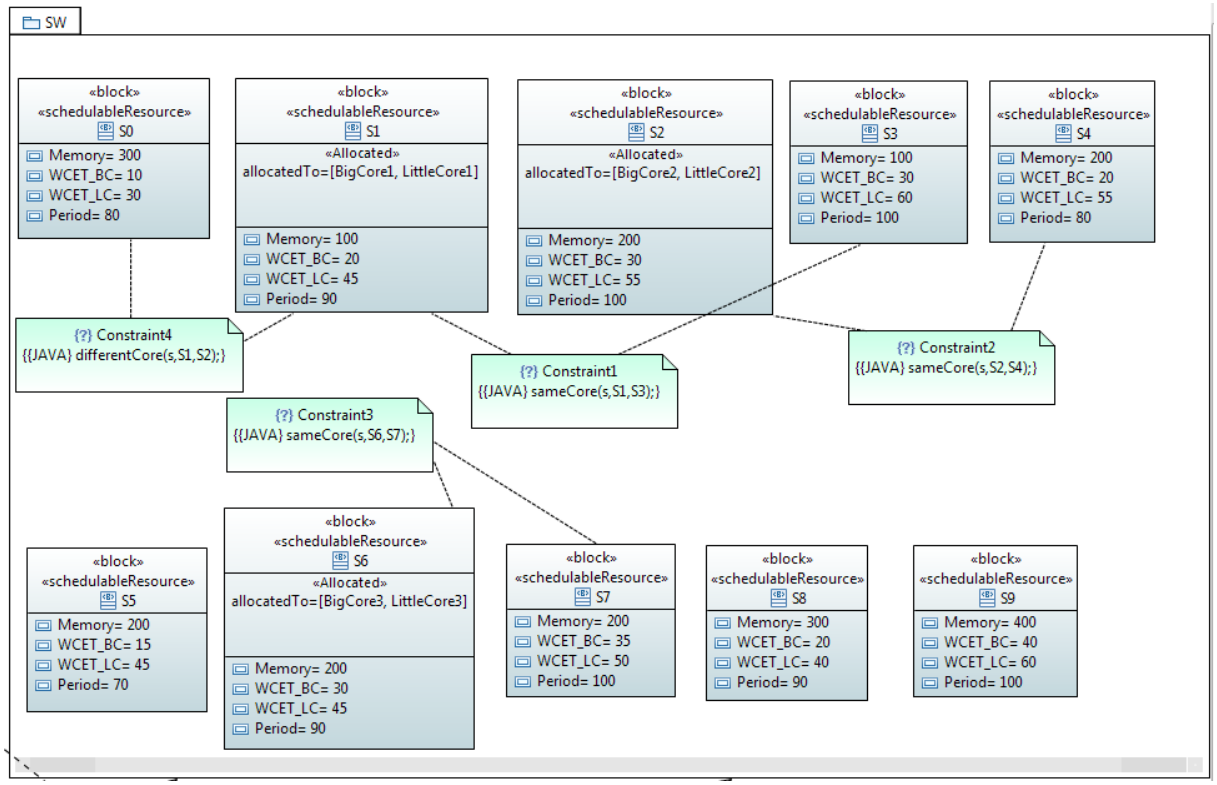


Figure 68 « Modélisation des tâches et des allocations »

Deux autres contraintes de type « Capacity » sont également modélisées dans ce cas d'étude. La première concerne la quantité mémoire de mémoire RAM disponible dans chaque cœur pour exécuter les tâches qui lui sont allouées. Il s'agit d'une contrainte de ressource appliquée à la mémoire, que nous avons exprimé précédemment par l'Équation 12 « contrainte de ressource pour l'allocation » dans la section 2.4. La deuxième contrainte concerne le taux d'utilisation de chaque cœur, qui ne doit pas dépasser 100%, et exprimé par Équation 13 « contrainte d'utilisation pour l'allocation ». Ces contraintes sont présentées dans le figure suivante.

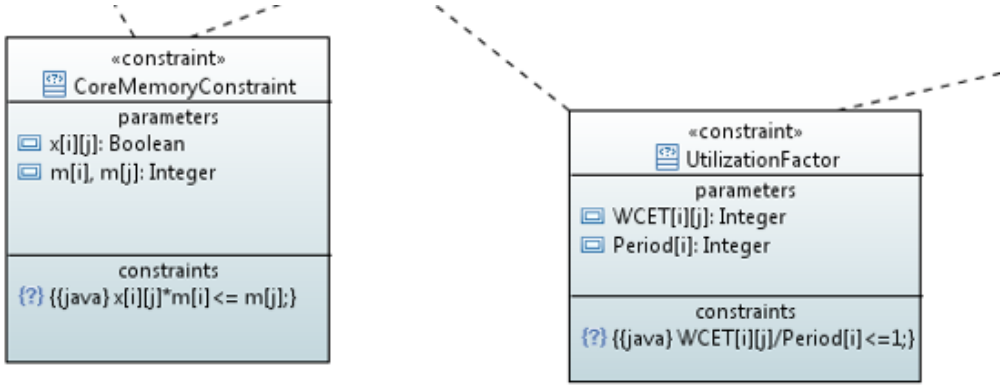


Figure 69 « Blocs de contraintes mémoire et facteur d'utilisation »

Concernant les fonctions objectifs modélisées dans le contexte d'optimisation, celle-ci sont au nombre de deux. La première exprime le nombre de cœurs réellement utilisés dans une

configuration. Comme le processeur SHMC permet d'arrêter complétement un cœur non-utilisé, nous souhaitons minimiser le nombre de cœurs utilisés. L'Équation 15 « Nombre de coeurs utilisés en fonction de l'allocation » permet d'exprimer cet objectif en fonction des variables de décision d'allocation x_{ij} , avec m le nombre de cœur et n le nombre de tâches.

$$min\ (S)\quad where\ S = \sum_{j=1}^{j=m} \max_{i\in\{1..n\}}(x_{ij})$$

Équation 15 « Nombre de coeurs utilisés en fonction de l'allocation »

La seconde fonction objectif représente la puissance totale consommée par tous les cœurs, à minimiser et donnée par

$$min(P)\ where\ \ P = \sum_{j=1}^{j=m} \sum_{i=1}^{i=n} x_{ij} \frac{C_{ij}}{T_i}$$

Équation 16 « Puissance totale dissipée en fonction de l'allocation »

Génération du problème CSMOP

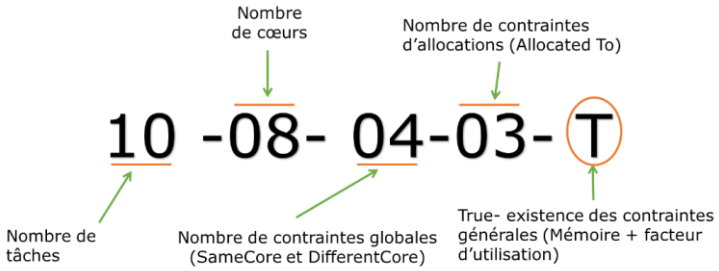
A partir du contexte d'optimisation SysML précédent, l'Algorithme 5 « Problème d'optimisation CSMOP en JAVA avec CHOCO » est généré pour le langage Java et le solveur CHOCO (Jussien, 2008) (CHOCO, 2016) . Ce solveur est choisi en raison des nombreuses contraintes dont il dispose, et de leur plus grande expressivité, par rapport au solveur *Labix* précédemment utilisé. Dans l'Algorithme 5, Les valeurs des constantes C_MAX (nombre de cœurs) et S_MAX (nombre de tâches) sont récupérées dans la modélisation SysML de la plateforme HW et des tâches. Les variables de décision "*s*" and "T_s" sont des matrices de booléens, créées par le solveur. "T_s" est la matrice transposée de "*s*". Les contraintes d'allocation basique correspondent à l' Équation 10 «contrainte d'unicité pour l'allocation» et sont implémentées avec les contraintes "*AtMostOne*" et "*OR*" du solveur CHOCO. Les contraintes globales d'allocation "*SameCore*" and "*DifferentCore*" sont implémentées avec une égalité booléen ainsi qu'une contrainte "*AND*" appliquée au vecteur de booléen `s[i]`. Les contraintes de mémoire et d'utilisation sont appliquées à chaque cœur. La contrainte « CoreMemoryConstraint » est de type « Capacity », elle nous est donnée par l' Équation 12 « contrainte de ressource pour l'allocation » de la page 69. Lorsqu'une contrainte de type « Capacity » est présente dans le modèle, nous utilisons le produit scalaire «`ICF.scalar`» du solveur. Dans les paramètres de cette contrainte, nous retrouvons la matrice transposée "T_s" , ainsi qu'un vecteur contenant la taille mémoire des différentes tâches « `MemoryTask` » et le vecteur des WCET par cœur « `MemCLittle` ». Ces informations sont récupérées dans le modèle, dans les attributs correspondants.

```
// Schedulable Resource number
private static final int S_MAX = 10;
// Core number
private static final int C_MAX = 8;
Solver solver = new  Solver("Allocation problem");
// Create variables through the variable factory
// A matrix of MAX_T tasks (s0..sn) and MAX_C cores (c0..cm)
BoolVar[][] s = VariableFactory.boolMatrix("s", S_MAX,C_MAX, solver);
// A transposed matrix of MAX_C cores (c0..cm) cores and MAX_T tasks (s0..s9)
BoolVar[][] Ts = VariableFactory.boolMatrix("Ts", C_MAX,S_MAX, solver);
// Basic allocation constraints :
for(int i=0;i<S_MAX;i++) {
// A task is allocated to one Core and at least once
   SatFactory.addAtMostOne(s[i]);
   SatFactory.addBoolOrArrayEqualTrue(s[i]);
   for(int j=0;j<C_MAX;j++) {
   // Transposed matrix constraint
       SatFactory.addBoolEq(s[i][j], Ts[j][i]);
   }
}
// Global constraints for Allocation
for(int j=0;j<C_MAX;j++) {
   SatFactory.addBoolEq(s[3][j], s[1][j]);
   SatFactory.addBoolEq(s[4][j], s[2][j]);
   SatFactory.addBoolAndEqVar(s[1][j], s[2][j], False);
}
// Memory Constraints for Little cores
for(int j=C_MAX/2;j<C_MAX;j++)
   solver.post(ICF.scalar(Ts[j],MemoryTask,"<=",MemCLittle));
// Utilization factor constraint for each core
for(int j=0;j<C_MAX;j++) {
for(int i=0;i<S_MAX;i++) {
     TArray[i]= 100*WCET_Taski_Cj[i][j]/PeriodTask[i];
   }
   solver.post(ICF.scalar(Ts[j], TArray,"<=", Hundred));
}
//Define the search strategy
solver.set(IntStrategyFactory.lexico_LB(s1));
// Problem resolution
solver.findSolution()
do{
   //For each allocated resource
   PowerPerCore = PowerPerCore +Ts[i][j].geValue()*100*WCET_Taski_Cj[j][i]/PeriodTask[j]
   ;
}while(solver.nextSolution());
```

Algorithme 5 « Problème d'optimisation CSMOP en JAVA avec CHOCO »

Tests et analyse des résultats

Trois ensembles de configurations ont été modélisés, et l'obtention des solutions optimales a pu être testée pour chaque configuration. Le premier objectif des tests a été de vérifier l'obtention de solutions optimales, permettant de réduire l'espace des solutions avant de procéder aux analyses d'ordonnançabilité. Pour chacune des solutions obtenues, une analyse complémentaire d'ordonnançabilité de type « OSEK » a pu être menée en utilisant l'outil SymTA/S (Henia, 2005) d'analyse temporelle symbolique. Le second objectif a été de déterminer les limites de l'approche par rapport à la taille du problème à résoudre. Pour identifier les différents cas de test, nous utilisons la notation suivante :



Les tableaux suivants présentent les attributs des trois ensembles de configurations C_1(10-08), C_2(14-08), C_3(18-08). Ces configurations disposent d'un nombre variable de taches (S_0..S_{17}) à allouer à 8 cœurs (BC_1..BC_4 et LC_1..LC_4). Les configurations C_{1G}, C_{2G}, C_{3G} possèdent des contraintes globales de type « SameCore » et « DifferentCore » ce qui n'est pas le cas de C_1, C_2, C_3.

Configuration	C_1 (**10**-08)																	
	C_2 (**14**-08)																	
	C_3 (**18**-08)																	
Id	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}	S_{16}	S_{17}
WCET (ms)	30	45	55	60	55	45	45	50	40	60	75	75	75	75	75	90	105	60
Période (ms)	80	90	100	100	80	70	90	100	90	100	80	80	80	90	90	100	110	70
Priorité (min=1)	4	3	2	2	4	5	3	2	3	2	4	4	4	3	3	2	1	5

Tableau 19 « WCET et Périodes »

C_1(10-08-00-**03**-T)	C_2(14-08-00-**07**-T)	C_3(18-08-00-**08**-T)
S_1 to BC_1, LC_1	S_1 to BC_1, LC_1	S_1 to BC_1, BC_2
S_2 to BC_2, LC_2	S_2 to BC_2, LC_2	S_2 to LC_3, LC_2
S_6 to BC_3, LC_3	S_4 to BC_4, LC_4	S_5 to BC_4, LC_3
	S_6 to BC_3, LC_3	S_7 to BC_3, LC_2
	S_8 to BC_1, LC_1	S_8 to BC_1, LC_4
	S_{11} to LC_3, LC_4	S_{10} to BC_1, LC_1
	S_{13} to BC_1, BC_2	S_{14} to LC_2, BC_2
		S_{17} to BC_3, LC_3

Tableau 20 « Contraintes d'allocation »

C_{1G}(10-08-**04**-03-T)	C_{2G}(14-08-**04**-07-T)	C_{3G}(18-08-**06**-08-T)
SameCore(S_1,S_3)	SameCore(S_1,S_3)	SameCore(S_1,S_3)
SameCore(S_2,S_4)	SameCore(S_6,S_{12}, S_{10})	SameCore(S_{11},S_{12})
SameCore(S_6,S_7)	DifferentCore(S_1,S_2)	SameCore(S_{13}, S_{15})
DifferentCore(S_1,S_2)	DifferentCore(S_{10},S_{13})	DifferentCore(S_6,S_{12})
		DifferentCore(S_1,S_2)
		DifferentCore(S_{16},S_2)

Tableau 21 « Contraintes globales pour C_{1G}, C_{2G}, C_{3G} »

Mise en œuvre des tests

Chaque configuration est modélisée avec l'outil Papyrus, en utilisant les stéréotypes d'allocation ainsi que les contraintes d'allocation et les contraintes globales. Les fonctions objectifs sont également modélisées : le nombre de cœurs à minimiser ainsi que la puissance totale dissipée. La description du problème CSMOP est alors générée automatiquement, pour le solveur CHOCO. Le problème est alors résolu par le solveur, et les solutions optimales sont calculées avec l'algorithme de dominance. Le résultat est présenté à l'utilisateur sous forme d'un front de Pareto, complété par une table de solutions. Les solutions optimales peuvent alors être analysées par l'outil SymTA/S pour déterminer si elles sont ordonnançables ou pas. Pour déterminer ordonnançabilité, nous avons affecté aux tâches une priorité selon le principe « Rate Monotonic », la priorité maximum correspondant à la tâche de plus faible période. Les tâches sont analysées selon le modèle « OSEK ». Les figures suivantes montrent les diagrammes de Pareto et les tables obtenues pour les trois ensembles de configuration C_{1G}, C_{2G} , C_{3G} avec 10, 14 et 18 tâches et plusieurs contraintes globales. Chaque point figure non pas mais plusieurs solutions dont les valeurs des objectifs sont identiques. Ces solutions vérifient le problème CSMOP, avant l'analyse d'ordonnançabilité.

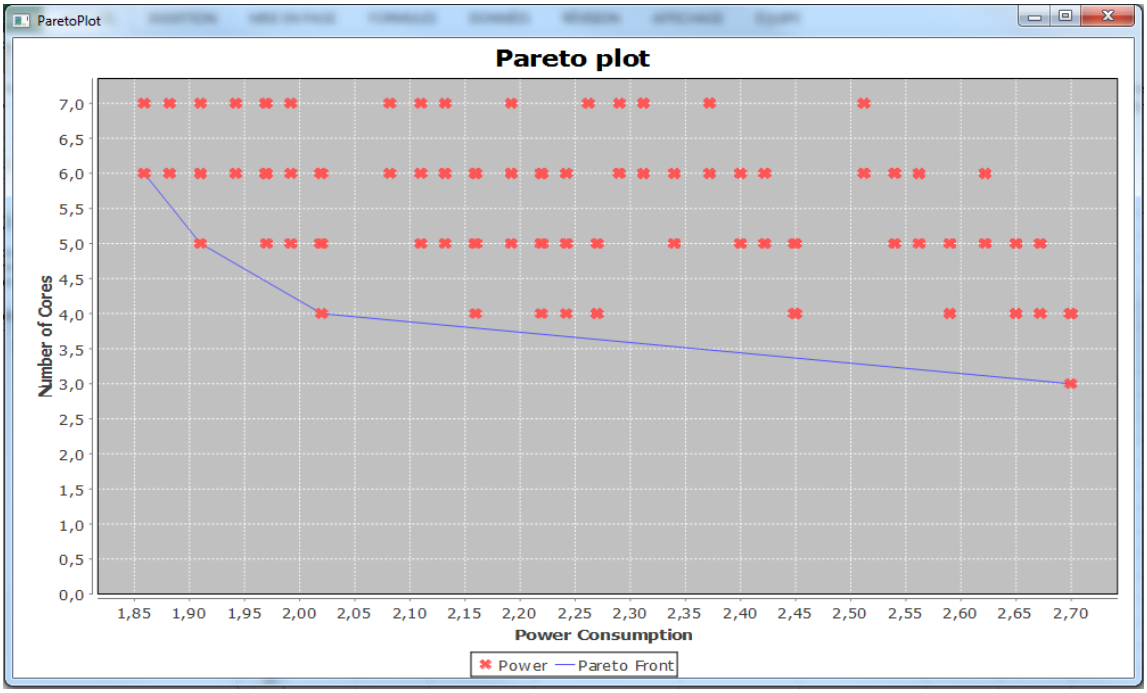


Figure 70 « Solutions de C_{1G} 10-08-04-03-T »

Nombre de cœurs	Alloca-tions pos-sibles	Puis-sance (W)	Charge du Cortex A15 (%)	Charge du Cortex A7(%)	Ordonnan-çable ?
3	1	2.699	1.799	0	Oui
4	9	2.02	1.266	0.6	Non
5	18	1.91	1.118	1.044	Non
6	12	1.859	0.904	1.687	Non

Tableau 22 « Solutions optimales de C_{1G} »

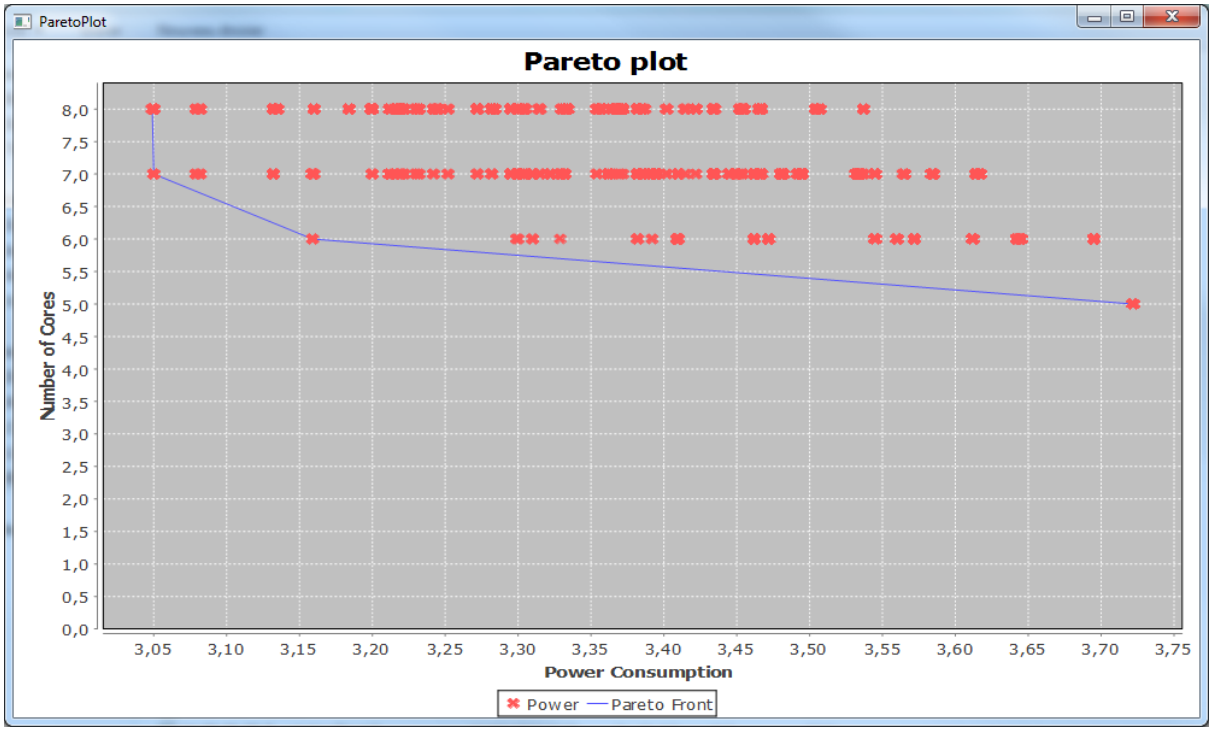


Figure 71 « Solution de C_{2G} 14-08-06-05-T »

Nombre de cœurs	Alloca-tions pos-sibles	Puis-sance (W)	Charge du Cortex A15 (%)	Charge du Cortex A7(%)	Ordon-nan-çable ?
5	24	3.722	2.669	0.937	Oui
6	4	3.159	2.152	2.487	Oui
7	8	3.05	2.004	2.931	Oui
8	8	3.049	2.004	2.931	Oui

Tableau 23 « Solutions optimales de C_{2G} »

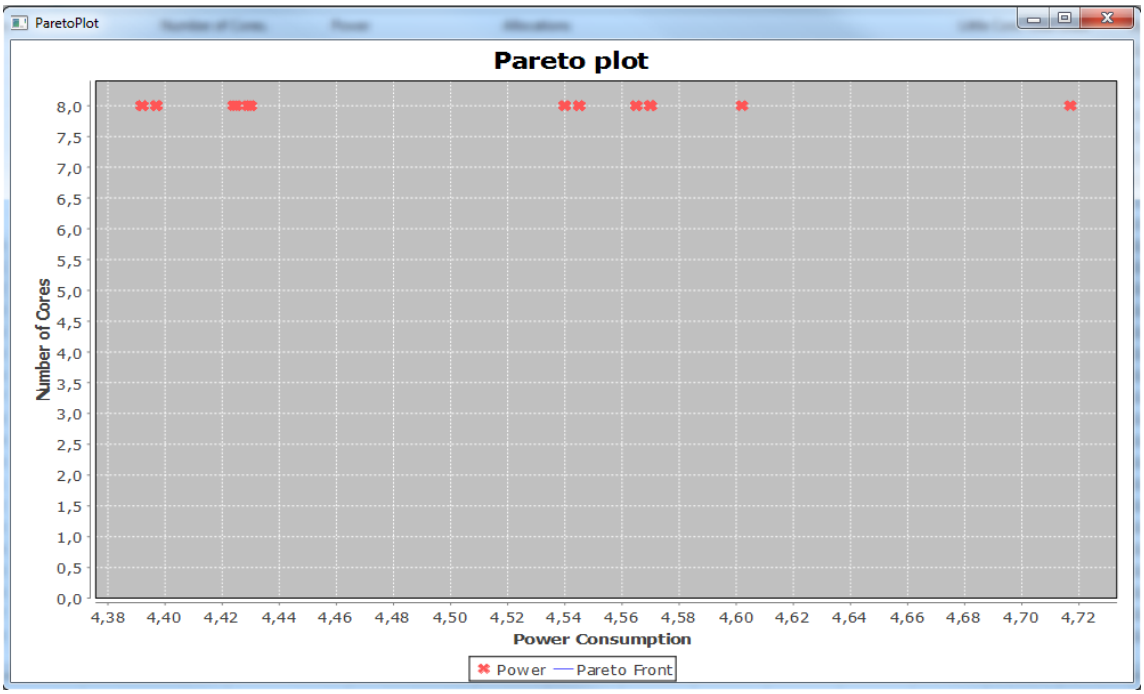


Figure 72 « Solution de la configuration C_{3G} 18-08-08-06-T »

Nombre de cœurs	Allocations possibles	Puissance (W)	Charge du Cortex A15 (%)	Charge du Cortex A7(%)	Ordonnançable ?
8	24	4.395	3.515	2.427	Oui

Tableau 24 « Solutions optimales de C_{3G} »

Nombres de solutions obtenues et temps de calcul

Les tableaux suivants présentent le nombre de solutions obtenues (solution totales), de solutions optimales et de solutions ordonnançables par analyse complémentaire SymTA/S. Pour chaque configuration identique en nombre de tâches, nous avons testé une configuration sans et une configuration avec contraintes globales. Les temps de calcul sont également présentés, et sont obtenus avec un PC de type Intel Xeon CPU ES-1607 3GHz disposant d'une mémoire RAM de 16 GB.

Configuration	Solutions totales *(temps calcul)*	Solutions optimales *(temps calcul)*	Solutions ordonnançables *(temps calcul)*
C_1(10-08-00-03-T)	1 693 095 (225 s)	35 (11,3 s)	3 (111,7 s)
C_{1G}(10-08-04-03-T)	3 135 (0,5 s)	40 (0,4 s)	1 (126,3 s)
C_2(14-08-00-07-T)	1 875 074 (413 s)	478 (0,1 s)	478 (1346 s)
C_{2G}(14-08-05-07-T)	4 602 (0,5 s)	44 (0,1 s)	44 (124 s)
C_3(18-08-00-08-T)	112 178 (11 s)	984 (4,3 s)	984 (3490 s)
C_{3G}(18-08-06-08-T)	420 (0,5 s)	24 (<0,1 s)	24 (119 s)

Tableau 25" Nombre solutions et temps de calcul"

L'utilisation de contraintes globales permet de réduire significativement le nombre de solutions pour les trois configurations testées, comme le montre également la Figure 73 « Réduction de l'espace des solutions ». Sans ces contraintes il serait impossible de réaliser directement l'analyse d'ordonnançabilité qui dure entre une et trois secondes par configuration comme le montrent les figures suivantes.

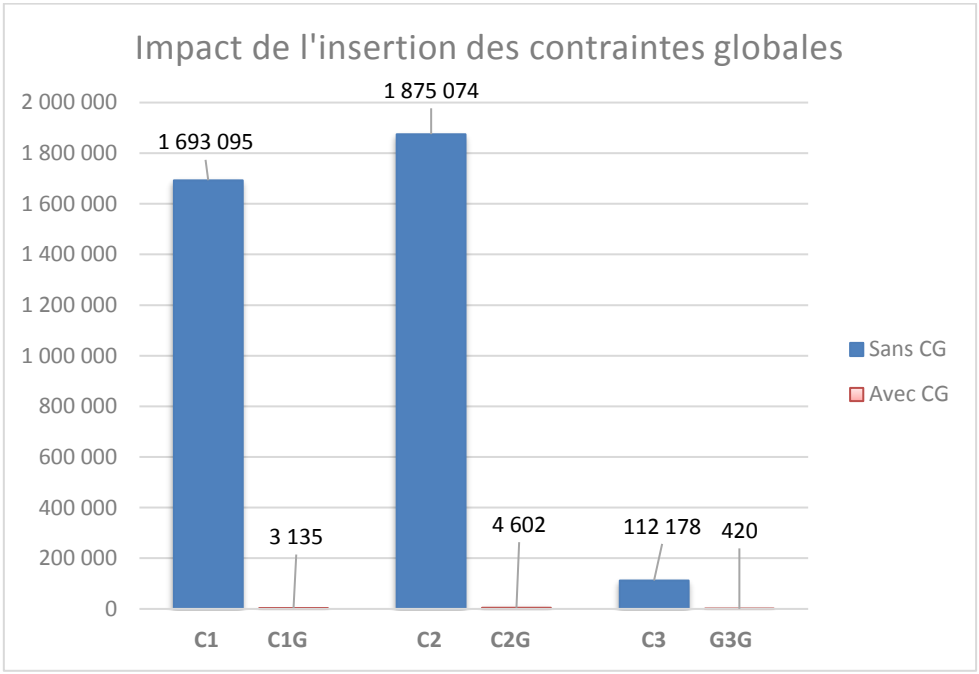


Figure 73 « Réduction de l'espace des solutions »

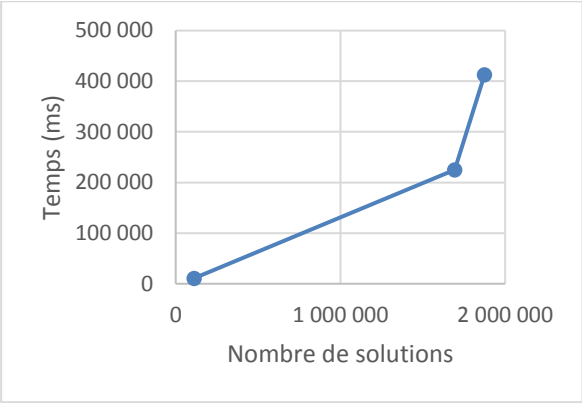


Figure 74 « Temps de calcul sans contraintes globales »

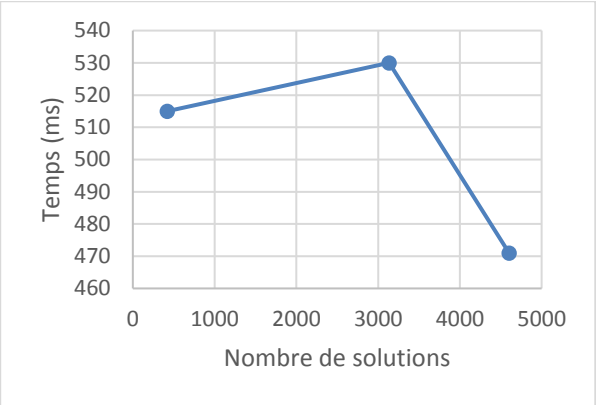


Figure 75 « Temps de calcul avec contraintes globales »

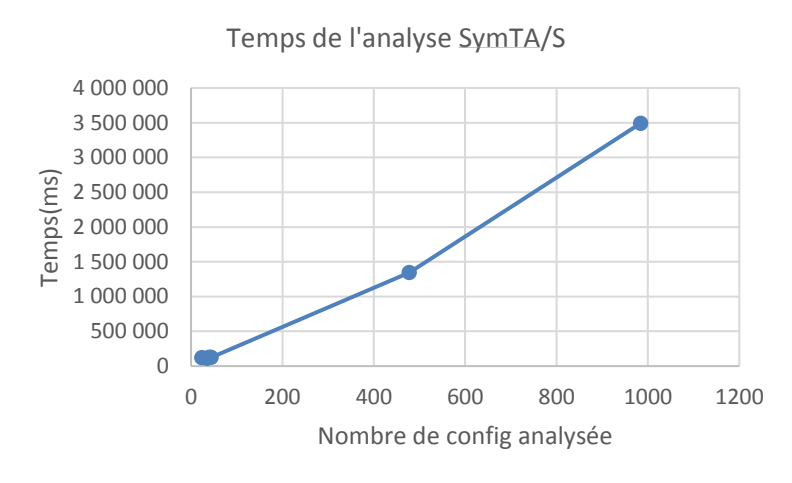


Figure 76 « analyse SymTA/S »

4.3.Conclusion sur les cas d'études et l'outillage

Dans ce chapitre, nous avons présenté trois cas d'études correspondant au trois types de points de décision présenté dans les chapitre 3 et 4. Le premier concerne un choix de composant ainsi qu'un niveau redondance, le second intègre des variables et le troisième consiste à résoudre un problème d'allocation optimale. Les algorithmes de transformation des chapitre 3 et 4 ont été implémentés et testés dans l'outil Papyrus, sous forme d'un plug-in. Ces algorithmes ont permis de générer un code compatible avec trois différents solveurs, à savoir Labix, PyOpt et CHOCO puis de résoudre le problème. Les temps d'exécution ont pu être mesurés en faisant varier le nombre de points de décision. Pour les points de décisions « discrets » (Choix de composant, redondance, allocation) le solveur CHOCO apparait bien plus performant que Labix. Il dispose également d'un grand nombre de contraintes notamment les contraintes globales permettant de réduire l'espace de solutions. Pour les problèmes continus ou mixte, le solveur PyOpt est plus adapté, en particulier pour les fonctions objectifs non-linéaire. Concernant l'outillage, le choix de la plate-forme Eclipse et de l'outil Papyrus a permis d'intégrer les différents solveurs ainsi que de s'interfacer avec un outil externe d'analyse comme SymTA/S. La limitation de la plateforme Eclipse concerne sa mise en œuvre relativement complexe et ses nombreuses mises à jour régulières.

5. Conclusion et perspectives

5.1. Conclusion

La complexité des systèmes embarqués a fortement augmenté au cours des dernières années. Ces systèmes doivent intégrer toujours plus de fonctions, à un coût acceptable. L'ingénieur système doit alors déterminer les architectures optimales du système, sachant que les objectifs comme le coût, la performance ou la fiabilité sont bien souvent contradictoires.

L'avènement des méthodes MBSE (OOSEM, ARCADIA) et du langage SysML marque une étape dans l'ingénierie des systèmes complexes. Cependant, lorsqu'il s'agit d'obtenir les solutions optimales, ces méthodes sont encore limitées. Dans la phase de conception de l'architecture, l'ingénieur système doit modéliser une à une les différentes variantes et évaluer les objectifs. Ceux-ci sont pondérés et regroupés dans une « mesure d'efficacité » obligeant le concepteur à réaliser un choix à priori parmi les architectures possibles. Plutôt que de modéliser chaque variante du système, il faut également pouvoir modéliser l'ensemble des variantes. Les langages OVM et CVL permettent de modéliser la variabilité dans les lignes de produit et pourraient être utilisés. Cependant leur mise en œuvre est complexe, car elle oblige l'utilisateur à réaliser et à maintenir un second modèle contenant les points de variation. L'objectif de ces langages est de décrire des lignes de produits, plutôt que d'assister le concepteur dans l'optimisation du système. L'ingénieur système peut également faire appel à un mathématicien ou à un spécialiste de l'optimisation, pour obtenir une représentation abstraite du problème. Dans ce cas, deux approches sont possibles : la programmation mathématique et la programmation par contrainte. Le spécialiste de l'optimisation doit alors reformuler le problème et échanger avec l'ingénieur système, au risque de perdre d'importantes informations. Si l'on rajoute à cela la complexité du système, on arrive à un long processus d'optimisation.

Obtenir les variantes optimales d'un système complexe à partir d'un modèle constitue donc le verrou scientifique principal que nous souhaitons lever.

Les contributions de cette thèse sont de trois ordres :

- La définition d'une méthode MBSE prenant explicitement en compte les problèmes d'optimisation multi-critères et les choix associés ;
- La définition d'une version étendue de SysML incluant de nouveaux éléments comme des points de décision et un contexte d'optimisation. Des algorithmes permettant de transformer ces éléments pour leur exploitation par les solveurs sont également proposés.
- Le support de ce SysML étendu par l'outil Papyrus et la démonstration de l'approche proposée sur plusieurs études de cas significatives.

La méthode MBSE que nous proposons (section 4.1) s'applique dans la phase de conception de l'architecture du système. Dans une première étape, le concepteur va enrichir le modèle du système avec des points de décision définis dans les section 2.2 et 3.1. Ces points de décisions correspondent à des choix de composants ainsi qu'au niveau de redondance des composants. Dans les approches existantes, cette étape est bien souvent ignorée, la topologie du système est figée. Les points de décisions peuvent s'intégrer dans un langage MBSE de modélisation système (section 2.2), à condition que ce langage puisse décrire une architecture hiérarchique de composants, et qu'il dispose des notions d'allocation et d'attributs. Le modèle incluant ces points de décision peut alors être transformé, pour obtenir un problème de satisfaction de contraintes pour l'optimisation CSMOP (Constraint Satisfaction Multi-criteria Optimization Problem). Des algorithmes de transformation sont proposés dans les sections 2.6 et 3.3. Ces algorithmes permettent notamment de créer les variables utilisées par les solveurs, en fonction des points de décisions du modèle.

Des contraintes d'association ou d'exclusion entre les composants ont pu être rajouté (section 2.4). Pour le langage SysML, les points de décisions sont définis par extension du métamodèle SysML dans le chapitre 3. Pour cette première étape, le modèle d'une caméra embarquée dans un véhicule a été développé dans la langage SysML (cas d'étude du paragraphe 4.2.1), incluant les stéréotypes pour les points de décision. L'outillage (section 4.1) est composé de l'outil Papyrus dans l'environnement Eclipse, d'un plug-in développé en JAVA pour la transformation ainsi que du solveur « Labix » pour résoudre le problème CSMOP. Nous obtenons un ensemble de solutions optimales pour le coût et la fiabilité. La principale limitation concerne le nombre de points de décisions, une quinzaine au maximum pour avoir un temps de résolution raisonnable.

Dans une seconde étape de la méthode (section 4.1), une fois la topologie et les composants choisis, le concepteur peut configurer les composants. Il s'agit de déterminer les valeurs optimales de certains attributs du modèle. Là encore, de nouveaux stéréotypes ainsi qu'une adaptation des transformations permettent d'obtenir un problème CSMOP (section 2.2 et 3.2). Par contre la mise en œuvre de variables de type continue est nécessaire dans cette étape. Nous avons utilisé un autre solveur « PyOpt » adapté à ce type de problème, en l'intégrant dans Papyrus et Eclipse. Dans cette étape, nous avons également défini la notion de contexte d'optimisation du système (section 2.5 et 3.2), associant le système avec les points de décisions ainsi que les fonctions objectifs et le solveur. Une nouvelle utilisation des diagrammes BDD et paramétrique de SysML est proposée à cet effet. Un second cas d'étude constitué d'une caméra stéréoscopique a été développé, modélisant les équations de l'optique stéréoscopique. Il permet de montrer comment optimiser les performances avec une fonction objectif non-linéaire et le coût d'un système optique intégrant des capteurs et des lentilles.

Dans une troisième étape, l'allocation de composants sources sur des composants destination est prise en compte (section 2.2 et 3.1). Il peut s'agir de ranger des composants phy-

siques dans des compartiments, ou bien d'allouer des tâches d'exécution à des unités d'exécution (problème d'ordonnancement). C'est le cas du contrôleur de mission d'un drone (le cas d'étude du paragraphe 4.2.3), pour lequel nous avons réalisé le modèle SysML/MARTE et l'outillage associé permettant de déterminer les solutions optimales. Ces solutions sont ordonnançables, et minimisent les charges d'exécution des processeurs. Pour cela, nous avons défini des stéréotypes permettant de représenter cette allocation variable, puis adapté l'algorithme de transformation précédent, et utilisé le solveur Choco. En effet ce solveur autorise l'utilisation des contraintes globales, utiles à ce type de problème. Ces contraintes augmentent les performances du solveur, et permettent de résoudre un problème avec d'avantages de variables dans un temps raisonnable. Nous avons pu résoudre un problème d'allocation disposant de dix-huit tâches, avec huit processeurs. Pour chaque solution optimale, nous avons réalisé une analyse détaillée d'ordonnançabilité en utilisant un outil d'analyse détaillée SymTA/S (le cas d'étude du paragraphe 4.2.3). Cette approche permet de réduire significativement le nombre de solutions, avant d'effectuer une analyse détaillée coûteuse en temps de calcul.

L'outillage (section 4.1) permettant d'effectuer ces trois étapes constitue une solution intégrée et évolutive autour de la plate-forme de logiciels libres Eclipse/Papyrus. Il est possible d'ajouter d'autre solveurs si besoin, disponibles dans des langages comme JAVA (solveur CHOCO) ou Python (solveurs Labix ou PyOpt).

5.2.Perspectives

5.2.1. Extension à CVL et à d'autres langages de modélisation

Bien que CVL ne soit pas standardisé, et que l'outillage correspondant reste assez peu répandu, ce langage pourrait être également être utilisé dans notre méthode. La modélisation des points de variations dans un modèle CVL offre des perspectives, notamment pour la génération des solutions du problème d'optimisation. En effet il est possible de générer avec un script un modèle de résolution décrivant une variante du modèle de base. Cette évolution est possible car les points de décisions que nous avons proposés sont compatibles avec les points de variation de CVL. Les algorithmes de transformation que nous avons développés devront être adaptés pour analyser ces points de variation et générer un problème CSMOP. Ces algorithmes devront cependant prendre en compte le contexte d'optimisation défini dans le modèle de base. CVL permet également de rester indépendant d'un langage de base, ce qui est un avantage. L'autre possibilité d'évolution consiste à intégrer les points de décisions que nous avons proposés ainsi que les notions de contexte d'optimisation dans d'autres langages équivalents à SysML. Cela est possible si ce langage dispose de notions de classes, d'attributs et d'allocation. C'est le cas du langage AADL comme nous l'avons présenté dans le tableau Tableau 3 « correspondance Elément générique/SysML/AADL » de la page 56. Il faut également que ce langage puisse être étendu, pour inclure les notions de points de décision, de nouvelles contraintes et le contexte d'optimisation. AADL peut l'être, en définissant un sous-langage (inclus dans le standard ou spécifique à un projet), disposant d'un métamodèle dont les classes sont des sous-classes du modèle AADL. Ce point reste à

vérifier dans notre cas. Concernant l'outillage, l'outil OSATE développé par le SEI est un logiciel libre intégré dans la plateforme TOPCASED basée sur Eclipse, tout comme Papyrus. L'intégration des algorithmes que nous avons proposé ne devrait donc pas poser de problèmes.

5.2.2. Evolution de l'outillage et de la méthode

Concernant la méthode et l'outillage, la génération des solutions pourrait être intégrée dans le modèle initial. Cette implémentation est en cours, concernant l'allocation variable. Actuellement, l'outillage est constitué d'un démonstrateur. L'utilisateur doit choisir lui-même le solveur, parmi les trois disponibles (Labix/PyOpt/CHOCO). Il serait intéressant de rajouter un assistant pour le guider dans ce choix, en fonction des points de décision présent dans son modèle, comme nous l'avons vu dans la conclusion du chapitre 4. Dans le démonstrateur, le code des fonctions objectifs est intégré dans le modèle du système. Pour ces fonctions objectifs, Il serait également intéressant de pouvoir interfacer les solveurs avec des librairies provenant d'autre outils, comme Matlab/Simulink par exemple. L'intégration d'un autre solveur, si elle s'avère nécessaire, ne devrait pas poser de problème si l'on reste dans le cadre de la plateforme Eclipse/Papyrus et avec des langages comme JAVA ou Python pour le solveur. La mise en œuvre de la méthode avec un autre outil SysML comme Capella et de la méthode ARCADIA (Voirin, et al., 2015) est également envisageable, bien que cet outil ne soit pas dans le domaine du logiciel libre.

5.2.3. Applications à d'autres cas d'études

Concernant l'application de la méthode, deux nouvelles applications sont envisagées. La première concerne l'optimisation d'un système avionique de portes de cabine. Il s'agit d'optimiser le coût et la fiabilité du système, ainsi que la masse constituée par le câblage reliant les composants avioniques. Le système est constitué de composants sélectionnés dans une librairie, chaque composant étant affecté à l'une des baies de l'avion, dont la position est connue à l'avance. Notre méthode est applicable, en utilisant les points de décisions d'instance et d'allocation en même temps, ce que nous n'avons pas encore fait. Il faut également modéliser les fonctions objectifs « fiabilité » et « masse » en prenant en compte la longueur des câbles. Ce cas d'étude est en cours de développement, les résultats seront disponibles prochainement.

La seconde application concerne un véhicule autonome développé dans le cadre de l'institut VEDECOM. Le véhicule dispose de multiples capteurs de position, notamment des caméras, LIDAR et radars longue portée. Il s'agit de déterminer la configuration optimale permettant un fonctionnement du véhicule en mode dégradé (basse vitesse) en cas de panne d'un capteur ou de condition conduite difficile. Pour cela on souhaite déterminer le nombre de capteurs de chaque type (redondance) ainsi que les références à utiliser parmi un ensemble de capteurs disponibles. Le coût de l'ensemble doit également être minimisé.

Bibliographie

Abele A., Lönn, H., Reiser, M. O., Weber, M., and Glathe, H. 2012. "EPM: a prototype tool for variability management in component hierarchies," in Proceedings of the 16th International Software Product Line Conference. 2012, Vol. 2, pp. 246-249.

Aleti, A., Bjornander, S., Grunske, L., and Meedeniya, I. 2009. "ArcheOpterix: An extendable tool for architecture optimization of AADL models,". in Proceedings of the 31st International Conference on Software Engineering. 2009, pp. 61-71.

Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., & Meedeniya, I. 2013. "Software architecture optimization methods: A systematic literature review,". *IEEE Transactions on Software Engineering.* 2013, Vol. 39, pp. 658-683.

Apt, K. R., & Wallace, M. 2006. *"logic programming using ECLiPSe,".* s.l. : Cambridge University Press, 2006.

Berger, N. 2010. *"Modélisation et résolution en programmation par contraintes de problèmes mixtes continu/discret de satisfaction de contraintes et d'optimisation,".* Université de Nantes : Thèse de doctorat, 2010.

Bessière, C., Freuder, E. C., and Régin, J. C. 1999. "Using constraint metaknowledge to reduce arc consistency computation,". *Artificial intelligence.* 1999, Vol. 107, pp. 125-148.

Bessière, C., Meseguer, P., Freuder, E. C., and Larrosa, J. 2002. "On forward checking for non-binary constraint satisfaction,". *Artificial Intelligence.* 2002, Vol. 141, pp. 205-224.

Bettini, L. 2013. *"Implementing Domain-Specific Languages with Xtext and Xtend,".* s.l. : Packt Publishing Ltd, 2013.

Brailsford, S. C., Potts, C. N., and Smith, B. M. 1999. "Constraint satisfaction problems: Algorithms and applications,". *European Journal of Operational Research.* 1999, Vol. 119, pp. 557-581.

Broodney, H., Dotan, D., Greenberg, L.,. 2012. "Generic approach for systems design optimization in MBSE,". *In : The 22nd Annual INCOSE International Symposium.* 2012.

Bruel, J. M., & Roques, P. "Presentation of SysML Concepts". *Embedded Systems.* pp. 45-69.

—. 2013. Embedded Systems. [auteur du livre] F., Hugues, J., Canals, A., & Dohet, A. Kordon. *Embedded Systems: Analysis and Modeling with SysML, UML and AADL.* s.l. : John Wiley & Sons, 2013, pp. 45-69.

Bulfin, R. L., & Liu, C. Y. 1985. "Optimal allocation of redundant components for large systems,". *IEEE Transactions on Reliability.* 1985, Vol. 34, 3, pp. 241-247.

Chankong, V., and Haimes, Y. 1978. "The interactive surrogate worth trade-off (ISWT) method for multiobjective decision-making,". *Multiple criteria problem solving.* s.l. : Springer Berlin Heidelberg, 1978, pp. 42-67.

Charbonnier, P., Muzet, V., Nicolle, P., Hautiére, P., Tarel, J. P., & Aubert, D. 2008. "Stereovision applied to road scene analysis". *Bulletin des Laboratoires des Ponts et Chaussées.* 2008, Vol. 76.

Chern, MS. 1992. "On the computational complexity of reliability redundancy allocation in a series system,". *Operations research letters.* 1992, Vol. 11, 5, pp. 309-315.

CHOCO. 2016. "Choco solver". [Online] EMN, 2016. http://choco-solver.org/..

Clarity. 2016. "Ecosystem for the MBSE solution Capella". [En ligne] 2016. www.clarity-se.org.

Coit, D. W., & Smith, A. E. 1995. *"Optimization approaches to the redundancy allocation problem for series-parallel systems," in Proceedings of Fourth Industrial Engineering Research Conference Proceedings.* 1995. pp. 342-349.

Collette, Y., & Siarry. 2013. *"Multiobjective optimization: principles and case studies,".* s.l. : Springer, 2013.

Czarnecki, K., & Kim, C. H. P. 2005. "Cardinality-based feature modeling and constraints : A progress report," In International Workshop on Software Factories. 2005, pp. 16–20.

De Saqui-Sannes, P., & Hugues, J. 2012. "Combining SysML and AADL for the design, validation and implementation of critical systems" in Proceedings of ERTS2. 2012, p. 117.

Desfray, P. 2012. "Etat de l'art, SysML où en est-on". [En ligne] 2012. http://www.miam.mips.uha.fr/sysml-uha/slides_desfray.pdf.

Dufay, F. 2010. *"CMMI par l'exemple : Pour une mise en place opérationnelle".* s.l. : Eyrolles, 2010.

EAST-ADL. 2016. *EAST-ADL Association.* [En ligne] Juin 2016. http://www.east-adl.info/index.html.

Eckert, A., Hohm, A., & Lueke, S. 2013. "An integrated ADAS solution for pedestrian collision avoidance," in Proceedings of the 23rd International Conference on the Enhanced Safety of Vehicles. Seoul, Republic of Korea : s.n., 2013, pp. 13-0298.

Eclipse. 2016. "Eclipse Modeling Framework". [En ligne] 2016. http://www.eclipse.org/modeling/emf/.

Ertl, D., & Krapfenbauer, H. 2009. "A Case Study of Developing an IDE for Embedded Software Using Open Source," in Proceedings of ICSEA'09. s.l. : IEEE, 2009, pp. 191-196.

Espinoza, H., Cancila, D., Selic, B., & Gérard, S. 2009. "Challenges in combining SysML and MARTE for model-based design of embedded systems." In : European Conference on Model Driven Architecture-Foundations and Applications. [éd.] Springer Berlin Heidelberg. s.l. : Springer Berlin Heidelberg, 2009, pp. 98-113.

Feiler, Peter H., David P. Gluch, and John J. Hudak. 2006. *"The architecture analysis & design language (AADL): An introduction.".* s.l. : Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst., 2006. No. CMU/SEI-2006-TN-011.

Fontoura, M., Pree, W., and Rumpe, B. 2001. *"The Uml Profile for Framework Architectures".* s.l. : Addison-Wesley, 2001.

Fourer, R., Gay, D. M., & Kernighan, B. W. 1990. "A modeling language for mathematical programming". *Management Science.* 1990, Vol. 36, pp. 519-554.

Friedenthal, S., Griego, R., and Sampson, M. 2007. *"INCOSE model based systems engineering (MBSE) initiative" In INCOSE 2007 Symposium.* 2007.

Friedenthal, S., Moore, A., & Steiner, R. 2014. *"A practical guide to SysML : the system modeling language".* s.l. : Morgan Kaufman, 2014.

Galster, M., Weyns, D., Tofan, D., Michalik, B., & Avgeriou, P. 2014. "Variability in software systems – a systematic literature review,". *IEEE Transactions on Software Engineering.* 2014, Vol. 40, 3, pp. 282–306.

Gent, I. P., Jefferson, C., & Miguel, I. 2006. *"Minion: A fast scalable constraint solver" In ECAI.* 2006. pp. 98-102.

Goualard, F. 2000. *"Langages et environnements en programmation par contraintes d'intervalles".* Doctoral dissertation, Université d'Orléans : s.n., 2000.

Granvilliers, L., & Benhamou, F. 2006. "Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques". *ACM Transactions on Mathematical Software (TOMS).* 2006, Vol. 32, 1, pp. p. 138-156.

Greenhalgh, P. 2011. *"Big. little processing with arm cortex-a15 & cortex-a7,".* s.l. : ARM White paper, 1-8., 2011.

Hachimi, H. 2013. *"Hybridations d'algorithmes métaheuristiques en optimisation globale et leurs applications" Thèse de doctorat. INSA de Rouen; École Mohammadia d'ingénieurs (Rabat, Maroc).* 2013.

Haugen, Ø., Møller-Pedersen, B., Oldevik, J., Olsen, G. K., & Svendsen, A. 2008. "Adding standardized variability to domain specific languages," in Proceedings of : Software Product Line Conference. 2008, pp. 139–148.

Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., & Ernst, R. 2005. "System level performance analysis-the SymTA/S approach, ". *IEE Proceedings-Computers and Digital Techniques.* 2005, Vol. 152, 2, pp. 148-166.

How, J. P., Fraser, C., Kulling, K. C., & Bertuccelli, L. F. 2009. "Increasing autonomy of UAVs". *Robotics &Automation Magazine.* Institute of Electrical and Electronics Engineers, 2009, Vol. 16, 2, pp. 43-51.

Hugues, J., Zalila, B., Pautet, L., & Kordon, F. 2008. "From the prototype to the final embedded system using the Ocarina AADL tool suite". *ACM Transactions on Embedded Computing Systems (TECS).* 2008, Vol. 7, 4, p. 42.

I.B. Jaâfar, N. Khayati, and K. Ghédira. 2004. "Multicriteria optimization in CSPs: Foundations and distributed solving approach". *Artificial Intelligence: Methodology, Systems, and Applications.* 2004, pp. (pp. 459-468).

INCOSE. 2007. *"Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities V3.1".* s.l. : Cecilia Haskins, CSEP, 2007.

ISO-15288. 2008. *"ISO/ISE/IEEE 15288 - Systems and software engineering - System life cycle processes".* New York : IEEE, 2008.

ISO-42010. 2011. *"ISO 42010- System and software engineering - Architecture Description".* New York : s.n., 2011.

Jussien, N., Rochart, G., & Lorca, X. 2008. "Choco: an open source java constraint programming library," CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08). 2008, pp. 1-10.

Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, A.,. 1990. *"Feature-Oriented Domain Analysis (Foda) Feasibility Study,".* s.l. : Software Engineering Institute, Carnegie Mellon University, 1990.

Kerzhner, R. A., Paredis, C. J., Amaral, V., Vangheluwe, H., Lengyel, L., Margaria, T. 2011. "Combining SysML and model transformations to support systems engineering analysis,", in Proceedings of the 4th International Workshop on Multi-Paradigm Modeling. 2011, Vol. 42.

Khan, T. H., & Wahid, K. 2011. "Design of a DVP compatible bridge to randomly access pixels of high speed image sensors," in Proceedings of IEEE International Conference on Consumer Electronics (ICCE). 2011.

KUMAR, Rakesh, FARKAS, Keith I., JOUPPI, Norman P., et al. 2003. "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction," in Proceedings of 36th Annual IEEE/ACM International Symposium on. s.l. : IEEE, 2003, pp. 81-92.

Laganière, R. 2011. *"OpenCV 2 Computer Vision Application Programming Cookbook: Over 50 recipes to master this library of programming functions for real-time computer vision,".* s.l. : Packt Publishing Ltd, 2011.

Leserf, P., Saqui-Sannes, P. D., Hugues, J., & Chaaban, K. Déc. 2015. "Architecture Optimization with SysML Modeling A Case Study Using Variability". [auteur du livre] P. Desfray. *MDE and software developement - CCIS.* s.l. : Springer, Déc. 2015, Vol. 580, 18.

Leserf, P., Saqui-Sannes, P. D., Hugues, J., and Chaaban, K. Fev. 2015. "SysML Modeling For Embedded Systems Design Optimization: A Case Study," in Proceedings of ModelsWard. s.l. : IEEE, Fev. 2015, pp. 449-457.

Leserf, P., Saqui-Sannes, P. D., Hugues. Sept. 2015. "Multi-Domain optimization with SysML modeling," in Proceedings of ETFA 20th IEEE conference. Luxembourg : IEEE, Sept. 2015, pp. 1-8.

Limbourg, P., & Kochs, H. D. 2008. "Multi-objective optimization of generalized reliability design problems using feature models—A concept for early design stages". *Reliability Engineering & System Safety.* 2008, Vol. 93, 6, pp. 815-828.

Liu, Y., Gao, J., & Liu, D. 2011. *"Design of Software System of Mobile Robot with Mixed Programming Based on Eclipse+ pydev" in Theoretical and Mathematical Foundations of Computer Science.* Berlin : Springer Berlin Heidelberg, 2011. pp. 231-238.

MARTE. 2011. "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems V1.1". [En ligne] 2011. http://www.omg.org/spec/MARTE/1.1.

Masin, M., Broodney, H., Brown, C., Limonad, L., Mashkif, N., & Sela A. 2014. "Reusable derivation of operational metrics for architectural optimization,". *Procedia Computer Science.* 2014, Vol. 28, pp. 481-488.

Mathews, J. H., & Fink, K. D. 1999. *"Numerical methods using MATLAB".* s.l. : Prentice hall Upper Saddle River, 1999.

Meedeniya, I., Buhnova, B., Aleti, A., & Grunske, L. 2011. "Reliability-driven deployment optimization for embedded systems". *Journal of Systems and Software.* 2011, Vol. 84, 5, pp. 835-846.

Min, B. I., Kerzhner, A. A., & Paredis, C. J. 2011. "Process integration and design optimization for model-based systems engineering with SysML," In ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. s.l. : American Society of Mechanical Engineers, 2011, pp. 1361-1369.

Moore, R. E., & Bierbaum, F. 1979. *"Methods and applications of interval analysis".* Philadelphia : Siam, 1979.

Mraidha, C., Tanguy, Y., Jouvray, C., Terrier, F., & Gérard, S. 2008. "An execution framework for MARTE-based models" in Proceedings of Engineering of Complex Computer Systems. 2008, pp. 222-227.

Niemeyer, G. 2016. *"Labix - python-constraint".* [En ligne] 2016. https://labix.org/python-constraint.

Older, W., & Vellino, A. 1990. "Extending prolog with constraint arithmetic on real intervals," in Proceedings of the Eight Biennial Conference of the Canadian Society for Computational Studies of Intelligence. 1990.

Padoin, E. L., Pilla, L. L., Castro, M., Boito, F. Z., Navaux, P. O. A., & Méhaut, J. F. 2015. "Performance/energy trade-off in scientific computing: the case of arm big. little and intel sandy bridge,". *IET Computers & Digital Techniques.* 2015, Vol. 9, 1, pp. 27-35.

Papyrus. 2015. "Outil Papyrus". [En ligne] CEA, 2015. https://eclipse.org/papyrus/.

Paret, D. 2012. *"FlexRay and its applications: real time multiplexed network".* s.l. : John Wiley & Sons, 2012.

—. 2007. *"Multiplexed networks for embedded systems: CAN, LIN, Flexray, Safe-by-Wire...".* s.l. : John Wiley & Sons, 2007.

Perez, R. E., Jansen, P. W., & Martins, J. R. 2012. "pyOpt: a Python-based object-oriented framework for nonlinear constrained optimization. Structural and Multidisciplinary Optimization". *Structural and Multidisciplinary Optimization.* 2012, Vol. 45, 1, pp. 101-118.

Pohl, K., Böckle, G., & van Der Linden, F. J. 2005. *"Software product line engineering : Foundations, principles and techniques.".* s.l. : Springer, 2005.

Régin, J. C. 1994. "A filtering algorithm for constraints of difference in CSPs,". *AAAI.* 1994, Vol. 94, pp. 362-367.

Reinhartz-Berger, I., & Figl, K. 2014. "Comprehensibility of orthogonal variability modeling languages: the cases of CVL and OVM," in Proceedings of the 18th International Software Product Line Conference. s.l. : ACM, 2014, Vol. 1, pp. 42-51.

Roques, P. 2013. *"Modélisation des systèmes complexes avec SysML".* s.l. : Eyrolles, 2013.

Rossi, F., Van Beek, P., & Walsh, T. 2006. *Handbook of constraint programming.* s.l. : Elsevier, 2006.

Rouillé, E. 2014. *"Gestion de la variabilité et automatisation des processus de développement logiciel,".* Université Rennes 1. 2014. Thèse de doctorat.

Rumbaugh, J., Jacobson, I., & Booch, G. 2004. *"Unified Modeling Language Reference Manual,".* s.l. : Pearson Higher Education, 2004.

Sabin, D., & Ereuder, E. C. 1997. "Understanding and improving the MAC algorithm," in Proceedings of International Conference on Principles and Practice of Constraint Programming. s.l. : Springer, 1997, pp. 167-181.

Samih, H. *"Test basé sur les modèles appliqué aux lignes de produits", Doctoral dissertation.* Rennes 1 : s.n.

Sélic, B., Gérard, S. 2014. *"Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE".* s.l. : Morgan Kaufmann, 2014.

Spyropoulos, D., & Baras, J. S. 2013. "Extending Design Capabilities of SysML with Trade-off Analysis: Electrical Microgrid Case Study". *Procedia Computer Science.* 2013, Vol. 16, pp. 108-117.

Svahnberg, M., Van Gurp, J., & Bosch, J. 2005. "A taxonomy of variability realization techniques,". *Software: Practice and Experience.* 2005, Vol. 35, 8, pp. 705–754.

SysML. 2015. "OMG Systems Modeling Language (OMG SysML™) V1.4". [En ligne] 2015. http://www.omg.org/spec/SysML/1.4/.

UML2. 2007. "OMG. OMG Unified Modeling Language (OMG UML™) V2.5". [En ligne] 2007. http://www.omg.org/spec/UML/2.5.

Van Hentenryck, P., McAllester, D., & Kapur, D. 1997. "Solving polynomial systems using a branch and prune approach". *Journal on Numerical Analysis.* 1997, Vol. 34, 2, pp. 797-827.

Van Huong, P., & Binh, N. N. 2012. "Embedded system architecture design and optimization at the model level". *International Journal of Computer and Communication Engineering.* 2012, Vol. 1, 4, p. 345.

Van Rossum, G., & Drake, F. L. 2011. *"The python language reference manual".* s.l. : Network Theory Ltd.,, 2011.

Voirin, J. L., Bonnet, S. et Exertier, D. 2015. "Document d'introduction à la méthode ARCADIA". *Polarsys.* [En ligne] 01 January 2015. http://download.polarsys.org/capella/publis/An_Introduction_to_Arcadia_20150115.pdf.

Walker, M., Reiser, M. O., Tucci-Piergiovanni, S., Papadopoulos, Y., Lönn, H., Mraidha, C., & Servat, D. 2013. "Automatic optimisation of system architectures using EAST-ADL". *Journal of Systems and Software.* 2013, Vol. 86, 10, pp. 2467-2487.

Weatherington, D., & Deputy, U. 2005. "Unmanned aircraft systems roadmap, 2005-2030". s.l. : Deputy, UAV Planning Task Force, OUSD (AT&L), 2005.

Whittle, J., Hutchinson, J., Rouncefield, M., Burden, H., & Heldal, R. 2013. "Industrial adoption of model-driven engineering: Are the tools really the problem?," in Proceedings of

International Conference on Model Driven Engineering Languages and Systems. s.l. : Springer, 2013, pp. 1-17.

Wirth, N. 1996. *"Extended Backus-Naur Form (EBNF)" ISO/IEC .* 1996. p. 2996. Vol. 14977.

Wymore, A. W. 1993. *"Model-based systems engineering".* s.l. : CRC Press, 1993.

Ziadi, T., Hélouët, L., & Jézéquel, J. M. 2004. "Towards a uml profile for software product lines in Software Product-Family," In International Workshop on Software Product-Family Engineering. s.l. : Springer, 2004, Vol. 129–139.

Architecture Optimization of Embedded Systems with a Model Based Approach

Abstract:

Finding the set of optimal architectures is an important challenge for the designer who uses the Model-Based System Engineering (MBSE). Design objectives such as cost, performance are often conflicting. Current methods (OOSEM with SysML or ARCADIA) are focused on the design and the analysis of a particular alternative of the system. In these methods, the topology and the execution platform are frozen before the optimization. To improve the optimization from MBSE, we propose a methodology combining SysML with the concept of “decision point”. An initial SysML model is complemented with “decisions points” to show up the different alternatives for component redundancy, instance selection and allocation. The constraints and objective functions are also added to the initial SysML model, with an optimization context and parametric diagram. Then a representation of a constraint satisfaction problem for optimization (CSMOP) is generated with an algorithm and solved with an existing solver. A demonstrator implements this transformation in an Eclipse plug-in, combining the Papyrus open-source tool and CSP solvers. Two case studies illustrate the methodology: a stereoscopic camera sensor module and a mission controller for an Unmanned Aerial Vehicle (UAV).

Keywords: MBSE, Optimization, SysML, CSP, Papyrus, System engineering, Optimal architecture design, Embedded systems, Component choice, Redundancy, Allocation, Cost, Performance, Reliability.

AUTEUR : Patrick LESERF

TITRE : Optimisation de l'architecture de systèmes embarqués par une approche basée modèle

DIRECTEURS DE THESE : Pierre de SAQUI-SANNES et Jérôme HUGUES

LIEU ET DATE DE SOUTENANCE : ISAE-SUPAERO, 2 mai 2017

RESUME

L'analyse de compromis d'un modèle système a pour but de minimiser ou de maximiser différents objectifs tels que le coût ou les performances. Les méthodes actuelles de type OOSEM avec SysML ou ARCADIA sont basées sur la classification ; il s'agit de définir les différentes variantes de l'architecture d'un système de base puis d'analyser ces variantes. Dans ces approches, les choix d'architecture sont contraints : la plateforme d'exécution et la topologie sont déjà figées. Nous proposons la notion de « points de décision » pour modéliser les différents choix du système, en utilisant de nouveaux stéréotypes. L'avantage est d'avoir une modélisation plus « compacte » des différentes variantes et de piloter l'exploration des variantes en utilisant des contraintes. Lorsque le concepteur définit l'architecture du système, des points de décisions sont insérés dans le modèle du système. Ils permettent de modéliser la redondance ou le choix d'une instance pour un composant, les variations des attributs d'un composant, ou l'allocation des activités sur les blocs. Les fonctions objectifs sont définies dans un contexte d'optimisation à l'aide du diagramme paramétrique de SysML. Nous proposons des transformations du modèle SysML vers un problème de satisfaction de contraintes pour l'optimisation (CSMOP) dont la résolution nous permet d'obtenir l'ensemble des architectures optimales. Cette transformation est implantée dans un démonstrateur (plug-in Eclipse) permettant une utilisation conjointe de l'outil Papyrus et de solveurs, disponibles sous forme de logiciels libres. La méthode est illustrée avec des cas d'étude constitués d'une caméra stéréoscopique puis d'un drone, l'ensemble étant modélisé avec Papyrus.

MOTS-CLEFS

IDM, MBSE, Optimisation, SysML, CSP, Papyrus, Ingénierie système, Conception d'architecture optimale, Systèmes embarqués, Choix de composants, Redondance, Allocation, Coût, Performance, Fiabilité.

DISCIPLINE : Informatique et Télécommunications

ISAE SUPAERO - Institut Supérieur de l'Aéronautique et de l'Espace
10 avenue Edouard Belin - BP 54032 - 31055 TOULOUSE CEDEX 4 FRANCE