Description du module INTERFACE/ACQUISITION rev.1

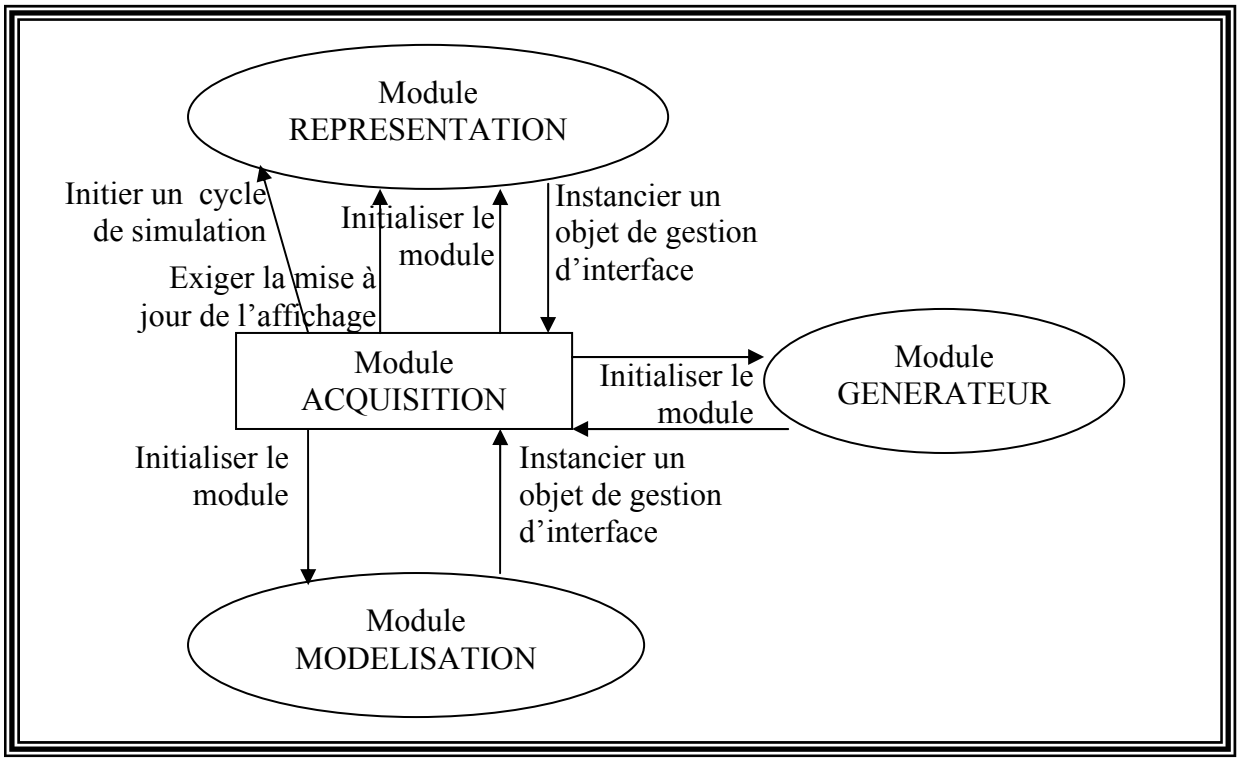
1. Description du rôle du module

Rev.1 : les rôles sont définis plus précisément.
Les rôles de ce module sont multiples : il doit

- Initialiser l'applet,
- Gérer les interactions avec l'utilisateur, à savoir :
  - Fournir à l'utilisateur un moyen simple et efficace de définir la configuration souhaitée pour la simulation,
  - Permettre à ce même utilisateur de visualiser la configuration courante,
  - Lui permettre également de visualiser le résultat de la simulation (la majeure partie de ce travail, ie la mise en forme des données selon la forme souhaitée par l'utilisateur est assurée cependant par le module représentation.),
- Piloter la simulation, sous la direction de l'utilisateur (début/arrête de la simulation, « rythme » de la simulation…),
- Fournir un moyen d'interfacer les différents modules, ou du moins d'initier les « communications ».

2. Moyens mis en œuvre

A. Représentation visuelle



B. Initialisation

Dans une phase d'initialisation, le module créera les objets principaux des trois autres modules : **CModuleReprésentation**, **CGénérateur**, et **CModèleComposé**. Le reste des initialisations aura lieu au fur et à mesure des demandes de l'utilisateur.

C. Echanges standardisés

La plupart des échanges avec les autres modules seront effectués au travers d'une interface standardisée, qui permettra d'obtenir ou de modifier un nom identifiant un objet, d'obtenir un ou des panneaux affichables (déclinés par thème : configuration, statistiques, aide…) spécifique à l'objet concerné.
Rev.1 : les thèmes sont abandonnés : complexité supplémentaire, gain minime.

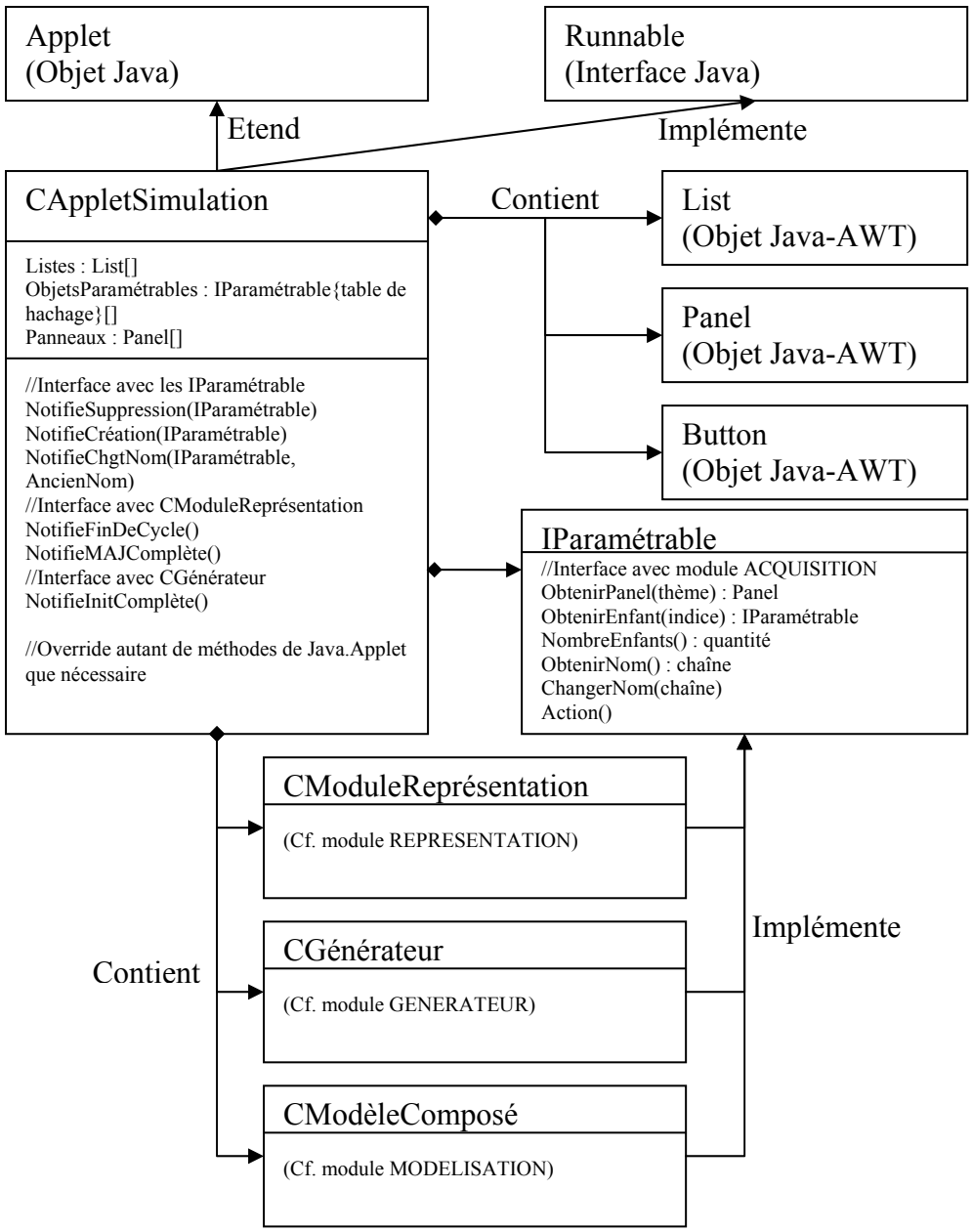
D. Pilotage de la simulation

Par ses échanges avec l'objet qui intancie la classe CModèleReprésentation, le module pilotera la marche de la simulation : c'est lui qui est chargé de donner l'impulsion qui démarre un cycle interne de la simulation, et l'impulsion qui génère l'affichage des résultats courants.
Rev.1 : l'impulsion d'affichage est pour l'instant abandonnéee, on lui préfère une mise à jour automatique entre chaque cycle, dont on peut préciser le nombre de points générés.

E. Interaction avec l'utilisateur

L'objet principal du module mettra en place une interface générale, comprenant un emplacement variable en fonction de l'objet sélectionné, dans une liste.
Des boutons séparés permettront le pilotage de la simulation par l'utilisateur prévu au paragraphe précédent.
Le panneau variable sera défini indépendamment dans chacun des objets configurables, et mis à disposition du module interface par l'intermédiaire de l'interface prévue pus haut.

3. Diagramme de classes



4. Description des classes et interfaces

A. La classe CAppletSimulation

Représentée par un unique objet créé par le chargeur d'applet, elle étend Java.applet, et devra implémenter l'interface runnable. Elle sera donc le thread principal de l'applet.
Du point de vue graphique, cette classe est un Panel, qui est lui-même un Container, qui est lui-même un Component : cela signifie que l'objet est un composant graphique (Component), qui peut contenir d'autres composants graphiques(Container), et les organiser dans un panneau (Panel) à l'aide d'un LayoutManager. Nous utiliserons ici un GridLayout, qui permet

d'organiser les composants simplement sur une grille : ceci a le mérite de faciliter la programmation en évitant l'utilisation de coordonnées graphiques dans les zones clientes des fenêtres, tout en assurant que l'applet aura à peu près le même aspect sur toutes les stations.
Les différents objets des autres modules qui implémentent l'interface IParamétrable seront accessibles à l'utilisateur via des listes (List)
Différentes notifications doivent jalonner les différentes étapes de l'initialisation et de la simulation, et permettre la mise à jour des listes affichées, (*rev.1*)ainsi que les dialogues entre modules.

B. L'interface IParamétrable

1. La méthode ObtenirPanel()

Elle doit renvoyer au module un panel sur le thème demandé (configuration, statistique, aide, graphique…) qui pourra être soit enfiché dans le Layout de l'applet, soit dans une Frame ou une Window (fenêtres à part avec ou sans menu)
Rev.1 : thèmes abandonnés ; tous les objets implémentant l'interface seront insérés à leur création dans le panel dédié, auquel sera associé un CardLayout(cf java awt) qui permet ensuite d'en afficher un à la demande.

2. La méthode ObtenirEnfants()

Rev1 : comportement modifié, nouvelle description suit.
Elle doit renvoyer une liste des références des objets IParametrable qui dépendent hiérarchiquement de l'objet qui implémente la méthode. Cette méthode est prévue pour faciliter la gestion de la hiérérchie des modèles (cf. module modélisation), mais pourra également être mise à profit avec le module représentation pour gérer de multiples vues.
Rev1 : la méthode NombreEnfants s'avère inutile et est abandonnée.

3. La méthode ObtenirNom()

Elle fournit le nom identifiant l'objet. Il faudra s'assurer que ce nom est différent pour tous les enfants d'un même objet, car il sera utilisé comme clé dans la table de hachage associée à chaque liste.
Rev.1 : l'utilisation de cette chaine comme identifiant s'est avéré plus délicat que prévu, notamment à cause de ces 2 points : unicité non garantie, modifiable en cours de route. Cette chaine n'est donc conservée que pour identifier l'objet aux yeux de l'utilisateur, tandis que l'applet utilise deux systèmes pour identifier un objet selon le cas : sa référence, son emplacement dans la hiérarchie + sa classe.

4. La méthode ChangerNom(chaîne)

Elle permettra de changer la chaîne identifiant l'objet.

5. La méthode ObtenirParent()

Ajoutée dans cette révision 1, pour combler un manque : cette méthode permet de remonter dans la hiérarchie en fournissant la référence de l'objet dont dépend hiérarchiquement l'objet qui implémente la méthode.

6. La méthode ObtientPanel()

Rev.1 : Cette méthode doit renvoyer la référence d'un panel que CappletSimulation pourra insérer dans son CardLayout. Elle ne devra être utilisée que par cette objet.

7. La méthode Disparaît()

Rev.1 : Cette méthode permet de signifier à un objet Iparametrable que l'utilisateur a entrepris une action qui implique la destruction ou l'abandon de cet objet, qui doit donc s'y préparer : exemple notifier les autres objets de son invalidation imminente…

C. L'interface INotifiable (rev.1)

Elle doit être implémentée non seulement par les objets IParametrable, mais aussi par l'objet CAppletSimulation ; cependant ces 2 implémentations seront différentes. En effet CAppletSimulation collecte les notifications effectuées par tout objet INotifiable, puis les broadcaste à tous les autres objets Inotifiables(l'initiateur de la notification inclus).

1. NotifieCréation(ref objet)

Cette notification doit être appelée par un objet IParametrable au cours de son initialisation, afin de signaler sa création aux autres objets(ie surtout pour ceux qui ne lui sont pas directement liés . Ex : le générateur pourra créer une nouvelle instance s'il apprend qu'un nouveau modèle a été ajouté à la hiérarchie d'objet.

2. NotifieDestruction(ref objet)

De même que précédemment, mais pour la destruction de l'objet ;cette méthode est censée être appelée à l'intérieur de la méthode Disparaît() du IParametrable.
Même utilité, par ex. supprimer un objet d'une liste quand il n'est plus disponible.

3. NotifiePret()

Cette notification est particuliere, elle est censée n'être invoqué que par CAppletSimulation : elle sert à annoncer que l'utilisateur veut démarrer la simulation. Tous les objets doivent signifier leur accord (configuration ok) en renvoyant une référence nulle. Une erreur est identifiée par une chaine de caractère (objet String) dont la référence est passée en retour.

4. NotifieStart()

Elle suit immédiatement la précédente si il n'y a pas d'erreur : elle indique que la simulation a commencé. Elle est prévue pour signaler à tout objet d'interdire toute modification critique de la configuration par l'utilisateur. En effet, il faut avoir à l'esprit que l'interface est toujours disponible à l'utilisateur en cours de simulation.

5. NotifieStop()

Annule les effets de la précédente notification. Celle-ci n'est **pas** précédée d'un avertissement comme la précédente.

Cas particulier : pour le module representation, cette notification ne pourra être générée qu'entre deux cycles de simulation, c'est-à-dire qu'au lieu d'appeler NouveauCycle() pour recommencer un cycle, cette notification sera appelée à la place.

6. NotifieChangement()

Cette méthode est prévue pour faciliter la coopération entre modules : elle permet à un objet de signaler à tous les autres que son état (sa configuration) a changé. Ceci permet d'assurer que la configuration restera cohérente à travers l'applet, même si l'utilisateur rentre des données contradictoires dans différents modules.

7. NotifieNouveauNom()

Comme ci-dessus, ce type de changement de configuration a été délibérément séparé des autres, parce qu'il est commun à tous les IParametrable, mais n'a pas la même utilité que les autres : il permet uniquement d'assurer qu'un objet est identifié auprès de l'utilisateur par le même nom dans toute l'applet.

8. NotifieReorganisation()

Comme ci-dessus, cette notification signale qu'un objet n'a plus le même emplacement dans la hiérarchie ; étant donné que l'interface affiche cette information pour l'utilisateur, il faut petre en mesure de la mettre à jour.

D. L'interface Ilocalisable et la classe Traduit

L'interface Ilocalisable répertorie tous les messages, étiquettes, et autres chaines de caractères spécifiques à une langue que l'utilisateur peut voir.
Pour chaque chaine répertoriée, la classe Traduit en fournit la version dans la langue demandée. Ainsi l'implémentation d'une nouvelle langue ne nécessite que l'ajout de nouvelles chaines à la classe Traduit.

5. Détails ou problèmes éventuels à ne pas oublier

A. Multithread

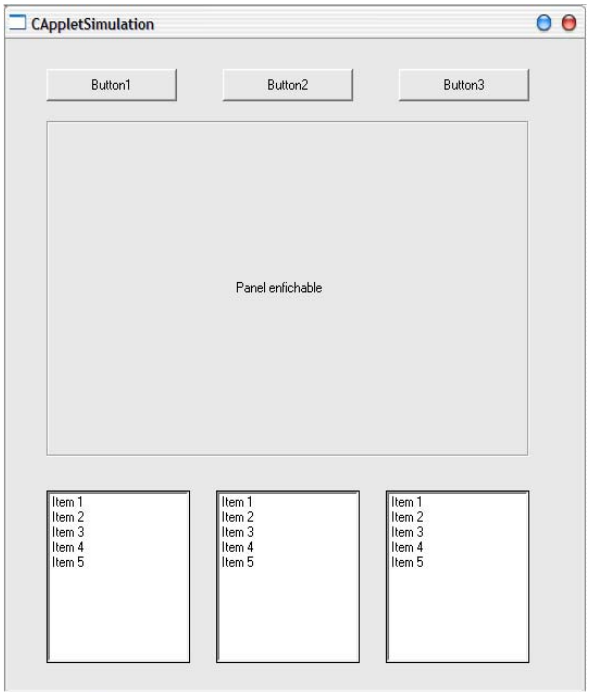
Il faudra réfléchir à l'opportunité d'utiliser les capacités multithread de Java. Voir l'utilisation du mot-clé synchronized.
Rev.1 : la méthode NouveauCycle de CmoduleRepresentation sera executée dans un thread à part afin de ne pas bloquer l'interface. Un « chien de garde » permettra de signaler à ce thread s'il doit initier des nouveaux cycles ou non.

B. Aspect graphique

*Rev.1 :*Il n'y a plus qu'une liste, à gauche, les boutons en haut, et le panneau enfichable au centre.
Je propose l'organisation suivante (cf. photo) :

- les boutons permettent de démarrer, arrêter la simulation, réinitialiser l'applet.

- Le panel enfichable correspond à l'item sélectionné dans la liste du milieu
- La liste de gauche contient l'objet parent à celui sélectionné dans la liste du milieu, ainsi que les objets « frères » de cet objet parent.
- La liste de droite contient les objets fils de l'item sélectionné



Lorsqu'un objet d'une des listes est sélectionné, on ferait les modifications suivantes : la liste qui contient l'item devient la liste du milieu, et les deux autres liste sont mises à jour en conséquence.
On pourrait en plus prévoir qu'un double clic sur un item provoque l'affichage du panel enfichable dans une fenêtre séparée (pratique pour les représentations). Les variations sont nombreuses, on peut aussi prévoir l'affichage d'un menu qui propose un des thèmes disponibles pour l'item sélectionné…
Une meilleure apparence a été proposée, qui ne remet pas en cause l'architecture interne du module, mais je n'ai pas eu le temps de mettre à jour la photo…

6. Découpage des tâches

Les axes de travail sont les suivants :

- ✓ Programmation de CAppletSimulation
- ✓ Evaluer l'opportunité d'utiliser le multithread pour séparer calculs et affichage
- ✓ Rédaction de la page Web dans laquelle sera incluse l'applet
- ☞ Définition des panels de chaque objet des autres modules (en collaboration avec eux)
- × Gestion des exceptions et erreurs (à définir)
- × Rédaction des aides contextuelles