Subvention accordée par la Région wallonne
à une unité de recherche universitaire
ou de niveau universitaire

Open Numerical Engineering LABoratory

ONELAB

Rapport technique

N° 2

période du 01/10/2011 au 30/09/2012

Numéros de convention

WIST 3.0 No 1017086

Note : ce rapport technique est un document évolutif. Il sera complété au fur et à mesure de l'avancement du projet.

Introduction

Ce rapport technique reprend les résultats principaux du projet ONELAB, dont l'objectif est de développer une plateforme d'intégration de logiciels libres pour le calcul scientifique en ingénierie.

Motivation générale et contexte économique

Les entreprises en Région Wallonne sont grandes utilisatrices de logiciels de calcul scientifique. Parmi les logiciels les plus utilisés, on peut citer Fluent pour la mécanique des fluides, Abaqus, Ansys ou Samcef pour le calcul de structures, et Flux2D/3D ou Ansoft pour la simulation électromagnétique. Le coût des licences de ces logiciels commerciaux peut s'élever à plusieurs dizaines de milliers d'euros par an. Si cet investissement est justifié et supportable par des entreprises ayant un besoin très régulier de simulations et disposant d'un personnel qualifié les utilisant de manière régulière, il sort très rapidement du budget d'une PME n'ayant besoin de recourir à la simulation que de manière sporadique. Les PMEs peuvent alors se tourner vers des bureaux d'études comme GDTech. Malheureusement, pour des simulations complexes, le coût des licences encouru par le bureau d'étude, répercuté sur la PME, devient également très rapidement prohibitif. Pour fixer les idées, une licence Fluent pour une station de travail bi-processeur est de l'ordre de 30.000 euros/an. Une PME ne peut donc habituellement pas se permettre un calcul de grande taille nécessitant quelques dizaines de processeurs... une situation aberrante à l'heure actuelle, où une simple station de travail possède 8 coeurs de calcul.

Les logiciels open source constituent une solution à ce problème. En effet, la communauté du logiciel libre produit depuis plusieurs années des logiciels de niveau professionnel. Un exemple en bureautique est la suite OpenOffice, de plus en plus répandue. Dans le domaine du calcul scientifique, la plupart des distributions Linux proposent également des logiciels scientifiques de grande qualité. Certains sont des librairies de bas niveau, réservées à un cercle de spécialistes (PETSc, Metis, VTK, MadLib, Open CASCADE, etc.). Mais il existe également des logiciels de haut niveau destinés à l'utilisateur final, par exemple pour des problématiques de segmentation d'images (ITK), de création de maillage éléments finis (Gmsh, Meshlab) ou de visualisation des résultats (Paraview, Gmsh). Dans les domaines de la mécanique des solides, de la mécanique des fluides et de l'électromagnétisme, la qualité de ces logiciels de haut niveau (Code_Aster en mécanique du solide, OpenFOAM en mécanique des fluides, GetDP en électromagnétisme) est telle qu'ils sont compétitifs par rapport aux solutions commerciales équivalentes, tant au niveau de leurs capacités que de leurs performances, tandis qu'il sont souvent techniquement supérieurs, plus ouverts et flexibles. Cependant, malgré leurs qualités reconnues, ces logiciels restent très peu utilisés par les PMEs wallonnes. Nous pensons que l'absence d'une interface facile d'emploi pour le pré- et le post-traitement est en grande partie responsable de cet état de fait, de même que, pour les logiciels émanant du monde académique, un manque de documentation et d'exemples adaptés. Un frein à l'adoption de solutions open source réside

également dans l'amalgame qui perdure dans le monde industriel entre « logiciel open source » et « freeware » (logiciel « gratuit », « limité », « non-professionnel »).

Forts de notre expertise dans le domaine du logiciel libre (Gmsh, Madlib, GetDP), nous souhaitons dans ce projet développer un outil logiciel qui permette de piloter divers logiciels libres de qualité professionnelle (OpenFOAM, Code_Aster, GetDP, Elmer) via l'unification de la gestion du pré- et du post-traitement. L'interface unique rendra l'accès à une plateforme libre beaucoup plus aisé pour les PME wallonnes, diminuant les coûts liés à l'achat de licences de logiciels propriétaires et augmentant donc leur compétitivité. Cette interface unique permettra également de mettre sur pied une plateforme très utile au niveau pédagogique, tant dans les Universités que dans les écoles supérieures et les centres de formation continuée. La formation à ces nouvelles technologies open source permettra à nouveau de diminuer les coûts associés à l'achat d'onéreuses licences des produits commerciaux mais aussi à préparer les futurs ingénieurs aux outils qu'ils rencontreront au sein des PMEs wallonnes une fois diplômés.

Motivation technique

Une des principales difficultés rencontrées par les ingénieurs non-spécialistes des méthodes de simulation et souhaitant exploiter des logiciels libres est l'**hétérogénéité des outils** d'une chaîne de résolution open source (en contraste avec le haut niveau d'intégration offert par les outils commerciaux équivalents). Une chaîne de résolution complète comprend en général les 5 étapes suivantes:

- Installation des logiciels
- Création ou import de géométrie dans un logiciel de CAO (Gmsh, Ansys, etc.)
- Maillage (Gmsh, Hypermesh, Ansys, etc.)
- Calcul dans l'environnement de simulation (GetDP, OpenFoam, Code_Aster, dg, etc.)
- Post-traitement ("post-processing") des résultats (Gmsh, Paraview, etc.)

Lorsque l'on prend le parti de travailler avec des logiciels libres, on est amené à devoir intégrer dans cette chaîne de résolution plusieurs logiciels indépendants. Cette hétérogénéité des outils logiciels va de pair avec une hétérogénéité des concepts utilisés, des formats de données, etc., ce qui rend la tâche en général difficile.

Une seconde difficulté est l'absence d'une formalisation explicite des relations existant entre les données d'entrée du modèle et les caractéristiques de la chaîne de résolution qui permet d'obtenir une solution satisfaisante. Les logiciels de simulation permettent d'obtenir la solution correspondant à un ensemble de données d'entrée mais ne disposent d'aucun moyen d'encoder les règles de bonne pratique et l'expérience acquise au cours d'une série de simulations successives. Cette formalisation demande un niveau d'abstraction spécifique, qu'on peut appeler **couche "experte"** (c.-à-d. relevant du domaine de l'expertise) qui représente l'essentiel de la valeur ajoutée des logiciels commerciaux, ainsi que la clé *sine qua non* d'accès pour l'utilisateur non-spécialiste. C'est précisément l'absence de cette couche experte formalisée, capable de guider l'utilisateur pas à pas dans la définition d'un modèle correct et correspondant à ses souhaits, qui représente l'obstacle majeur à la diffusion des logiciels libres

dans le domaine de l'ingénierie industrielle.

Dans le cas des logiciels commerciaux, la composante “experte” est fournie comme un produit fini. Il s'agit d'un ensemble de fonctionnalités prédéfinies, matérialisées par un système de menus et de procédures internes, permettant de sélectionner des schémas de modélisation préétablis et validés. C’est ce qui garantit la robustesse de ces logiciels et , partant, ce qui représente la valeur ajoutée du produit et justifie son prix. Dans le cas de logiciels libres par contre, sachant que l'on vise à la combinaison de logiciels indépendants, il est impossible d’assurer une validation préalable des schémas de résolution, puisque les fonctionnalités disponibles dépendent du choix des logiciels utilisés. La composante “experte” repose donc sur l’ajout d’une couche ONELAB, comme il sera expliqué plus loin.

C'est à cette double problématique (hétérogénéité des outils, absence d'une couche "experte") que le projet ONELAB vise à apporter une solution.

Principes généraux de la mise en oeuvre

Les principes de base de l’interface ONELAB sont au nombre de 5. Ils sont maintenant détaillés.

Interface basée sur une triple abstraction

Les interactions entre les différents maillons d'une chaîne de résolution se font essentiellement par l'échange de fichiers de données et de paramètres. Pour pallier l'hétérogénéité des outils utilisés, il faut donc créer une interface abstraite et persistante capable de gérer ces interactions de façon centralisée et transparente pour l'utilisateur. Les différents maillons de la chaîne de résolution sont en effet amenés à partager un ensemble de paramètres communs, lesquels doivent être introduits aux différents niveaux sous des formats spécifiques.

Le rôle de l’interface abstraite est de centraliser l'ensemble des paramètres du modèle dans une structure de données unique, et de les distribuer ensuite aux différents modules. L’interface abstraite donne ainsi à la chaîne de résolution hétérogène l'aspect extérieur d'un modèle unique. C'est le niveau d'abstraction recherché par l'utilisateur. C'est aussi celui qui permet de contrôler de façon centralisée les interactions entre les différents modules de la chaîne de résolution.

L'interface abstraite ONELAB est implantée dans le logiciel open source Gmsh. Elle est basée sur une triple abstraction :

1. abstraction de l'interface vers les modeleurs géométriques (CAO) et la génération/ simplification de maillages ;
2. abstraction de la définition des propriétés physiques, des contraintes et des paramètres de pilotage des différents logiciels utilisés ;
3. abstraction et consolidation des fonctionnalités de post-traitement.

Approche client-serveur

L’approche choisie pour la fonctionnalité d'intégration de ONELAB est celle d'une relation client-serveur : l’ensemble des paramètres sont stockés sur un serveur vis-à-vis duquel les logiciels de simulation et l’utilisateur jouent le rôle des clients. Une approche de type client-serveur permet de considérer de façon naturelle et transparente les différents cas de figure où les clients tournent soit sur la même machine soit sur des machines différentes.

Implémentation en C++

Pour une portabilité la plus large possible (y compris vers les plateformes de type iPad), la librairie de gestion client-serveur est écrite en C++ plutôt que dans un langage interprété.

Fonctionnement générique

L'implémentation de la relation client-serveur est conçue pour fonctionner de façon générique, c’est-à-dire qu’elle peut utiliser les logiciels clients tels quels, au moyen de leurs canaux de communications (entrées-sorties) propres. Lorsque le logiciel client intègre directement l’interface ONELAB on l’appelle client “natif”. Pour tous les autres logiciels, une couche interface est ajoutée pour permettre la communication du client avec le serveur ONELAB (client “interfacé”).

ONELAB ne construit donc aucun méta-langage, ni format de fichier commun et le serveur ONELAB ne connaît rien *a priori* des logiciels qu'il commande. Toute l'information à teneur technique, mathématique ou physique vient des clients, ce qui assure que l’interface ONELAB se cantonne bien dans son rôle d’interface générique, et que l’on se prémunit ainsi du risque d’aboutir à une librairie tentaculaire coûteuse à développer et maintenir sur le long terme. Ceci constitue une différence notable par rapport à la plupart des approches classiques d’écriture d’interfaces (graphiques ou non) de pilotage de codes de calcul scientifique. Habituellement en effet, l’interface est écrite de manière *ad hoc* pour un solveur donné : en fonction des choix de l’utilisateur, l’interface génère le jeu de données nécessaires au code de calcul dans son format natif, puis exécute celui-ci. L’approche de ONELAB est différente en ceci que l’interface ignore la syntaxe propre des logiciels pilotés ; ce sont ces derniers qui interagissent avec l’interface et requièrent les données dont ils on besoin.

Couche experte

En pratique, un modèle numérique se compose de deux choses : un schéma de résolution paramétrique et un ensemble de valeurs pour les paramètres de ce schéma. Le modèle est opérationnel lorsque le schéma de résolution et les valeurs des paramètres ont été déterminés de façon à ce que le modèle soit exempt d’erreur et fournisse les résultats désirés.

Partant d'un modèle opérationnel, la question se pose cependant rapidement à l’utilisateur de savoir dans quelle mesure on peut s'écarter des paramètres initiaux sans détériorer la qualité, voire la validité (physique), de la solution obtenue. Si, d'autre part les modifications de paramètres sont plus importantes, la question devient alors de savoir comment modifier le schéma de résolution pour s'adapter aux nouvelles caractéristiques du problème. La fonction de la couche experte est de fournir à l'utilisateur une aide à la détermination du schéma de résolution est des “bons” paramètres d’entrée pour chaque cas particulier de simulation. À cette

fin, la fonctionnalité d'intégration décrite précédemment (l'interface abstraite) n'apporte pas de solution et une fonctionnalité complémentaire doit être ajoutée.

D'une façon générale, le projet ONELAB n'aborde pas ce problème de façon théorique. Nous pensons que la définition *a priori* des conditions générales et théoriques de validité/qualité/ précision d'un modèle est illusoire et de surcroît contre-productive dans les cas d'applications pratiques. L'approche ONELAB est plus pragmatique, mais aussi plus générale. Outre le fait qu'il permet l'utilisation de différents clients dans un environnement unique, l'environnement ONELAB permet également l'accumulation des connaissances acquises. Chaque simulation effectuée apporte en effet un certain nombre d'informations utiles (qu'elles soient positives ou négatives) concernant le problème étudié. Il est en règle générale du ressort exclusif de l'utilisateur d'archiver, traiter et hiérarchiser ces données de simulation. L'accumulation de connaissance se passe à deux niveaux. Tout d'abord dans la description de schémas de résolution génériques validés (ce que nous appellerons les métamodèles), et d'autre part dans un certain nombre d'outils d'aide au choix des paramètres du métamodèle en se basant sur l'exploitation des résultats stockés dans une base de données de simulation. La base de données de simulation et la librairie de métamodèles constituent conjointement la couche experte de l'environnement de modélisation ONELAB.

C'est en fournissant ces outils que ONELAB pourra reconstruire "de l'extérieur" la composante "experte" qui est implémentée en dur "de l'intérieur" dans un logiciel commercial tel que COMSOL.

Délivrables

Les délivrables du projet ONELAB sont :

1. L'interface abstraite ONELAB vers les logiciels open source de calcul scientifique, sous forme :
    a. d'une librairie générique d'échange de données
        i. de CAO/maillage
        ii. de paramètres physiques, de contraintes, de paramètres de pilotage de code
        iii. de post-processing
    b. des interfaces nécessaires vers les logiciels clients en mécanique du solide, en mécanique des fluides et en électromagnétisme
2. Un ensemble représentatif de métamodèles couvrant les domaines de la mécanique du solide, de la mécanique des fluides et de l'électromagnétisme. Même s'ils peuvent aussi servir de cas-test, les exemples que l'on choisit doivent viser à mettre en évidence des simulations représentatives de simulations réelles. Ces exemples sont directement pilotables depuis l'interface ONELAB. Ils sont documentés pour permettre aux utilisateurs d'aller au-delà des "templates" s'ils le désirent.

Toutes les informations relatives au projet ONELAB sont disponibles sur le site internet http://onelab.info. Ce site contient, sous forme de wiki, la documentation des différents codes et modèles pilotés par l'interface ONELAB, ainsi que la plupart des informations contenues dans ce rapport.

Organisation du rapport

L'organisation du rapport suit celle du projet : les résultats obtenus pour les cinq tâches (T1 à T5) sont repris dans les cinq chapitres numérotés T1 à T5.

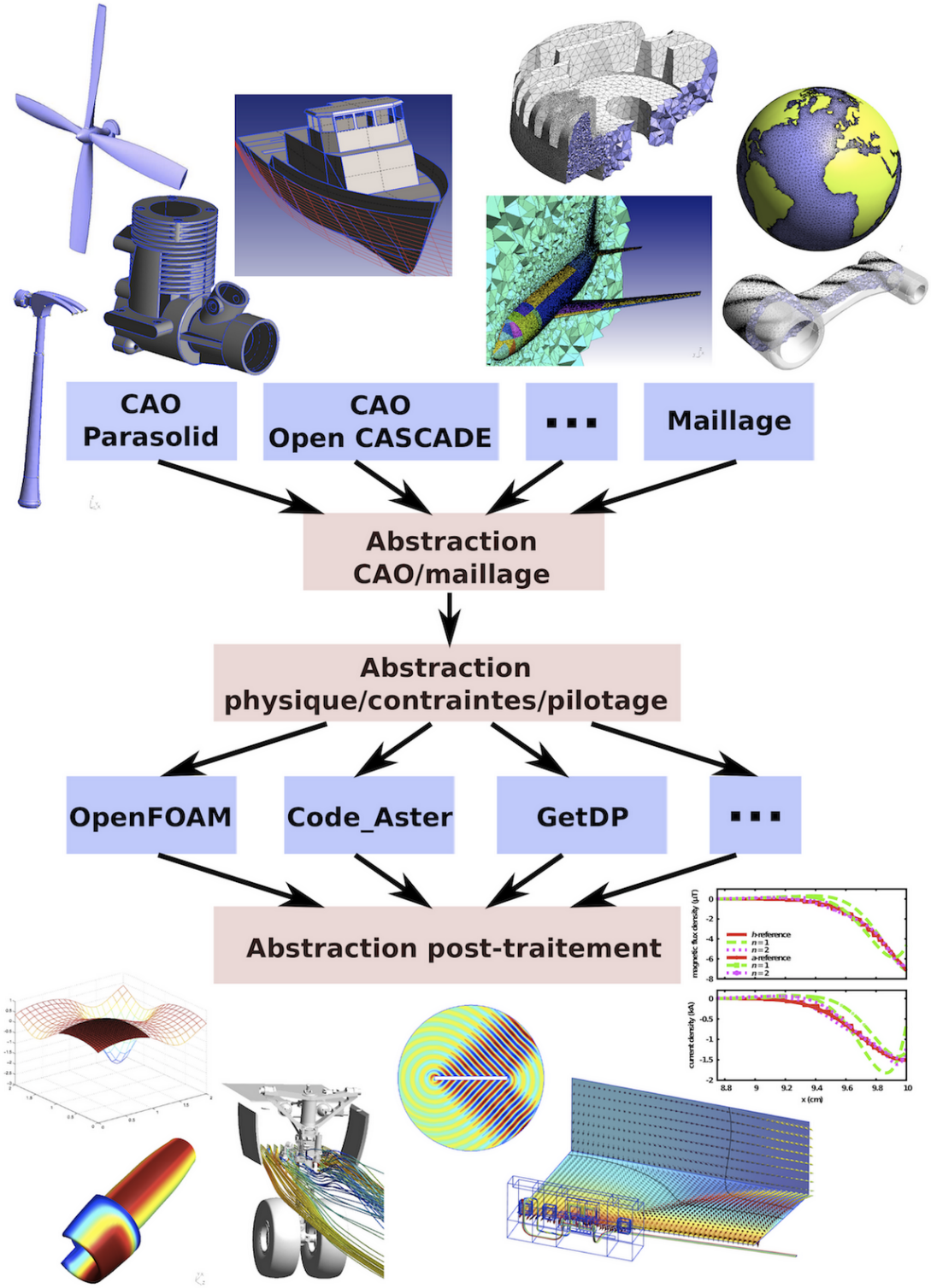


Figure 1. L'interface ONELAB est basée sur une triple abstraction : 1) de la CAO et du maillage, 2) des propriétés physiques, des contraintes et des paramètres de pilotage des codes open source "clients" et 3) du post-traitement.

T1 Spécification des interfaces

La première tâche du projet ONELAB est de concevoir l'interface abstraite vers les codes open source de calcul scientifique dans les domaines de la mécanique des fluides, de la mécanique du solide et de l'électromagnétisme. La spécification de cette interface se situe à trois niveaux : CAO/maillage (T1.1), paramètres physiques/contraintes/pilotage des codes (T1.2), et post-processing (T1.3).

T1.1 Spécification de l'interface CAO/maillage

Le but de cette tâche est de spécifier l'interface abstraite (API) vers la CAO et les maillages. Il a été décidé de baser cette interface abstraite sur l'interface de modèle géométrique "GModel" du code open source Gmsh développé par les membres du consortium.

La description géométrique pourra être fournie

- soit sous forme d'un maillage, auquel cas l'utilisateur pourra procéder à des opérations de nettoyage, de remaillage ou de simplification (pour mieux adapter la discrétisation à la physique et/ou à la méthode de calcul utilisée par le logiciel client) ou d'ajouts d'autres parties déjà discrétisées (afin de modifier un design existant) ;
- soit sous forme d'une CAO, en format natif tel Parasolid ou OpenCASCADE ou dans un format d'échange standard comme STEP ou IGES. Dans ce cas, l'utilisateur pourra également procéder à des opérations de nettoyage, de simplification ou d'ajout avant de procéder au maillage proprement dit.

L'abstraction vers la CAO et les procédures de (re)maillage est basée sur l'accès direct aux représentations géométriques des utilisateurs dans leur format natif, sans traduction de fichiers. A cet effet, le module géométrique de l'interface s'appuie sur une hiérarchie de classes abstraites permettant de représenter la topologie de n'importe quel modèle CAO, suivant le modèle B-REP ("Boundary REPresentation"). Ensuite, à chaque entité topologique abstraite est associée une implémentation concrète de sa géométrie pour chaque moteur CAO supporté (Parasolid, OpenCASCADE, Gmsh, etc.) Cette implémentation concrète se fait directement via l'API ("Application Programming Interface") de chaque moteur CAO. L'absence de traduction est un aspect crucial pour une utilisation industrielle, car toute traduction de modèles géométriques complexes est entachée d'erreurs liées aux tolérances géométriques et aux légères différences mathématiques entre les représentations natives et les représentations internes.

Après modifications éventuelles pour s'adapter aux exigences particulières du problème physique considéré ou de la méthode numérique utilisée par le logiciel client (ajouts de composants, raffinement ou simplification de maillage, etc.), l'utilisateur pourra mailler son modèle et exporter le maillage dans un format compris par les logiciels clients (MED, UNV, MSH). Tous les algorithmes de modification de modèles (ajouts, simplification, nettoyage et

maillage), de même que les algorithmes de visualisation et de manipulation, seront écrits en termes des classes abstraites. Une nouvelle interface CAO ne demandera dès lors aucune modification à ces algorithmes—uniquement l'ajout d'une implémentation concrète supplémentaire de la hiérarchie des classes d'accès.

Un point délicat au niveau des interfaces CAO dans un contexte industriel concerne la réparation de modèles CAO incorrects (présentant e.g. des chevauchements ou des trous non-physiques). Les développements récents des membres du consortium dans le domaine de la reparamétrisation des surfaces discrètes seront exploités dans ce cadre afin de proposer une nouvelle méthode de réparation de géométries défectueuses, indépendante du moteur CAO utilisé pour les créer.

Les modifications interactives ou scriptées de la géométrie seront gérées par la même interface abstraite de gestion des paramètres que celle conçue pour l'échange des propriétés physiques définies dans la section suivante.

T1.2 Spécification de l'interface physique / contraintes / pilotage

Le but de cette tâche est de déterminer les caractéristiques souhaitées de l'interface abstraite vers les propriétés physiques, les contraintes et les paramètres de pilotage des différents logiciels clients.

La notion de métamodèle

On peut voir une simulation numérique comme une application qui associe à une ensemble de variables d'entrée un ensemble de résultats numériques qui sont l'objet de la modélisation entreprise. Une simulation est basée sur la séquence d'opération: maillage / pré-processing / résolution / post-processing effectuées par une chaîne d'outils numériques déterminés. Deux situations sont alors à distinguer. Si cette chaîne d'opération est effectuée au moyen d'un outil commercial intégré, l'utilisateur est guidé dans son travail de modélisation par l'application et il n'a pas besoin de connaissances particulières en calcul scientifique. Dans le cas contraire, c'est-à-dire si l'on souhaite ne pas dépendre d'une application commerciale, les différents éléments de la chaîne de simulation doivent être bien maîtrisés par l'utilisateur. Cette dernière situation est celle qui prévaut dans les universités, centres de recherche et bureaux d'études.

L'objectif du projet ONELAB est de réaliser une intégration semblable à celle offerte par les outils de simulation commerciaux, mais sur base d'outils numériques open-source indépendants les uns des autres. Plusieurs modalités d'intégration sont possibles. Le projet SALOME a, par exemple, entrepris un intégration explicite de nombreux codes open-source dans une superstrucure logicielle globale. Ce type d'intégration monolithique est cependant extrêmement lourd à réaliser et à maintenir et ne peut être mené à bien que par un consortium.

L’analyse nous a montré qu’une alternative existe et qu’un type d’intégration plus léger et flexible, tout en restant fiable et en offrant sensiblement les mêmes fonctionalités, est possible. Il repose sur la notion centrale de **métamodèle**.

Concrètement, on attend d’un métamodèle qu’il offre les fonctionnalités suivantes:

1. une structure persistante permettant l’échange de paramètres entre codes différents, c’est-à-dire communication entre des tâches (threads) distinctes, tournant sur la même machine ou sur des machines distantes,
2. la possibilité de définir des chaînes de simulation (séquentielles, et éventuellement parallèles) avec une logique de contrôle (if... then... else...) et sans limitation de complexité,
3. la possibilité d’incorporer des outils scientifiques standard (python, etc...) pour le traitement automatique des résultats numériques de simulation,
4. la possibilité de contrôler la cohérence des modèles multi-code en vérifiant les flux de données entre les différents clients,
5. des outils de monitoring en cours de simulation,
6. des outils de diagnostic et une aide à l'interprétation des erreurs,
7. une structure persistante permettant l’archivage et le traitement des résultats obtenus lors de simulations successives, structure évolutive appelée à s'enrichir (par l’addition de données supplémentaires et par l’amélioration des procédures) au fil de l’expérience acquise,
8. la possibilité de travailler à un niveau d’analyse supérieur utilisant le modèle numérique comme une boîte noire afin de réaliser des études paramétriques, des optimisations simples, etc.

Puisque ONELAB autorise l’utilisation conjointe de logiciels différents, elle rend aussi possible la comparaison de résultats obtenus avec des logiciels concurrents ou des approches théoriques différentes, permettant un niveau de validation des résultats supérieur à ce qu’est en mesure d’offrir un code unique. L’interface abstraite pourra également servir d'outil de validation pour les logiciels clients (et des métamodèles associés) : en spécifiant les valeurs attendues de certaines variables de sortie, on peut utiliser ONELAB (e.g. via un environnement de test comme CTest : www.cmake.org) pour effectuer des tests automatiques de validité des codes et des métamodèles.

Suivent deux exemples de métamodèles, pour fixer les idées:

Exemple 1: Dans un problème de magnétodynamique, la profondeur de pénétration fait partie des données de sortie, elle dépend de la fréquence de travail et des caractéristiques physiques des matériaux. Elle doit cependant être prise en considération lors de la définition des tailles de maille, qui elle est une donnée d'entrée. La règle de bonne pratique "au moins 3 éléments finis sur la profondeur de pénétration" est une spécification inverse en ce sens qu'elle nécessite la connaissance de la solution du problème. Dans l’environnement ONELAB, il devra être possible d’effectuer une première simulation pour évaluer la profondeur de pénétration (pour laquelle aucune estimation analytique n’existe dans la cas non-linéaire) puis de prendre compte de cette

valeur pour la réalisation d'une seconde simulation avec un maillage adapté. Avec des outils conventionnels, l'intervention de l'utilisateur serait nécessaire.

Exemple 2: Un métamodèle de poutre pourra combiner un simple modèle analytique en éléments de réduction, une modèle 2D et un modèle 3D, et encore éventuellement un modèle dual en contraintes. Ces différents modèles partagent une partie de leurs données d'entrées (dimensions, matériaux, etc). Chacun des modèles concurrents a ses avantages et inconvénients en termes de précision, temps de calcul, applicabilité, etc... On peut ainsi découper l'espace des paramètres du modèle (y compris des caractéristiques de la discrétisation telles que une carte de taille, etc.) en terme des différentes modèles disponibles. Le métamodèle peut alors être augmenté d'une logique de sélection de modèle permettant son exploitation optimale en fonction des exigences d'exactitude, de temps et de précision donnée par l'utilisateur.

Abstraction des propriétés physiques

Nous sommes jusqu'à présent restés vague quant à la définition exacte de ce que l'on entend par "paramètres du modèle". On n'entend pas toujours par paramètre une valeur numérique. Il existe en effet d'autres types de grandeurs que l'on peut laisser libres lors de la définition d'un modèle paramétrique. Il y en a essentiellement trois:

- des chaînes de caractères: nom de matériaux prédéfinis, nom de formulation, …
- des fonctions: le modèle paramétrique attend une relation algébrique entre 2 (ou plus) inconnues (p.ex. la relation constitutive d'un matériau) mais l'expression explicite de cette relation est laissée libre au niveau du modèle paramétrique et doit être fournie par l'utilisateur avant l'exécution du modèle,
- des régions abstraites: le modèle paramétrique travaille sur base de régions abstraites (p.ex. un domaine conducteur, une condition limite de type Neumann, …) et c'est à l'utilisateur d'identifier les zones du maillage avec ces entités abstraites.

L'abstraction de ces données "non numériques" sera également réalisée au travers du serveur de paramètres permettant de stocker de manière unifiée toutes les informations requises par le modèle éléments finis sous-jacent. Chacun des types de paramètres considérés (nombres, chaînes de caractère, fonction, région) est implémenté sous forme d'une classe C++ spécifique. Les propriétés physiques (matériaux) peuvent être fournies par une loi de comportement linéaire ou non linéaire, une constante, un tenseur, une fonction analytique, des données émanant de mesures, etc. Les contraintes peuvent être associées à des entités géométriques de dimensions diverses (points, lignes, surfaces, volumes) et imposées elles aussi au moyen de constantes, vecteurs, tenseurs, fonctions qui varient dans le temps, des résultats de mesures, etc.

Base de donnée de simulation

Comme expliqué plus haut, le métamodèle est doté d'une "mémoire" matérialisée par une base de données de simulation dans laquelle sont stockées les données d'entrée et de sortie de chacune des simulations effectuées. L'objectif est de faire en sorte que les informations

obtenues aux étapes antérieures ne soient pas perdues mais restent utilisables (par un traitement adéquat des données accumulées dans la base de données) et interprétables afin de guider l'utilisateur vers un choix de paramètres valide correspondant à un modèle opérationnel. De cette façon, on enrichit au fil des essais et erreurs une heuristique propre à chaque problème et on construit "de l'extérieur" une fonctionnalité experte. Cette base de données de simulations permet également la comparaison des résultats obtenus, pour le même problème, avec des modèles concurrents. En ce sens elle contribue au choix non seulement des "bonnes" variables d'entrée, mais également au choix de la "bonne" formulation, du "bon" logiciel, etc... Concrètement, en plus de l'espace des paramètres, les information suivantes sont archivées:

- des "diagnostics" des logiciels clients :
    - diagnostics de compatibilité (paramètre manquant, formulation incomplète, région géométrique non-définie, etc.)
    - diagnostics concernant la solution (convergence, nombre de pas de temps moyen, erreur, temps de calcul)
    - diagnostics "temps réel" (état d'avancement du calcul, mémoire utilisée, etc.)
- des solutions (au niveau du post-processing), sous forme
    - de "pointeurs" vers des fichiers (e.g. .pos, .gnuplot, etc.)
    - de valeurs numériques (e.g. grandeurs globales)
    - d'images
    - de commentaires en format texte

T1.3 Spécification de l'interface post-processing

Le but de cette tâche est de spécifier l'interface abstraite (API) vers les données de post-traitement. Il a été décidé de baser l'approche sur l'interface actuelle de post-processing de Gmsh, et de l'étendre pour accéder aux données des logiciels clients :

- il a été décider de supporter le format d'échange de données MED3, développé par EDF et le CEA, qui est le format natif de Code_Aster (et de Code_Saturne) et sur lequel repose la plateforme Salomé ;
- il a été également décidé d'évaluer les capacités actuelles du format CGNS, en vue d'une possible intégration dans l'interface ONELAB.

T2 Interface CAO / maillage

T2.1 Implémentation de l'interface abstraite

L'interface abstraite vers la CAO et les maillages est basée sur le modèle géométrique GModel de Gmsh. L'implémentation est réalisée en C++, et repose sur une hiérarchie de classes abtraites, dérivées de la classe GEntity. Elle est accessible dans le répertoire `gmsh/Geo` du logiciel Gmsh. Diverses implémentations concrètes en lecture existent pour le noyau CAO Gmsh, OpenCASCADE, ParaSolid et ACIS. Pour l'écriture, la classe GModelFactory est en cours de développement pour le noyau CAO Gmsh et OpenCASCADE.

L'interface abstraite CAO/maillage est présentée en détail dans l'annexe B.

En conjonction avec cette interface abstraite vers les géométries et le maillage, Gmsh est devenu un client du serveur ONELAB de gestion des propriétés physiques, des contraintes et des paramètres des logiciels de simulation (voir chapitre suivant). Dans l'implémentation actuelle, cela signifie que Gmsh sert à la fois de serveur et de client ; et que des paramètres géométriques peuvent être modifiés en temps réel de la même manière que les paramètres des logiciels de simulation.

T2.2 Reparametrisation et nettoyage de géométries

Les fonctionnalités de reparamétrisation sont présentées dans l'annexe C.

T3 Interface physique / contraintes / pilotage

T3.1 Implémentation de l'interface abstraite

Organisation logique

Pour satisfaire aux spécifications de la liste établie à la section T1.2, le métamodèle ONELAB s'organise en trois niveaux, à savoir:

- le **métamodèle** (exactement un serveur de paramètres)
- la **simulation** (exactement un record dans la base de données de simulations)
- la **tâche** (exactement un client).

Une **tâche** est une opération de calcul réalisée par un client bien défini. Ensuite, la **simulation** est l'exécution de A à Z des opérations nécessaires pour l'obtention des résultats correspondant à un ensemble déterminé de données d'entrées. Les différentes tâches d'une simulation s'enchaînent logiquement selon un schéma classique maillage/ préprocessing/ processing/ postprocessing. Une simulation peut cependant également contenir un certain nombre de tâches concurrentes dont on veut comparer les résultats. Le **métamodèle**, finalement, est la structure abstraite dont chaque invocation est une simulation. Autrement dit, le métamodèle est un ensemble de simulations possibles, étant donné les clients et l'espace paramétrique qui ont été choisis.

L'invocation du métamodèle comporte deux aspects: 1. transmettre aux clients un ensemble de données d'entrée cohérent tout au long de la simulation, et 2. appeler successivement les clients. Cette succession d'appel est structurée par une logique de branchement dépendant de la physique, du résultat de simulations préalables, ainsi que de différents outils de contrôle et de diagnostic. Le métamodèle peut donc être vu comme un script multi-code paramétrique assorti d'une base de donnée permettant l'échange des paramètres entre les différents codes invoqués.

Des métamodèles élémentaires (p. ex. Gmsh+GetDP) peuvent être gérés directement par le GUI. Pour les métamodèles plus complexes, une **syntaxe** simple et générale a été définie, qui sert à la fois pour la description des métamodèles (indiquer la succession des tâches à effectuer) et pour l'interfaçage des différents codes contrôlés par le métamodèle.

Cette syntaxe permet d'abord de définir les quatre types de paramètres reconnus par ONELAB (`number, string, region, function,` voir plus bas pour une description plus détaillée), d'accéder à l'ensemble de leurs propriétés (bornes, visibilité dans la fenêtre interactive...), d'accéder à leur valeur (`get`) et de les modifier (`set`). Pour permettre un compréhension plus

intuitive, des paramètres spécialisés (bouton radios ON/OFF, liste de choix numériques, etc..) sont également reconnus.

Le métamodèle proprement dit, d'autre part, est décrit dans un fichier texte (extension `.ol`) et consiste en l'enregistrement des clients (`register`) et la déclaration de leurs fichiers d'entrée (`in`), de leurs fichiers de sortie (`out`) et de leurs arguments de ligne de commande (run). Les commandes `up` et `merge` permettent d'indiquer à ONELAB des informations, respectivement numérique ou graphique, qui doivent être transmise au GUI après exécution du client.

Des branchements logiques peuvent être effectués, tant dans la définition des paramètres que dans celle des clients. Les commandes de branchement logique sont les suivantes: `iftrue, ifntrue, if, else, endif`.

Clients natifs et clients interfacés - préprocesseur ONELAB

Les logiciels Gmsh et GetDP (développés à l'ULg) sont qualifiés de "natif" car les fonctionnalités ONELAB y ont été intégrées directement dans le code comme surcharge de fonctionnalités existantes. Cette implémentation "en dur" de ONELAB dans un logiciel requiert non seulement l'accès aux sources de celui-ci mais également une connaissance détaillée (niveau développeur) de celui-ci. Pour tous les autres logiciels, l'interfaçage ONELAB se fait au travers de l'instrumentation des I/O naturels (ligne de commande, fichiers d'entrée) des clients. Cette approche est facile et rapide à mettre en oeuvre, et elle ne nécessite qu'une connaissance comme utilisateur du logiciel. Elle est applicable à n'importe quel code travaillant avec des fichiers d'entrée texte.

Le principe de cette instrumentation est d'insérer des commandes ONELAB dans le fichier d'entrée aux endroits où une interaction avec le serveur de paramètre est nécessaire. Les fichiers instrumentés ont pour nom le nom du fichier original auquel est ajoutée l'extension "`.ol`". Dans la phase d'analyse du métamodèle, ONELAB agit alors comme un **préprocesseur** et substitue aux commandes ONELAB les valeurs obtenues du serveur, délivrant ainsi un fichier d'entrée valide pour le client avec les valeurs actuelles des paramètres de simulation. Dans les fichiers instrumentés, les commandes de définition/modification de paramètres ONELAB sont placées soit dans des blocs délimités par les mots-clés `OL.block... OL.endblock`, soit sur des lignes commençant par le mot-clé `OL.line`. Les lignes de commentaire ONELAB commencent par le symbole "#"; elles sont ignorées par le préprocesseur et ne sont pas transférées au fichier converti.

Pour que la conversion soit correcte, les commandes ONELAB doivent pouvoir être reconnues par le préprocesseur dans les fichiers d'entrée de n'importe quel client, quels que soient la syntaxe et les mots et symboles réservés de ce dernier. Pour que cela soit possible, ONELAB dispose d'un mécanisme permettant la modification, chaque fois que c'est nécessaire, des marqueurs (tags) ONELAB "OL." et "#".

La syntaxe complète est détaillée sur http://onelab.info.

Les notions de métamodèle et de préprocesseur ONELAB permettent de rendre un certain nombre de services que les logiciels de simulation isolés ne sont pas en mesure de lui fournir. Le fichier de description du métamodèle, par exemple, permet de centraliser dans un même fichier les définitions de paramètres qui, sans cela, peuvent être éparpillées dans plusieurs fichiers, voire plusieurs répertoires. De plus, les commandes de branchement conditionnels permettent de structurer les fichiers d'entrée instrumentés des clients de façon à ce qu'ils décrivent, non plus seulement un problème particulier mais au contraire toute une gamme de problèmes. La couche d'interprétation ONELAB permet de construire des fichiers template beaucoup plus riches et utiles. C'est l'étape de préprocessing qui est alors chargée de générer les fichiers d'entrées spécifiques en fonction de la valeur des paramètres sur le serveur.

Communication entre codes - sockets

Lorsque client et serveur sont compilés au sein d'une même application monolithique, la communication avec le serveur peut se faire en mémoire. Dans le cadre d'une approche multicode, la communcation client-serveur doit également pouvoir se faire entre codes distincts, tournant sur le même système ou sur une machine distante. La technologie des sockets UNIX et TCP/IP permet cette communications entre process.

L'approche par socket a de nombreux avantages. Elle permet

- de créer automatiquement (et dynamiquement) une interface graphique permettant à l'utilisateur d'entrer ses données de manière interactive (par exemple, en sélectionnant à la souris les entités géométriques, des fonctions prédéfinies dans une base de donnée, etc.) ;
- de créer à la volée les jeux de données pour chaque logiciel client interfacé, ainsi que la ligne de commande pour lancer le calcul localement ou à distance avec les bons paramètres d'exécution.

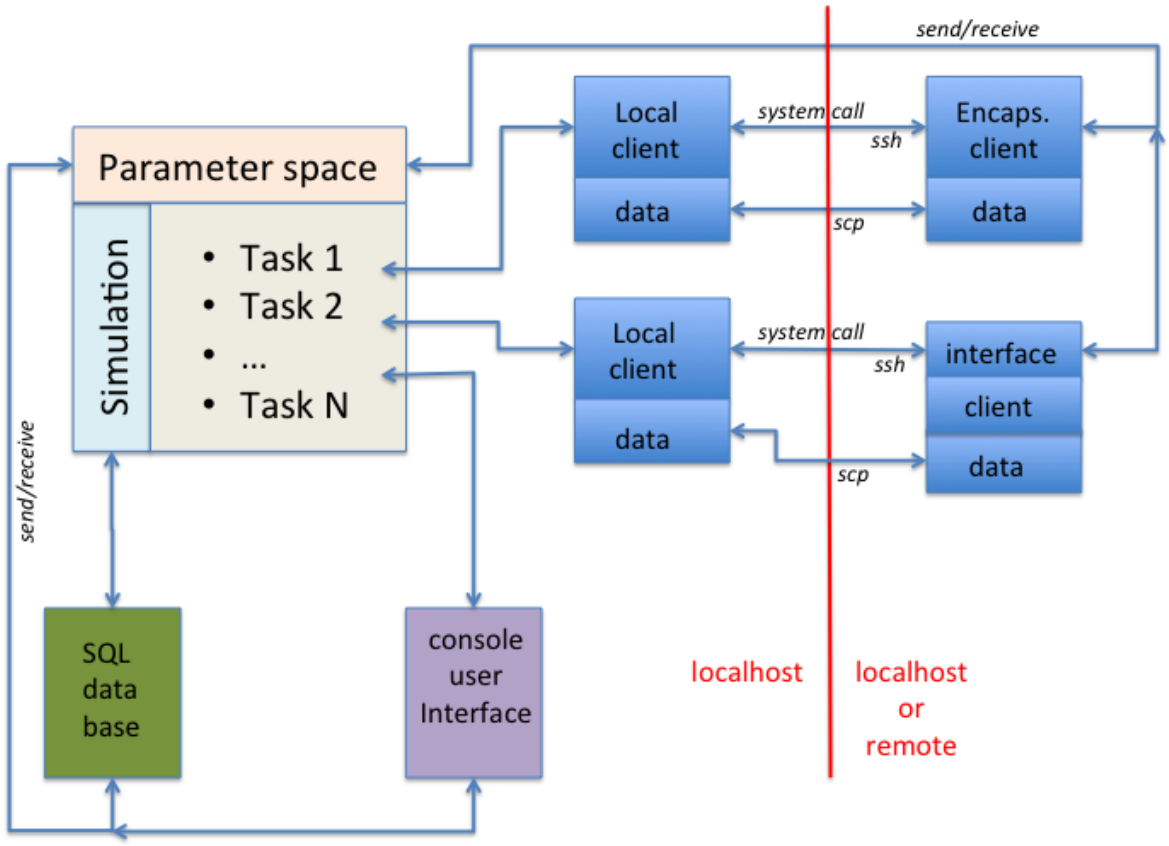


Figure 2. Représentation schématique de l'implémentation client-serveur de ONELAB avec communication par sockets. En haut à gauche le métamodèle avec son espace de paramètres unique et la liste des tâches décrivant les opérations successives de la modélisation. Chaque tâche communique avec un client local, qui, le cas échéant communique avec un client distant via un socket. La séparation physique entre programmes correspondant à des processes-système différents, ou tournant sur des machines différentes, est représentée par la ligne verticale rouge.

Communication client-serveur - la librairie onelab.h

Pour être utilisable sur le plus grand nombre de plateformes informatiques, nous avons choisi d'implémenter la librairie de base de ONELAB en C++. Cette libraire est soit utilisée par des outils ONELAB (préprocesseur, loader), soit linkée "en dur" (iOS) dans des application natives ONELAB. Cette approche permettra d'intégrer l'interface ONELAB complète dans les plateformes (iOS) où imposer l'utilisation d'un langage interprété n'est pas possible ou désirable.

Le header "onelab.h" contient le code C++ décrivant les structures de données du modèle client-serveur. La structure de serveur (classe `onelab::server`) contient une configuration de l'espace des paramètres (classe `onelab::parameterSpace`). Si la librairie est compilée en dur, les clients ONELAB communiquent directement avec le serveur ONELAB (un singleton). Si le serveur est indépendant, on utilise les mêmes appels clients/serveur, mais via un socket. Tous les clients dérivent de la classe abstraite `onelab::client`.

La spécification d'un paramètre (classe abstraite `onelab::parameter`) comprend:

- un type (variable, fonction, groupe)
- le nom des logiciels clients utilisant cette variable
- un label (optionnel)
- une aide (optionnel)
- une valeur min/max (bornes) pour les valeurs numériques, ou valeurs acceptables pour entiers et chaînes de caractères, etc.
- une catégorie, donnée sous forme d'arborescence (e.g. "Physical parameters/Magnetic Permeability" ou bien "Geometrical parameters/Width of airgap", ou encore "Advanced/ Solver parameters/Relaxation factor") et permettant l'organisation des paramètres dans la fenêtre interactive
- un flag disant si le paramètre est obligatoire ou optionnel
- un flag disant si la valeur du paramètre a été modifiée depuis le dernier calcul
- ...

C'est le serveur qui gère les problèmes de préséance (quels paramètres peuvent être modifiés par quels clients et à quel moment) et qui détermine, en fonction des paramètres modifiés depuis la dernière invocation du modèle, quels clients doivent être recalculés.

Dans la représentation graphique de l'espace des paramètres, l'arborescence décrite dans la catégorie peut mener e.g. cacher certains paramètres avancés. Les classes `onelab::number`, `onelab::string`, `onelab::region` et `onelab::function` dérivent toutes de la classe `onelab::parameter` et y ajoutent chacune leurs spécificités.

Le code source de l'interface abstraite est disponible dans `gmsh/Common/onelab.h`.

Communication utilisateur-métamodèle - Interface graphique

L'implémentation de ONELAB a été réalisée dans Gmsh, avec une interface graphique basée sur FLTK (http://www.fltk.org). Dans cas, Gmsh est à la fois un serveur et un client, ce qui permet d'agir de manière unifiée à la fois sur les paramètres de géométrie et sur les paramètres des solveurs.
L'interface graphique du serveur ONELAB dans Gmsh est implémentée dans `gmsh/Fltk/onelabGroup.{h,cpp}`.

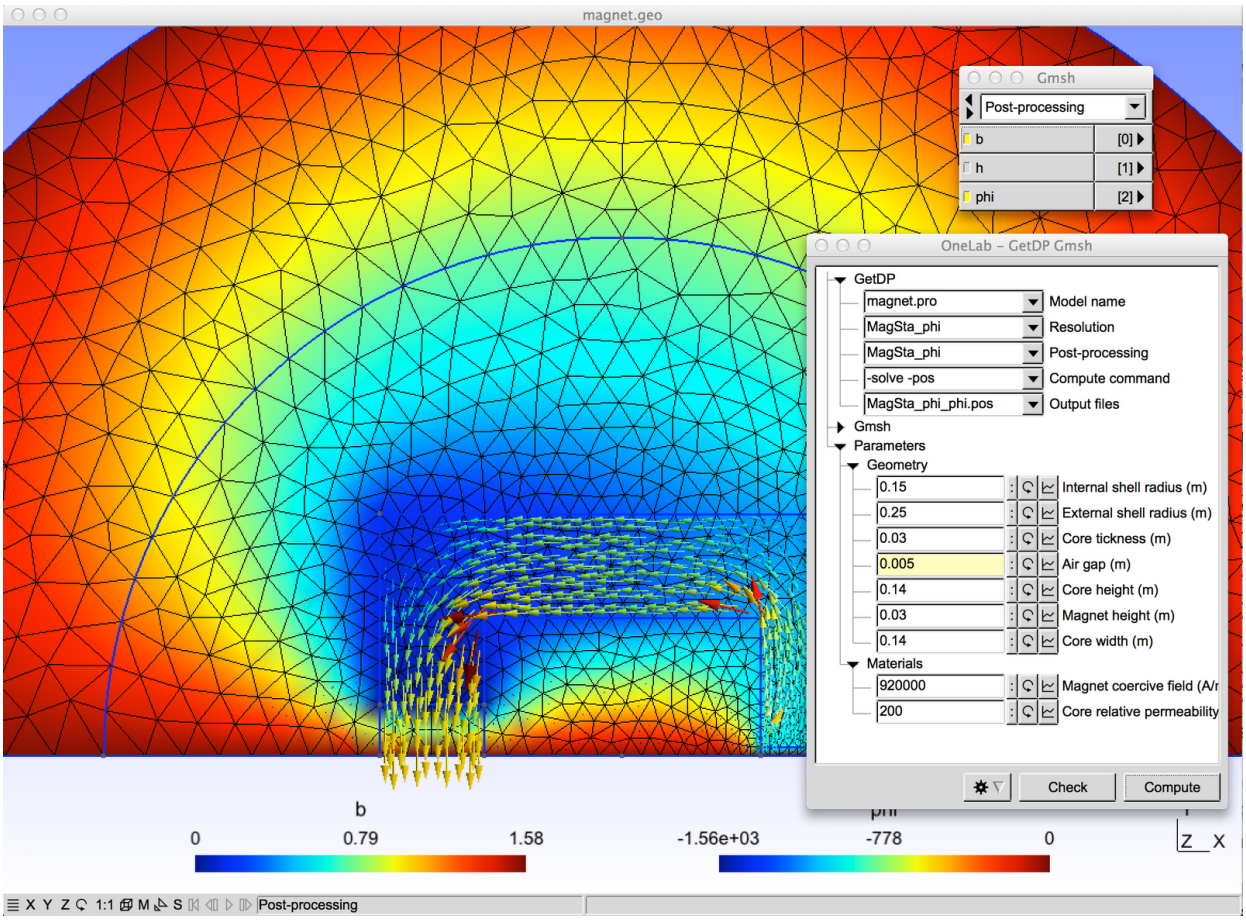


Figure 3. Interface graphique au-dessus de `oneLab::server` dans Gmsh. Le serveur pilote ici deux clients : Gmsh et GetDP. Les paramètres “Internal shell radius” et “External shell radius” sont utilisés à la fois par Gmsh et GetDP.

Distribution

Un aspect essentiel du projet est celui des modalités permettant de mettre les métamodèles à la disposition des utilisateurs. Plusieurs approches sont possibles qui ont chacune leurs avantages et leurs inconvénients.

Installation par l’utilisateur

La première solution est de guider l’utilisateur dans l’installation de Gmsh et des différents clients sur son propre système. C’est l’installation la plus efficace en terme d’espace mémoire et de temps de calcul, mais elle nécessite un certain niveau de compétence informatique que n’ont pas nécessairement tous les utilisateurs du public cible du projet. Cette approche est utilisée avec les étudiants dans le cadre d’un cours donné à l’ULg (100 étudiants de niveau 3ème bachelier), de même que pour tous les utilisateurs du logiciel GetDP.

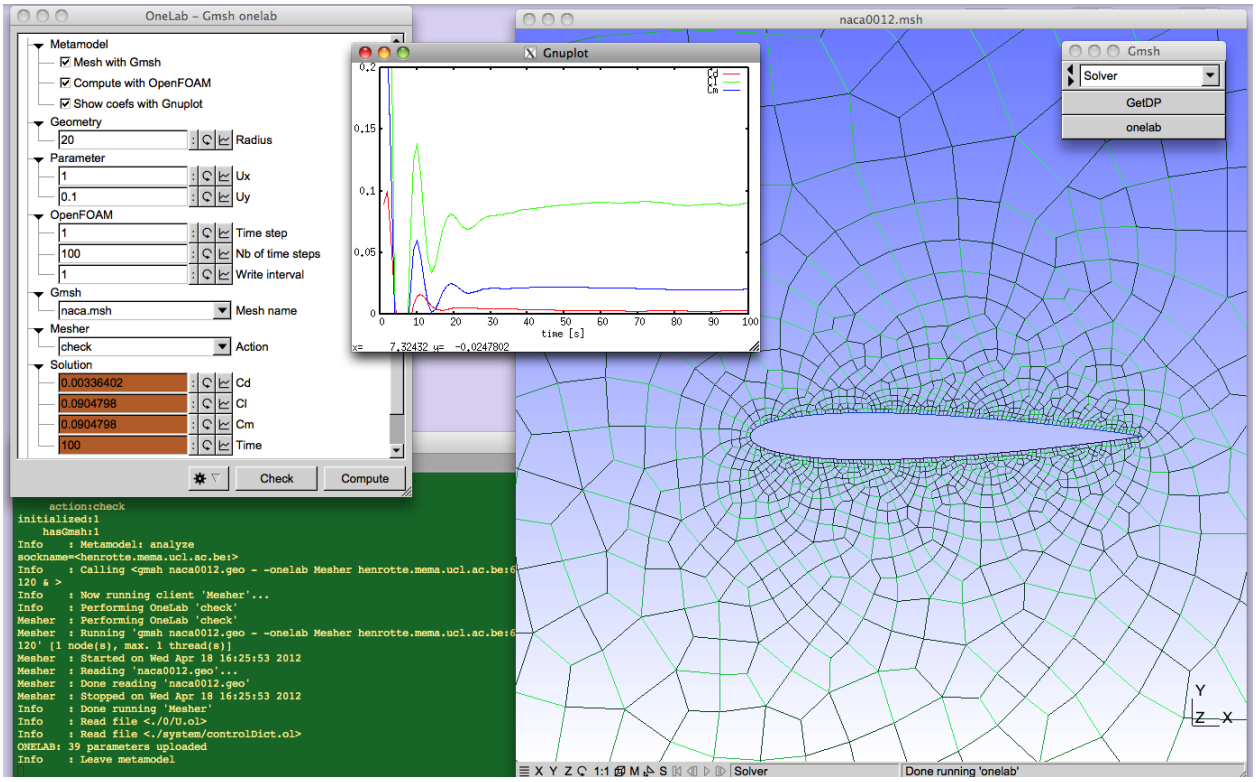
Machine virtuelle

Avec cette approche, l'utilisateur doit seulement installer un hôte de virtualisation (p.ex. Virtualbox) et télécharger un fichier "appliance" contenant un système d'exploitation virtuel sur lequel ONELAB et tous les clients ont été installés et testés au préalable. On a de cette façon une indépendance totale par rapport aux caractéristiques de l'ordinateur de l'utilisateur mais, en contrepartie, le fichier à télécharger est relativement volumineux (de l'ordre de 1 à 2 Gb). Cette approche a été utilisée avec des étudiants dans le cadre d'un cours donné à l'UCL.

Serveur d'application distant

Une troisième approche consiste à installer uniquement Gmsh sur l'ordinateur de l'utilisateur et d'accéder aux clients sur un serveur d'application remote. Cette approche nécessite de disposer des droits de connexion sur une machine distante.

T3.2 Clients Mécanique des Fluides - OpenFOAM



T3.3 Clients Mécanique et Thermique - Code_Aster

Le logiciel multi-physique ELMER () a été utilisé jusqu'à présent pour les simulations dans le domaine des structures. Son interfaçage a été réalisé sur le modèle expliqué à la section précédente. L'interfaçage effectif avec Code_Aster se fera effectué lors de la prochaine étape du projet.

T3.4 Clients Electromagnétisme - GetDP

Le modèle abstrait du problème est déjà très structuré dans le logiciel GetDP : il n'est dès lors pas utile de le redéfinir par ailleurs dans un "driver" externe.

Pour GetDP, nous avons donc décidé:

- d'exploiter directement les champs DefineConstant, DefineGroup et DefineFunction du langage .pro : en modifiant légèrement le parseur, GetDP échange désormais les paramètres définis dans DefineConstant avec le serveur ONELAB. DefineFunction et DefineGroup sont encore à implémenter.
- A chaque lecture du fichier de donnée .pro GetDP interroge le seveur ONELAB, ce qui permet une gestion simple des erreurs, et une construction dynamique de l'espace des paramètres possibles (en fonction de "flags" donnés, certaines options deviennent accessibles ou non). Avec cette approche, gérer les dépendences entre paramètres dans la couche experte est inutile : c'est le modèle template .pro qui s'en charge. (Comme mentionné précédemment créer une couche abstraite qui garantit que le problème est bien posé est très ardu---cf. efforts du groupe de recherche du Prof. Kettunen de la Tampere Technical University dans ce domaine; cf. également Eficas, le "structurateur" de définition de problème utilisé par Code_Aster.)
- GetDP transmet également au seveur ONELAB une liste d'opérations possibles en fonction des choix ci-dessus (noms de "résolutions" et noms des opérations de post-pro disponibles)

Syntaxe dans le langage de GetDP :

- `DefineConstant[ CircuitCoupling ];`
- `DefineConstant[ CircuitCoupling = 0 ];`
- `DefineConstant[ CircuitCoupling = {0, Choices {0,1}, Label "Enable circuit coupling?"} ];`

Cette syntaxe a également été intégrée dans le language de définition de géométrie (.geo) de Gmsh. Grâce à cette syntaxe commune, un métamodèle peut dès à présent être écrit dans les languages respectifs de GetDP et Gmsh, et inclure des fichiers de définition de paramètres communs.

La figure 4 montre un méta-modèle en cours de développement en partenariat avec Alstom Transport pour la modélisation d'inductances HF dans des busbars de puissance.

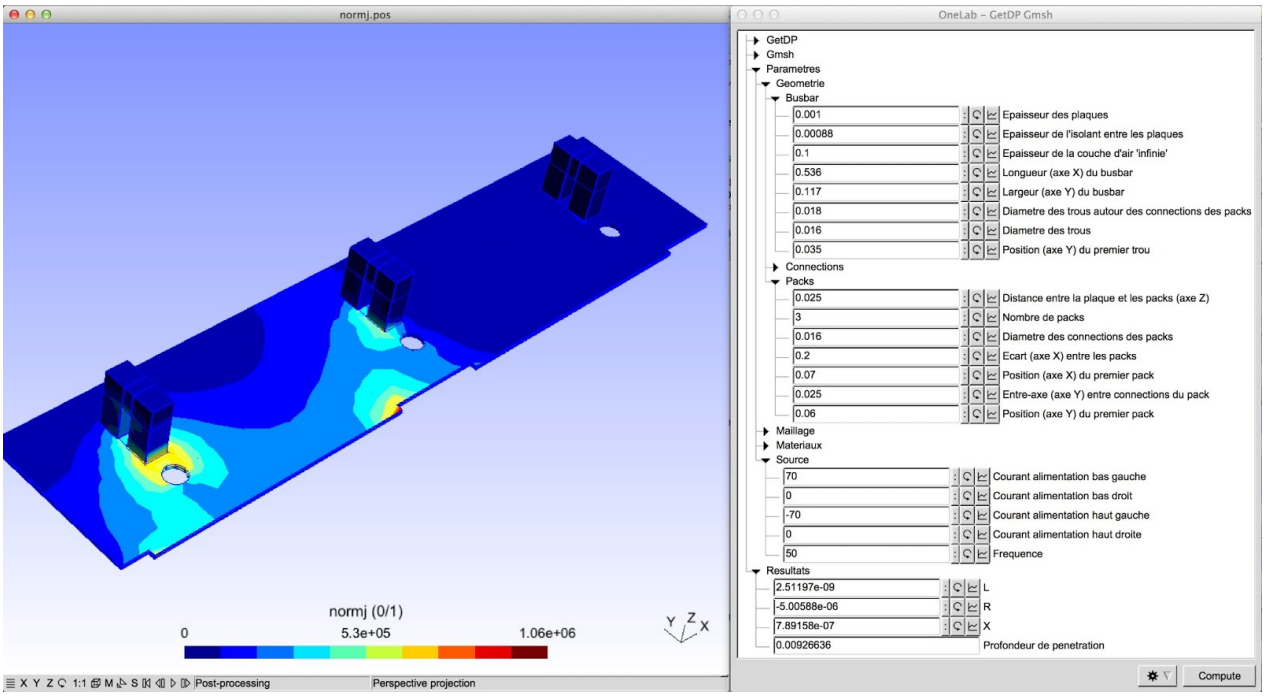


Figure 4. Exemple de métamodèle interfactif Gmsh/GetDP : busbar pour une alimentation de puissance haute-fréquence.

T4 Interface post-traitement

L'interface abstraite de post-traitement a été étendue pour supporter les nouveaux types de données des logiciels clients, principalement MED3.

Cette interface est présentée dans l'annexe C.

T5 Documentation et validation

Le travail de rédaction de la documentation vient de débuter. Il est réalisé de manière collaborative sous forme d'un wiki à l'adresse http://onelab.info.

Ce chapitre contiendra à terme un bref résumé des modèles développés.