# Un logiciel orienté objet d'analyse et de conception de plaques composites

**R. Le Riche**[a]**, J. Gaudin**[b]**, A. Vinet**[c]

[a] SMS, URA 1884, Ecole des Mines de Saint Etienne,
158, cours Fauriel, 42023 Saint Etienne Cedex 2
leriche@emse.fr
[b] EADS CIMPA, 92 route de la Reine, 92100 Boulogne Billancourt
jocelyn.gaudin@eads.net
[c] EADS CCR, 12 rue Pasteur, 92152 Suresnes Cedex
alain.vinet@airbus.com

**(Proc. of the 6th Colloque en Calcul de Structures, Vol.1, CSMA / AFM publ., Giens, France, May 2003, pp. 87-94)**

**Résumé**

Les travaux réalisés en calcul de structure induisent une croissance constante du nombre de modèles et de méthodes de conception à considérer. L'architecture du logiciel liant modèles et optimiseurs est cruciale pour capitaliser ces connaissances et ne pas brider la conception. Le logiciel LAMKIT fédère un ensemble de modèles semi-analytiques d'analyse de stratifiés composite et des méthodes d'optimisations. Le modèle d'interface orientée objet entre la simulation et l'optimisation mis en œuvre dans LAMKIT est décrit.

## 1 Introduction

Les progrès réalisés dans les domaines de l'analyse de structures et de l'optimisation accroissent en permanence la complexité de la conception qui, s'attachant à résoudre des problématiques plus complètes (conception en présence d'incertitudes, sélection de matériaux et dimensionnement simultanés, ...), doit à la fois prendre en compte davantage de modèles physiques et davantage d'algorithmes d'optimisation. L'architecture du programme qui met en œuvre les modèles structuraux et les algorithmes d'optimisation devient aujourd'hui cruciale pour capitaliser ces connaissances, sans que les possibilités de la conception ne soient limitées par le poids de la programmation. Dans le domaine des éléments finis, la complexité logicielle a été reconnue comme une pierre d'achoppement il y a plus de 10 ans et des solutions par programmation orientée objet (POO) proposées (e.g., [2]). L'apport de la programmation orientée objet pour la conception est plus récent ([4]).

Le logiciel LAMKIT ([3]) est présenté où une attention particulière est portée à l'interface entre les modules d'analyse et les méthodes d'optimisation. Ce logiciel fédère un ensemble de modèles analytiques ou semi-analytiques de plaques composites et des optimiseurs permettant des formulations diverses de problèmes de conception. Un exemple de problème couplant sélection de matériaux et dimensionnement est donné.

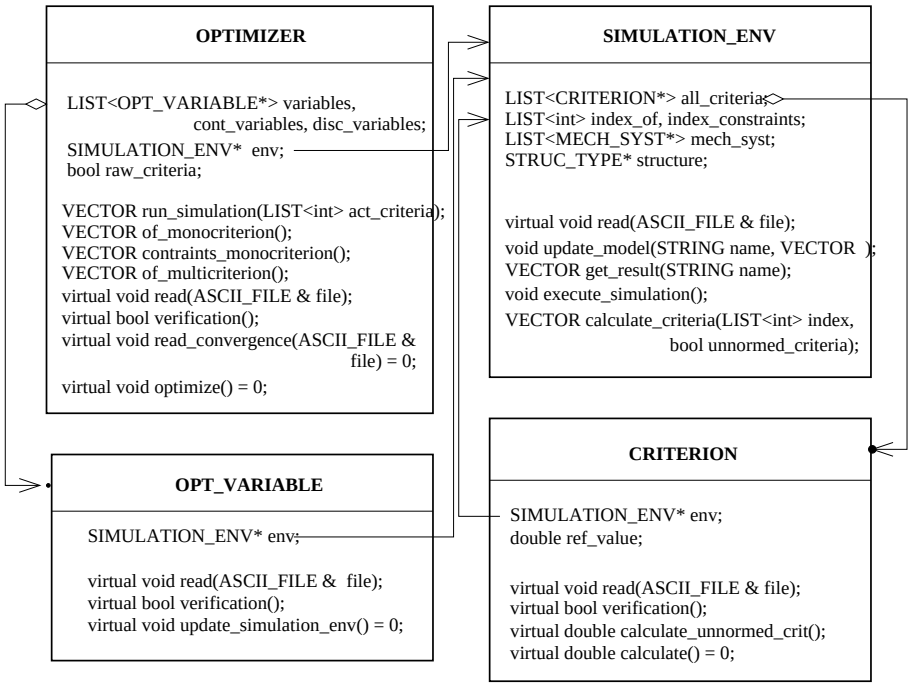


FIG. 1 – Vue d'ensemble de l'interface simulation-optimisation.

# 2 Interface simulation-optimisation

## 2.1 Une interface interne

Un programme de modélisation peut être interfacé avec l'optimisation de manière interne ou externe.

Dans le cas d'une interface externe, au moins deux programmes exécutables (la simulation et l'optimiseur) dialoguent par fichiers ou par signaux. Une interface externe requiert typiquement un sous-programme de traduction du vecteur des variables d'optimisation vers le fichier de mise en données de la simulation, et réciproquement, des fichiers résultats de simulation vers les fonctions objectifs et contraintes de l'optimisation. L'interface externe est utilisée dans les projets où l'optimisation n'a pas été prévue au départ. Ses inconvénients sont que le traducteur doit être ré-écrit à chaque nouveau modèle et que la lecture des données doit être intégralement recommencée à chaque analyse, même si les variables d'optimisation affectent une faible portion de l'analyse, ce qui représente une perte de ressources informatiques. Il faut néanmoins signaler que des logiciels utilisant des interfaces externes proposent des solutions ergonomiques pour le traducteur (les "templates" dans [8] et les classes utilitaires `IOFilter` dans [4]).

D'autres programmes contenant des possibilités de simulation et d'optimisation (e.g., [7])

ont recours à des interfaces internes permettant une gestion sur-mesure de la mise en mémoire des données. L'inconvénient potentiel d'une interface interne est de trop brider les possibilités de formulations de problèmes de conception en définissant de manière non-évolutive variables et critères d'optimisation.

## 2.2 Style de programmation

Contrairement aux langages de programmation séquentiels (FORTRAN, C, BASIC, PASCAL, ...), les langages orientés objet tels que le C++ ([9]) permettent de créer des types de variables et leurs fonctionnalités (objets). L'utilisation d'objets spécifiques au problème traité résulte en des expressions plus directes que les langages séquentiels.

Les descriptions de programme qui suivent utilisent des notions de programmation orientée objet (POO), du pseudo-C++, mais de nombreux détails du langage sont omis. En effet, l'article vise à décrire un modèle de programmation (ou "pattern", cf. [5]), c'est à dire comment des objets interagissent, sans entrer dans le détail de comment chaque objet est programmé.

Le style de programmation de LAMKIT s'inspire des travaux présentés dans [2]. La composition d'objets (un objet contenant un pointeur vers un autre objet) constitue le modèle le plus souvent employé pour traduire une relation "l'objet A a un objet B". La relation "est un" est, classiquement, obtenue par héritage simple. Ces deux modèles de programmation permettent d'éviter, dans la plupart des situations, l'héritage multiple, ce qui est souhaitable. Une technique de "fabrique d'objets" ("object factory", cf. [5]) vient s'ajouter à la composition d'objets et rend l'installation des objets entièrement dynamique dans le programme. Des classes utilitaires telles que des conteneurs génériques (`LIST<>`), des classes mathématiques (`VECTOR`), ..., sont employées par la suite sans autre forme de commentaires.

## 2.3 Un modèle d'interface simulation-optimisation

Les ingrédients de tout problème d'optimisation, à savoir un algorithme d'optimisation, des variables $x$ et des critères (les fonctions objectifs $f$ et les contraintes $g$) constituent les classes de base de l'interface, `OPTIMIZER`, `OPT_VARIABLE` et `CRITERION`, respectivement. Une classe supplémentaire, `SIMULATION_ENV`, contient l'environnement de simulation. Une vue d'ensemble de l'interface est donnée sur la figure 1.

`OPTIMIZER` a deux rôles : il contient les algorithmes d'optimisation, et il fournit des variables et fonctions utilisées par la plupart des optimiseurs. `OPTIMIZER` est la classe de base d'optimiseurs évolutionnaire mono et multi-critères en variables mixtes ([1, 6]) ainsi que d'un algorithme d'étude paramétrique. `OPTIMIZER` définit deux fonctions abstraites pures (qui doivent être spécifiées dans les classes dérivées), `optimize()` et `read_convergence()`. `OPTIMIZER` permet donc de rendre le logiciel indépendant des algorithmes d'optimisation particuliers. L'autre rôle d'`OPTIMIZER` est de contenir et d'instancier les variables d'optimisation, et de fournir des fonctions pour calculer une ou plusieurs fonctions objectifs (cas mono et multi-critères) et des contraintes.

`OPT_VARIABLE` est la classe de base des variables d'optimisation discrètes et continues. Sa première responsabilité est de factoriser les fonctions communes à toutes les variables de même nature mathématique. Par exemple, toutes les variables continues ont une valeur de référence et des fonctions de normalisation et dénormalisation. La seconde responsabilité

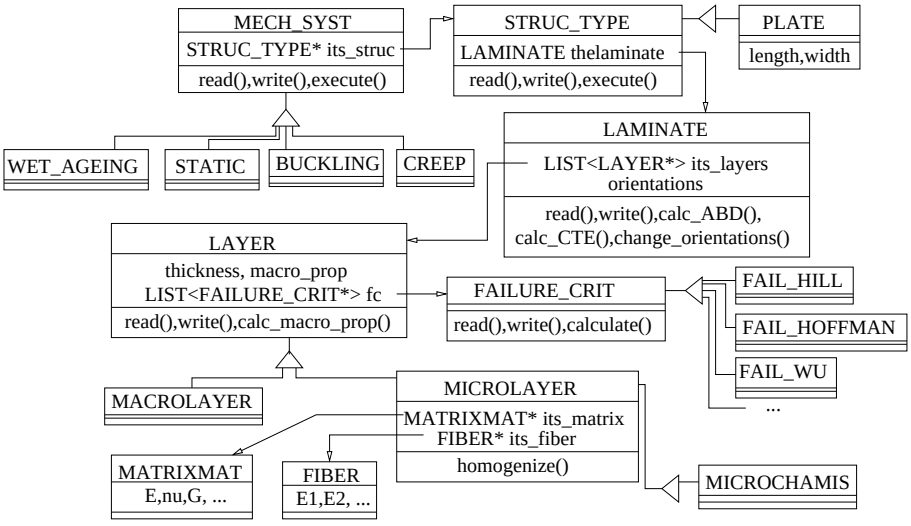


FIG. 2 – Vue d'ensemble des classes de la simulation.

d'`OPT_VARIABLE` est de définir, dans la fonction abstraite pure `update_simulation_env()` comment une variable d'optimisation affecte un module de simulation. Les modules de simulation sont accessibles par l'intermédiare d'un pointeur vers l'environnement de simulation, `env` (cf. figure 1). En termes de POO, `OPT_VARIABLE` est un adaptateur entre `OPTIMIZER` et `SIMULATION_ENV` car il joint ces deux classes dont les interfaces sont a priori incompatibles. Des exemples typiques de classes dérivées d'`OPT_VARIABLE` dans LAMKIT, que nous reverrons en Section 3, sont les orientations de plis (discrètes ou continues), les matériaux ou les nombres de plis (discrets).

`CRITERION` est la classe de base des critères d'optimisation, que ce soient les fonctions objectifs, $\min_x f(x)$ ou les contraintes, $g(x) \leq 0$. Son rôle est d'une part, d'aller chercher des quantitées dans le module de simulation (en ce sens, c'est aussi un adaptateur) et, d'autre part, de réaliser le calcul du critère dans la fonction `calculate()`. Par exemple, le critère de déformation longitudinale `EPSX_CRITERION`, qui est définit comme $|\varepsilon_x|/\varepsilon_x^{ref} - 1$, est écrit,

```
double EPSX_CRITERION::calculate()
{  VECTOR v = env->get_result("epsx");
   return fabs( v[0]/ref_value-1. ; }
```

On note qu'un critère peut être calculé sous forme normalisée ou non (fonction `calculate_unnormed_crit()`). Une cinquantaine de critères existent dans LAMKIT concernant les coefficients de dilatations thermiques et hydriques, les rigidités, les critères de ruptures (déformations ou contraintes principales, Tsai-Wu, Hoffman, Tsai-Hill), le flambement, l'épaisseur, ainsi que les écarts types des critères sus-cités.

`SIMULATION_ENV` est une classe de l'interface assurant deux responsabilités. Tout d'abord, elle tient lieu de "façade" à la simulation, c'est à dire qu'elle est une interface simple entre

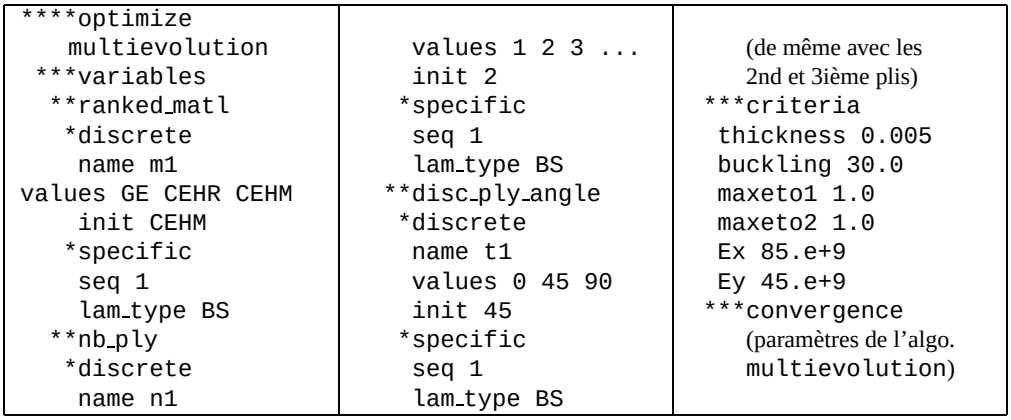


FIG. 3 – Fichier de mise en données pour optimisation multi-critères.

les optimiseurs et les nombreux et changeants modules de simulation. Un accès paramétré à toutes les classes constituant la simulation est donné par les fonctions `get_result (STRING )` et `update_model(STRING, VECTOR )`. Ces deux fonctions utilisent un paramétrage de la simulation programmé au moyen de classes comme, par exemple pour les paramètres réels, `IO_DBL_PARAMETER`,

```
    class IO_DBL_PARAMETER  {
      protected :
        static LIST<STRING> all_names;
        static LIST<IO_PARAMETER*> all_params;
        STRING name;
        double value; ...
      public :
        IO_DBL_PARAMETER(double );
        operator double();
        double get(STRING name_parameter);
        void set(STRING name_parameter, double new_value);
... };
```

Les listes statiques de noms et de pointeurs vers des paramètres permettent d'accéder à n'importe quel paramètre n'importe où dans le programme au moyen d'une instance de classe cherchant son pointeur au moyen de son nom. Ces opérations sont réalisées dans les fonctions `set(STRING )` et `get(STRING, double)`. Le constructeur à partir de `double` et l'opérateur `double` rendent un `double` et un `IO_DBL_PARAMETER` interchangeables. Il suffit donc de remplacer un `double` dans la simulation par un `IO_DBL_PARAMETER` pour qu'il soit accessible depuis `SIMULATION_ENV`. `SIMULATION_ENV` contient en outre des pointeurs vers les classes principales de la simulation qui sont la structure (géométrie et matériau, classe `STRUC_TYPE`) et l'analyse mécanique réalisée sur la structure, `MECH_SYST`. La figure 2 schématise comment, à partir de ces deux classes, les autres modules d'analyse de stratifiés de LAMKIT sont organisés.

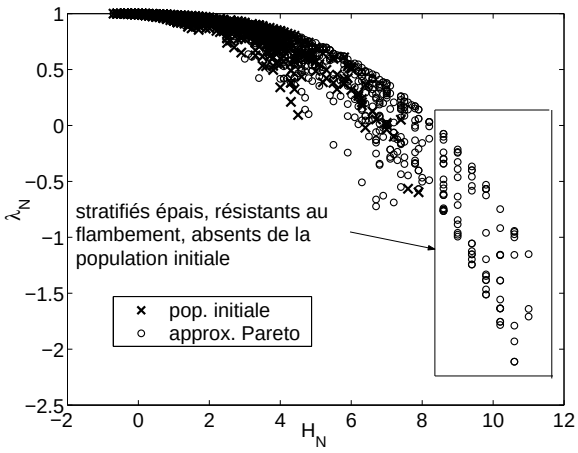


FIG. 4 – Projection du front de Pareto approché pour le problème de choix de matériaux. $H_N$ et $\lambda_N$ sont les épaisseurs et charges critiques de flambement normalisées, $H_N = (\text{épaisseur}/0.005 - 1)$, $\lambda_N = (1 - \lambda/30)$.

En même temps qu'elle est une façade, `SIMULATION_ENV` gère la mémoire et les calculs de simulation : lecture des données (dans `read(ASCII_FILE)`), mise à jour de la simulation pendant les itérations d'optimisation (fonction `update_model(STRING, VECTOR )`), exécution de la simulation puis calcul des critères au moyen de la liste de pointeurs `all_criteria`.

Grâce à ce modèle d'interface, de nombreux problèmes de conception peuvent être traités car les variables sont continues, discrètes ou mixtes et une grande variété de critères sont disponibles. De plus, le style de programmation permet d'ajouter variables, critères et algorithmes d'optimisation de manière non-intrusive pour le reste du logiciel.

# 3 Application à la conception multi-critères de stratifiés composites

Lors des premières étapes d'un projet de conception, il est utile d'avoir un aperçu non biaisé des effets du choix des matériaux sur les critères. Une optimisation multi-critères est alors indiquée car elle vise à exhiber tous les compromis possibles entre les différents critères (le front de Pareto). Un exemple de ce type d'analyse est maintenant réalisé avec LAMKIT. La figure 3 montre le fichier de mise en données optimisation. Les 9 variables, déclarées dans la rubrique `***variables`, sont le matériau (graphite-époxyde ou carbone-époxyde haute résistance ou carbone-époxyde haut module), le nombre de plis (par type de plis) et l'orientation des fibres dans chaque pli. Le stratifié reste équilibré et symétrique grâce aux instructions `lam_type BS`. Les critères de conception sont déclarés dans la rubrique `***criteria` : il s'agit de minimiser l'épaisseur, maximiser la charge critique de flambement, maximiser la résistance à la rupture pour des critères de $|\varepsilon_1|$ et $|\varepsilon_2|$ maximaux, et de maximiser les rigidités longitudinales et transverses. L'algorithme d'optimisation multi-critères évolutionnaire (de type Niched-Pareto, [6]) est invoqué par le mot clé `multievolution`. La plaque fait $35\ cm$ de longueur et $20\ cm$ de largeur et est soumise à une charge compressive de $10^6\ N$ suivant $x$ et extensive de $10^4\ N$ suivant $y$.

L'exécution de l'optimisation multi-critères produit une approximation du front de Pareto qui est une surface dans l'espace à 6 dimensions des critères. Cette surface, ainsi que de la population initiale (un ensemble de stratifiés choisis au hasard), est projetée dans le plan (`épaisseur, flambement`) sur la figure 4. On constate que l'optimisation multi-critères améliore les solutions aléatoires de la population initiale et, simultanément, propose des nouveaux stratifiés épais, résistants au flambement. Le front de Pareto permet ici, en prédimensionnement, d'estimer le nombre de plis souhaitables pour soutenir un chargement donné. On peut ensuite extraire du front de Pareto les solutions qui paraissent les plus intéressantes. Par exemple, le stratifié suivant conjugue fibres à haute résistance et à haut module : $((0_{12})^{CEHR}/(\pm 45)_9^{CEHM}/(0_{20})^{CEHR})_s$. Il a une épaisseur totale $H = 1.25\ cm$, et $\lambda_{flamb} = 2.29, \lambda_{maxeto1} = 1.02, \lambda_{maxeto1} = 1.07, E_x = 81.69\,GPa, E_y = 25.75\,GPa$. On remarque comment les plis à haut module, qui contribuent à la rigidité de la structure, sont désaxés pour réduire les déformations principales.

# 4 Conclusions

Le logiciel LAMKIT© EADS-CCR 2002 intègre l'interface orientée objet simulation-optimisation décrite à des modules de simulation de plaques composites : calcul des rigidités, des coefficients de dilatation thermique et hydrique, critères de rupture, flambement, calcul probabiliste des propriétés par la méthode de Monte Carlo, analyse en fluage-relaxation, calculs d'absorption d'humidité. Il constitue la plate-forme logicielle du Centre de Recherche d'EADS dans le domaine des matériaux composites et est constamment enrichi des avancées qui y sont effectuées, tant en optimisation que dans le domaine du comportement des structures. Sous l'environnement Windows, LAMKIT dispose d'une interface graphique conviviale et intuitive, qui met ainsi à portée des ingénieurs un large spectre d'outils avancés en pré-dimensionnement composite. Il est maintenu et distribué par EADS-CIMPA.

## Références


[1] BÄCK, T. *Evolutionary Algorithms in Theory and Practice*. Oxford Univ. Press, New York, USA (1996).

[2] BESSON, J., FOERCH, R. Object-oriented programming applied to the finite element method – Part 1 : General concepts. *Revue Européenne des Eléments Finis*, **7**(5), 535–566 (1998).

[3] EADS Centre Commun de Recherches. *LAMKIT – Online Presentation* (2001). available at http ://www.eads.net/lamkit.

[4] ELDRED, M.S., GIUNTA, A.A., VAN BLOEMEN WAANDERS, B.G., WOJTKIEWICZ, S.F., HART, W.E., ALLEVA, M. P. *DAKOTA, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis. Version 3.0 Users Manual*. Sandia Natl. Laboratories, Albuquerque, NM, USA (December 2001). Sandia Technical Report SAND2001-3796P.

[5] GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addisson-Wesley professional computing series. Addisson-Wesley (1994).

[6] HORN, J., NAFPLIOTIS, N. Multiobjective Optimization Using The Niched Pareto Genetic Algorithm. Tech. Rep. IllIGAL 93005, Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, USA (1993).

[7] MOORE, G.J. *MSC/NASTRAN Design Sensitivity and Optimization User's Guide*. The MacNeal-Schwendler Corporation, Los Angeles, CA, USA (1992).

[8] North Western Numerics, Ecole des Mines de Paris, Onera, Seattle, WA, USA or Centre des Matériaux P.-M. Fourt, Evry, France. *Z-set – User Manual, release 8.0* (2001). available at http ://www.nwnumerics.com.

[9] STROUSTRUP, B. *The C++ Programming Language*. Addison-Wesley, third edn. (1997).