Réunion MORSE du 14/11/03

1. Feuille de route pour le travail attribué à Aonix

1. **Etudier et analyser les concepts LfP**
2. **Transformation d'UML vers LfP**
   a) définir les règles de transformation d'UML vers LfP
      Pb: concepts UML non représentables dans LfP (entités du modèle statique : classes passives, paquetages, associations, rôles, cardinalités, généralisations, dépendances, stéréotypes, étiquettes, etc.). Ces concepts devront soit être ajoutés au langage LfP, soit faire l'objet de restrictions d'utilisation d'UML.
   b) définir un profil UML permettant d'exprimer tous les concepts de LfP :
      média, *binders*, *ports* pour les diagrammes d'architecture de LfP,
      protocole (enchaînements d'opérations), *appels d'opérations et retours* (dans les diagrammes d'états ou d'activités), *déclarations*, *invariants*, *pseudo-code*, ..., pour les diagrammes de comportement de LfP.

      modèle UML ➡ **AMEOS** ➡ *(format interne AMEOS)* ➡ **ACD** ➡ modèle LfP *(format XML)*

      règles de transformation

      Enrichissement d'AMEOS/ACD pour supporter 2.0 (en particulier diag. pertinents pour la génération LfP)
3. **Ecrire une première ébauche du guide méthodologique**
4. **Implémenter le profil avec AMEOS**
5. **Implémenter les règles de transformation avec AMEOS / ACD**
   a) étudier et analyser la forme XML des modèles LfP
   b) développer les templates ACD pour la génération de modèles LfP exprimés en XML
6. **Intégration à ECLIPSE**
   Invocation d'AMEOS / ACD. La chaîne de transformation serait :

   modèle UML ➡ **AMEOS** ➡ *(format interne AMEOS)* ➡ **ACD** ➡ modèle LfP *(format XML)*

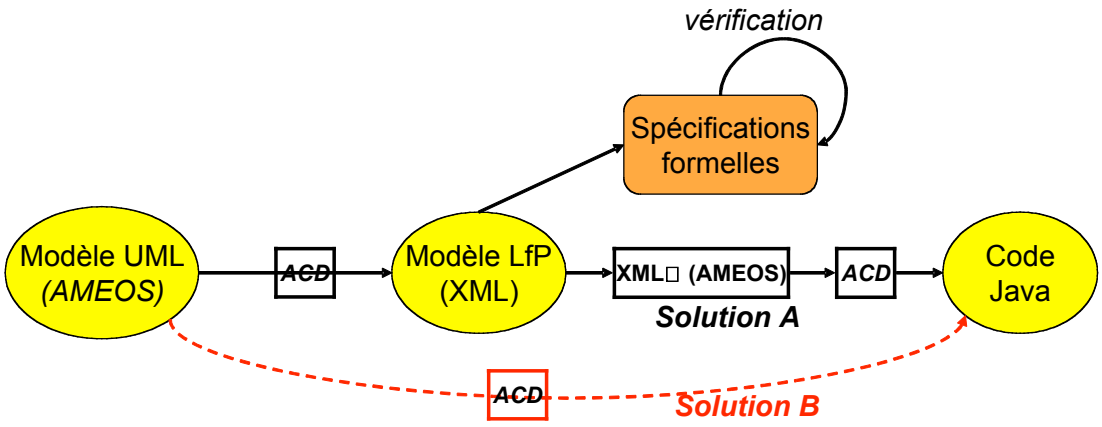
   (remarque : le modèle UML en entrée peut être élaboré au moyen d'AMEOS, mais il peut aussi être importé par AMEOS sous forme XMI – une possibilité pour ceux qui utilisent un outil UML autre qu'AMEOS)

   Question : Coordination de l'intégration des outils à Eclipse : comment procèdera-t-on ?

A la fin de cette étape, compléter le guide méthodologique.

7. **Définir des règles de transformation du modèle (LfP ou UML) vers Java**

2 solutions :



Remarques pour la solution A :

Les informations du modèle UML de conception nécessaires à la génération automatique de squelettes de code (entités du modèle statique : classes passives, paquetages, associations, rôles, cardinalités, généralisations, dépendances, stéréotypes, étiquettes, etc.) peuvent-elles toutes être propagées dans le modèle LfP ? dans l'état actuel de LfP, la réponse est non (il manque notamment les classes passives).

Deux options semblent a priori possibles pour combler ce manque :

- soit enrichir le langage LfP en y ajoutant l'équivalent des entités UML manquantes,
- soit prendre en entrée de la transformation le modèle LfP enrichi d'une partie des informations du modèle UML (sous forme textuelle – XMI ?)

Remarques pour la solution B :

En toute logique, pour générer le code Java, il est possible de court-circuiter la phase de transformation UML-vers-LfP, puisque le modèle UML est censé contenir la même somme d'informations que le modèle LfP, grâce au profil spécifique qui a été défini. On a bien précisé en effet que le modélisateur ne doit jamais avoir besoin d'enrichir ni même de connaître la forme LfP de son modèle. Donc il ne manque rien au modèle UML, il doit être aussi complet que le modèle LfP.
La transformation UML-vers-LfP reste néanmoins utile, car le modèle LfP sera utilisé pour générer des spécifications formelles et pour permettre la vérification formelle du modèle.

A la fin de cette étape, compléter le guide méthodologique.

8. **Implémenter les règles de transformation de LfP ou UML vers Java (avec ACD)**

A la fin de cette étape, compléter le guide méthodologique.

2. Questions

a) Les classes LfP sont actives (correspondent à un *thread* ou un *process* UML). Or le concepteur UML utilise très fréquemment des classes passives. Si le modèle UML est conçu dans la perspective d'une génération de code dans un langage orienté objet, ces classes passives se traduiront plus tard dans le code par des classes du langage. En LfP ces classes passives ne vont plus être représentées par des classes, mais par des "déclarations" de données et d'opérations, fondues au sein d'une classe LfP, une "grosse" classe active. Dans le code généré, si on génère à partir de LfP, on ne retrouvera plus de manière claire les classes passives du modèle de conception. Par conséquent, le programmeur, qui doit compléter les squelettes du code généré, peut avoir quelques difficultés à faire le lien entre le modèle de conception et le code généré.

Mais d'autres problèmes se posent aussi : la modularité du modèle de conception n'est pas conservée dans le code. Ni l'encapsulation : si les données et opérations de plusieurs classes passives sont introduites dans le code correspondant à une même classe LfP, le respect des restrictions sur la visibilité de ces données et opérations ne sera plus contrôlable par le compilateur : le programmeur pourra en complétant les squelettes faire indûment référence à des déclarations qui devraient être inaccessibles, car *privées* dans l'esprit du concepteur.

Des problèmes d'homonymie peuvent aussi survenir si deux classes passives UML possèdent des attributs ou des opérations de même nom, ou si deux paquetages UML incluent des classes passives de même nom : cela contraint la transformation UML-vers-LfP à modifier les noms choisis par le concepteur, pour éviter l'homonymie des déclarations au sein d'une même classe LfP.

Quelles solutions LfP peut-il proposer pour résoudre ces problèmes ? peut-on ajouter au langage LfP de nouveaux concepts, analogues aux classes passives et aux autres entités UML qui s'y rattachent (associations, rôles, paquetages, etc.) ? ou bien devrons-nous ajouter au modèle LfP obtenu à partir d'UML toutes les informations qui permettront de reconstituer ces classes au moment de générer les squelettes du code ?