**RAPPORT PROJET JAVA**

Auteur : Aurélien Reysset

Date dernière modification : 05/06/2014

Contenu :

Dans ce rapport nous faisons un bilan dans une première partie de la réunion visant à définir les « modèles » de base et la façon de les composer. Ainsi une classe a été définie, comportant des attributs et des méthodes utiles au projet.

Dans une seconde partie la définition du format d’entrée/sortie à traiter par les diverses fonctions Matlab (appairage, étude de singularité, composants fortement connexes, rupture des boucles algébriques) seront abordés de manière à pouvoir partager le travail de développement Java/Matlab.

**1 – Bilan sur les modèles de base :**

Il a été décidé que la brique ou « modèle » (au même titre qu’une composition) fait partie de la classe Modèle définie de la manière suivante :

|  |
| --- |
| Classe Modèle |
| Attributs :   * nom * <liste de paramètres> (≥1) * <liste de équations> (≥0) * <liste d’adresse de modèles insérés> (≥0) * information |
| Méthodes :   * Ajout\_modele * Ajout\_paramètre * Ajout\_equation |

Concernant les paramètres, ils sont constitués des champs suivants :

|  |
| --- |
| Paramètre |
| Champs :   * nom * type (input, undetermined, output) * sous-type (input: constant/range/set, undetermined: input\*/free/output\*, output:-) * valeurs (pour une constante 12.4 une seule valeur, pour un range un intervalle continu [1.0 ;15.3[, pour un set un ensemble de valeurs {1 ;2.3 ;5 ;18}, vide dans les autres cas). * propriété (constante/hypothèse/référence/gamme…> point à mieux définir, mais cela servira au classement dans le fichier Excel et à l’utilisation de filtres). * unité (‘m/s’,…) * information (‘couple électromagnétique….’) |

Concernant les équations :

|  |
| --- |
| Equation |
| Champs :   * contenu (‘y=a\*x+b’) * type (equation, contrainte) * sous-type (équation : orientée/non\_orientée, contrainte:-) * parameter\_attribué (orientée: mettre le parameter en question ‘y’ ici, non\_orientée:-) * propriété (loi d’échelle/régression polynomiale…> point à mieux définir, mais cela servira au classement dans le fichier Excel et à l’utilisation de filtres) * information (‘équation électrique du moteur….’) |

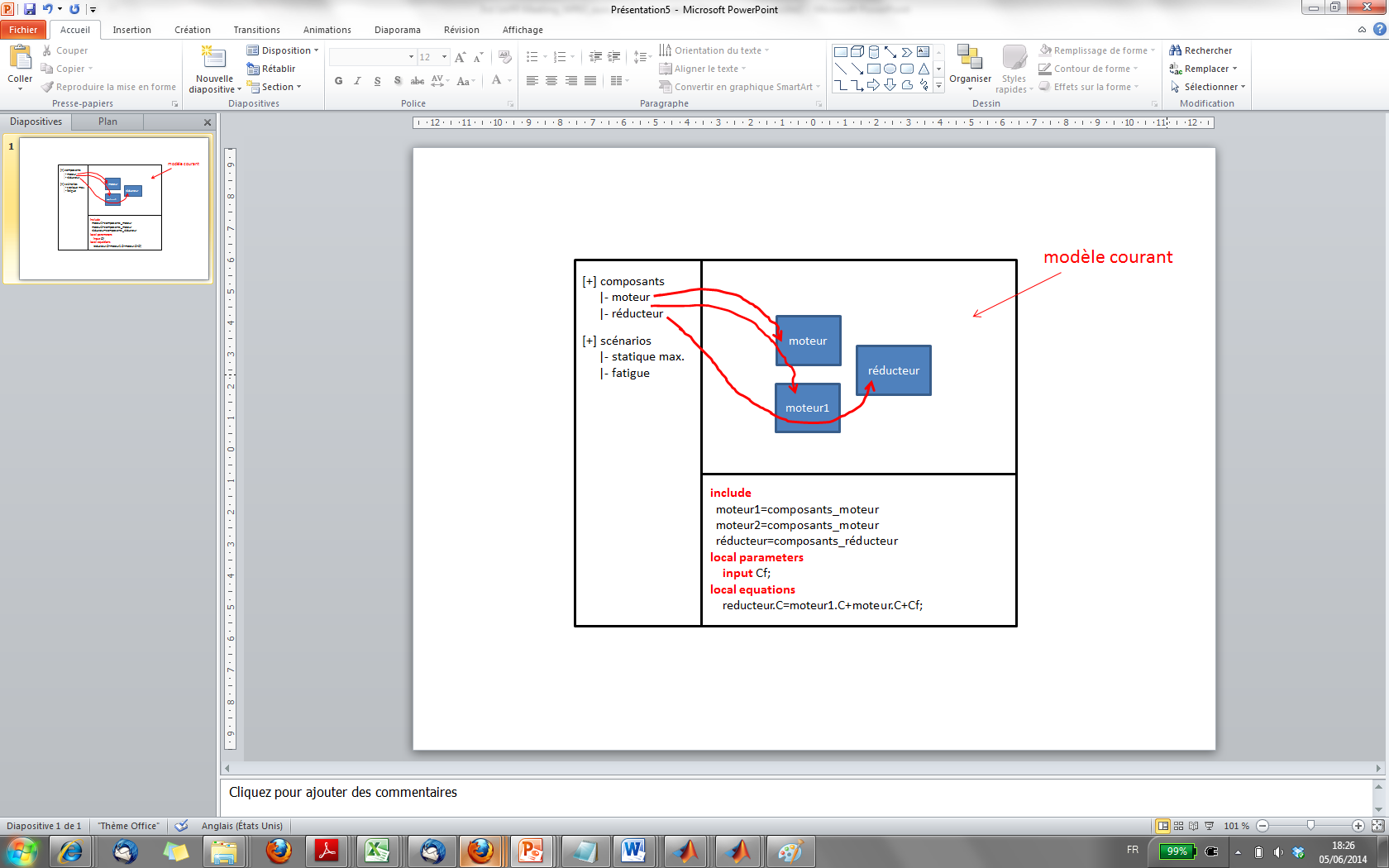
Pour les équations, il a été décidé que seules quelques règles de validation seront à définir de manière à valider l’équation, imposer trop de règles est non pertinent car l’on aurait alors pu utiliser une grammaire. Il ne faut pas perdre de temps là-dessus.

De plus, il est théoriquement faux de rentrer deux fois la même équation, ou une combinaison d’équations existantes car le problème est alors sur-contraint. Pour un système d’équations non-linéaires ce critère est assez compliqué à valider. Nous ne vérifierons donc que les doublons.

Dans le cas où le problème serait sur-contraint, le fait de modifier le type équation>contrainte permet de rendre le problème solvable et l’on sait que la contrainte sera toujours validée. Néanmoins, si à la base le projet et d’ores et déjà sur-contraint par nature, le fait d’ajouter une combinaison d’équations déjà enregistrées risque de fausser la donne quant à la modification de typage (et la vérification d’appairage contrainte-entrée « variable »).

Composition de modèles :

Par composition de modèle on empiète un peu sur la partie interface graphique

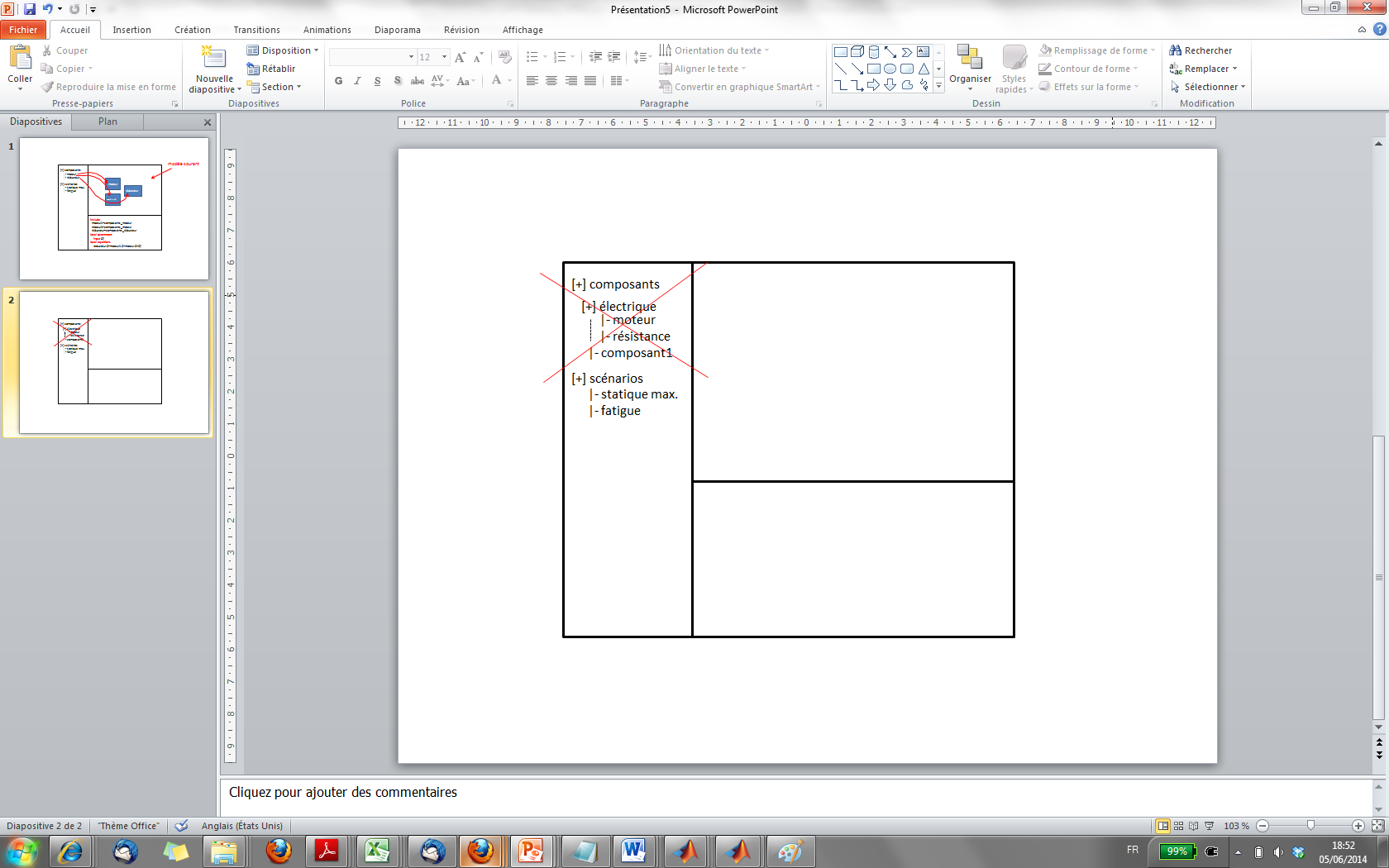


L’idée est la suivante : la validation du bloc se fera en cliquant sur un bouton du genre « compile » qui fera une mise à plat du code en réécrivant les équations en renommant les paramètres compris dans les sous-modèles (ex : dans moteur U=R\*I+k\*w deviendra moteur\_U=moteur\_k\*…). Les paramètres des équations ajoutées au modèle courant et n’apparaissant dans aucun des sous-modèles seront alors ajoutés à la liste des paramètres à la racine du modèle (ici dans « local parameters »).

Problème mis en avant :

Nous nous sommes posé la question quant à l’enregistrement des modèles dans des packages, des sous-package… l’intérêt étant de pouvoir les réorganiser comme bon nous semble et de partager des packages (à charger) avec les industriels (avec éventuellement une librairie de base). Il nous serait alors possible de dupliquer un modèle dans un autre package, d’en modifier le contenu…

Néanmoins, même si cela est possible pour ce qui concerne l’enregistrement, il semblerait que pour la gestion de l’arborescence dans l’interface graphique l’encapsulation d’un package dans un autre doit être limité à un seul niveau pour simplifier le codage. Il n’existe pas de méthode générique. De plus la représentation sur un même niveau de sous-package et modèles ne semble pas possible (cf. figure ci-dessous).



**2 – Interface avec Matlab/Excel :**

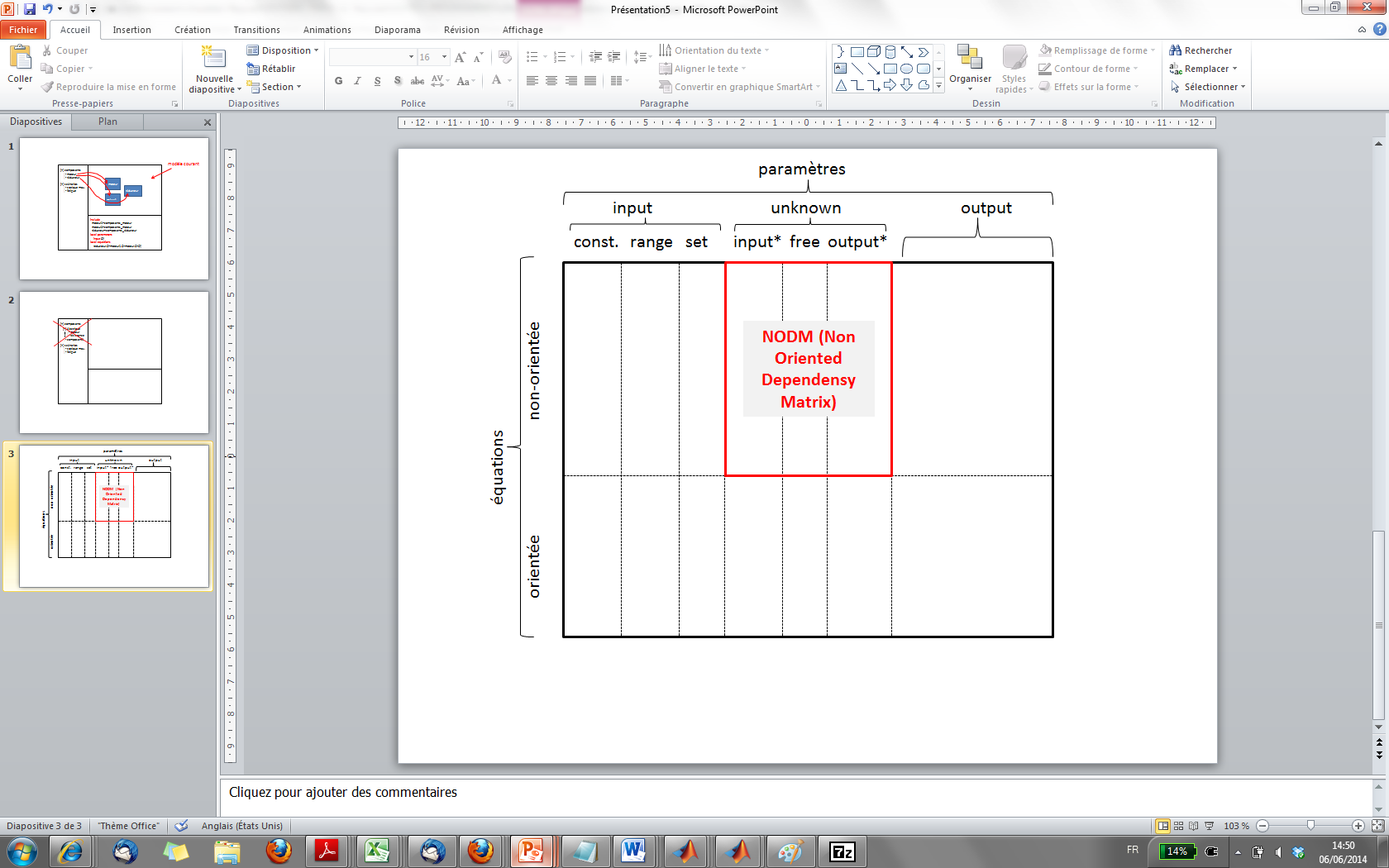
L’une des premières étapes avant de s’interfacer avec Matlab est de mettre à plat le problème, pour ce faire, il sera nécessaire de renommer les paramètres par leur « chemin d’accès », comme expliqué précédemment avec sursurmodele\_surmodele\_moteur\_U… Ca a tendance à alourdir l’équation à la lecture, mais aucune erreur n’est permise et l’on voit bien la notion d’appartenance. Pour la saisie des équations, il faudra aider à l’écriture. Par exemple je rentre ‘m’ j’appuie sur tab et l’interface me propose d’insérer ‘m’ ou ‘moteur’ ou ‘moteur1’, je peux alors descendre et taper entrer j’obtiens ‘moteur’, j’appuie sur underscore et tab et il me propose la liste de paramètres du modèle ‘moteur’, soit C/J/M…

Je ne l’ai pas dit mais en commençant mon équation j’aurais pu débuter par un tab et il me proposait ‘’, ‘moteur’, ‘moteur1’ et ‘réducteur’. Ceci sous-entend qu’il arrive à analyser ce que l’on tape et si l’on est dans un modèle…

Ceci n’est peut-être pas si aisé que ça, dans ce cas on peut éventuellement écrire les équations supplémentaires au moyen de liens tirés entre sous-modèles, et dans ce cas on a accès uniquement aux paramètres de ces 2 modèles ou des nouveaux (ex : Cf).

Bref, après la mise à plat, on a des équations et des paramètres pour faire simple.

On va créer une matrice sub-divisable, avec des regroupements par type/sous-types des lignes et des colonnes. En sachant qu’en ligne on mettra les équations et en colonne les paramètres comme suit :



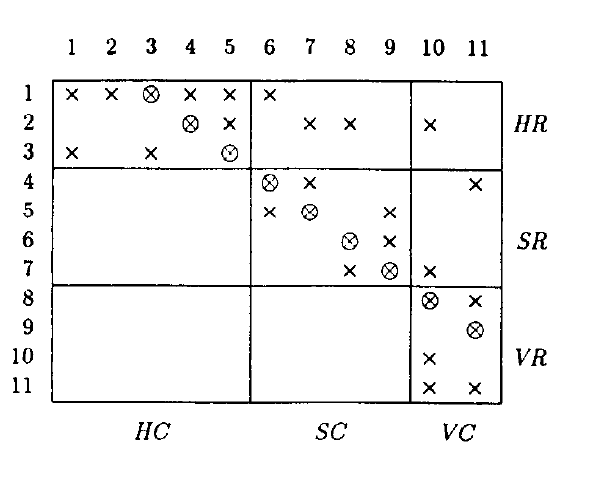
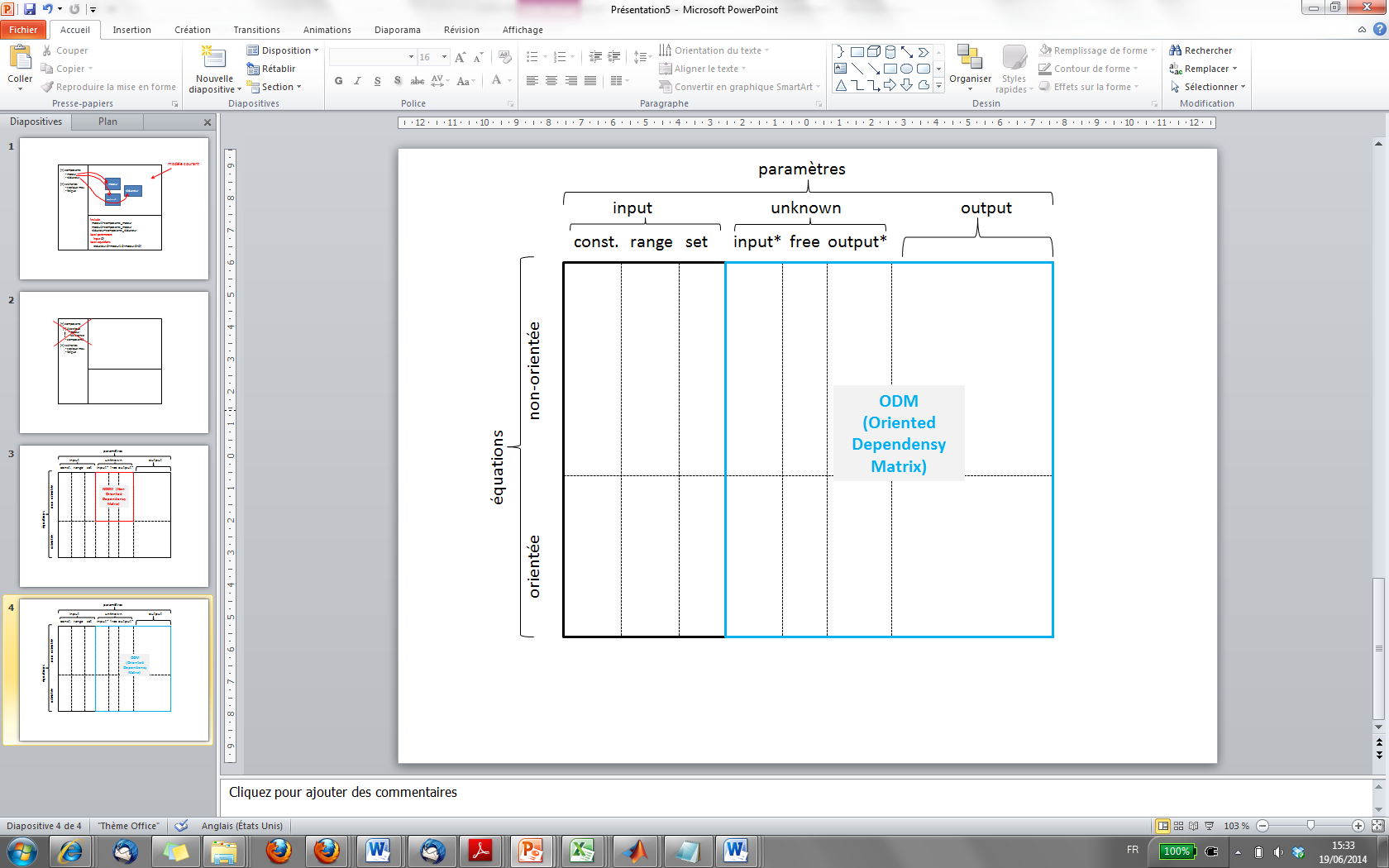
Pour la première étape d’appairage, il s’agira de séparer les inconnues/outputs des paramètres typés input et les équations non-orientées des orientées.

/!\ Mais attention quant à la définition d’output dans la matrice ci-dessus. En effet certaines équations sont définies comme orientées car en général difficilement inversible (ex : méta-modèles) et un paramètre de sortie leur est affecté, dans ce cas le paramètre est un output, il n’y a pas de problème. Mais il arrive aussi que des paramètres soient définis comme output (donc impossible à définir en entrée) sans être à priori affectés à une seule équation. Dans le cas où il n’apparait que dans une seule équation, là non plus aucun problème, c’est un output et l’équation est orientée. Mais dans les autres cas, il faut le considérer comme output\*.

La matrice NODM enregistrée dans le fichier « matching\_in.txt » contiendra des 0 et des 1. La valeur ‘1’ signifiera que le paramètre peut être défini comme une sortie de l’équation, ‘0’ que cela n’est pas possible. Il faut donc voir qu’il y a deux conditions : la première que le paramètre soit dans l’équation, la seconde qu’il ne soit pas encapsulé dans une fonction non inversible (bijective).

En sortie de l’étape d’appairage, l’on obtient dans le fichier « matching\_out.txt » la matrice ODM (Oriented Dependency Matrix), qui en réalité est une sous-matrice de ODM, et qui contiendra un maximum de un ‘2’ par ligne signifiant que le paramètre de la colonne correspondante est défini comme étant la sortie de l’équation, alors qu’un ‘1’ signifie seulement que le paramètre est une entrée locale de l’équation.

Le reste de la matrice globale présentée dans la figure précédente aura les mêmes informations.

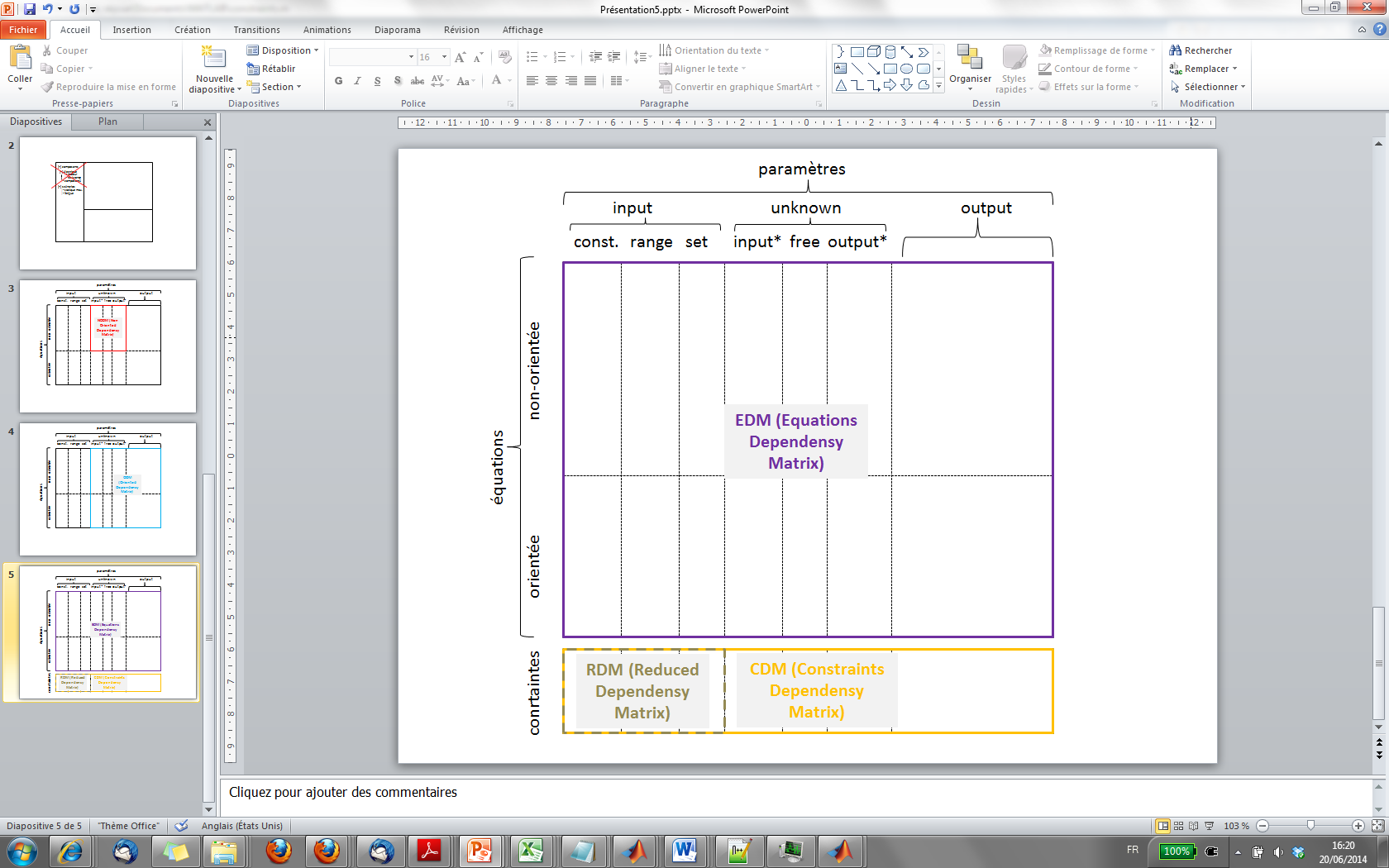


L’étape d’après consiste à effectuer des permutations de lignes/colonnes de la matrice globale mais essentiellement de la sous-matrice ODM, de manière à retomber sur le découpage en matrice sous-contrainte HC-HR sur la figure que l’on renommera UCC-UCR : UCM (Under-Constrained-Column/Row/Matrix) ; en matrice normalement contrainte SC-SR sur la figure, renommé NCC-NCR : NCM ; et la matrice sur-contrainte VC-VR, renommée OCC-OCR : OCM.

Pour ce faire on procèdera de la même manière que pour le matching, en passant la matrice ODM dans un fichier singularity\_in.txt et en récupérant dans singulaity\_out.txt deux vecteur RO=[… pour ordre des lignes et CO=[… pour l’ordre des colonnes .

C’est après cette étape que de l’aide est apportée à l’utilisateur quant à la qualité du problème posé.

Vient ensuite une étape complémentaire : le chainage potentiel entre input variables (dans un intervalle ou un ensemble) et contraintes. On va vérifier qu’il existe bien au moins une entrée variable sur laquelle jouer pour valider la contrainte, et un appairage maximal sera aussi réalisé de manière à savoir si les contraintes peuvent être validées indépendamment. Pour ce faire, il est nécessaire de fournir plusieurs informations : la matrice de dépendance des équations (EDM), le vecteur d’indice des inputs variables (VIC-Variable Input Column) sur un intervalle continu ou discret, le vecteur d’indice des inputs constants (CIC-Constant…) et la matrice de dépendance des contraintes (CDM) comme le montre la figure suivante :



Une fois le problème correctement posé (correctement contraint), il s’agit de vérifier qu’aucune boucle algébrique ne persiste, si c’est le cas, il faut apporter une aide à l’utilisateur pour déplacer ces boucles au niveau de la surcouche d’optimisation en supposant des paramètres comme des entrées connues mais pouvant varier dans un intervalle, et l’équation permettant de les évaluer comme étant une contrainte. Mais pour déterminer ces « sets » de paramètres à poser, il s’agira d’appliquer la fonction loops.exe sur le fichier loops\_in.txt contenant la matrice ODM.

Dans le fichier loops\_out.txt, seront listées les combinaisons de paramètres à fixer pur chaque composante fortement connexe.

En conclusion, 4 programmes différents sont à utiliser, dans l’ordre : matching.exe, singularity.exe, puis une fois le problème correctement posé, constraints.exe et loops.exe. Ces fonctions sont à utiliser en mode démonstration, mais seront soit à recoder, soit nécessiteront l’achat de la toolbox de compilation pour le prototype final. Ça nous sert juste de validation sur le prototype sans avoir pour autant l’outil Matlab.