T H E R M O P T I M ®

MANUEL

DE

RÉFÉRENCE

TOME III

UTILISATION ET CONCEPTION DE CLASSES EXTERNES

CORPS, TRANSFOS ET NŒUDS EXTERNES

PILOTES

VERSIONS JAVA 2.5 A 2.8

© R. GICQUEL OCTOBRE 2018

SOMMAIRE

© R. GICQUEL 1997 - 2012. Toute représentation ou reproduction intégrale ou partielle faite sans autorisation est illicite, et constitue une contrefaçon sanctionnée par le Code de la propriété intellectuelle.
Avertissement : les informations contenues dans ce document peuvent faire l'objet de modifications sans préavis, et n'ont en aucune manière un caractère contractuel.

1 AVERTISSEMENT

Documentation de Thermoptim

La documentation du progiciel THERMOPTIM se présente sous trois formes complémentaires :

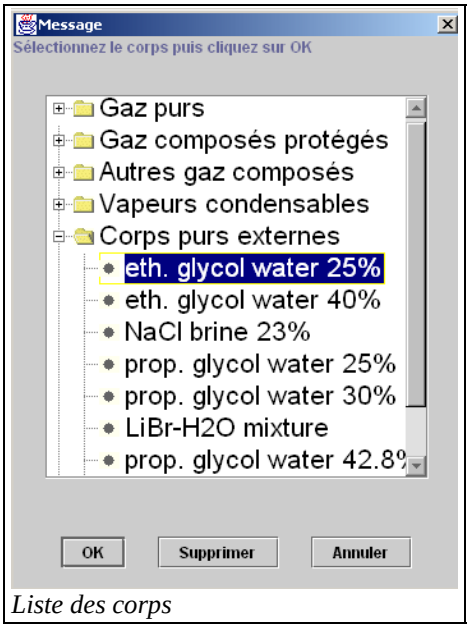
- une documentation succincte appelée "Référence rapide", accessible à partir du menu Aide du progiciel, sous forme d'une fenêtre à onglets présentant les principales notions utilisées
- une documentation imprimable, essentiellement sous format pdf
- de nombreuses ressources numériques mises en ligne dans le portail Thermoptim-UNIT (www.thermoptim.org), dont des modules de formation sonorisés

Des détails sur cette documentation sont donnés dans le premier tome du manuel de référence.

Le présent document en constitue le troisième tome. Il est dédié à la conception de classes externes. Après une présentation rapide du mécanisme d'extension qui a été adopté, il explique comment utiliser et programmer des classes externes, et introduit pour cela un environnement de développement librement distribué.

2 MECANISME D'EXTENSION DE THERMOPTIM PAR AJOUT DE CLASSES EXTERNES

L'environnement graphique de Thermoptim présente le grand intérêt de permettre de construire visuellement des modèles de nombreux systèmes énergétiques, des plus simples comme un réfrigérateur, aux plus complexes comme des centrales électriques à cycle combiné à gazéification intégrée mettant en jeu plusieurs centaines d'éléments.



Liste des corps

Non seulement une telle manière de faire simplifie notablement la démarche de modélisation et facilite ultérieurement l'utilisation et la maintenance du modèle, mais surtout elle sécurise sa construction en automatisant l'établissement des couplages entre les différents éléments qui le composent et en garantissant leur cohérence.

Jusqu'à la version 1.4, seuls les composants disponibles dans le noyau de Thermoptim pouvaient être ainsi assemblés, ce qui limitait les possibilités de l'outil, et plusieurs utilisateurs ont exprimé le souhait de pouvoir définir leurs propres éléments ou leurs propres corps.

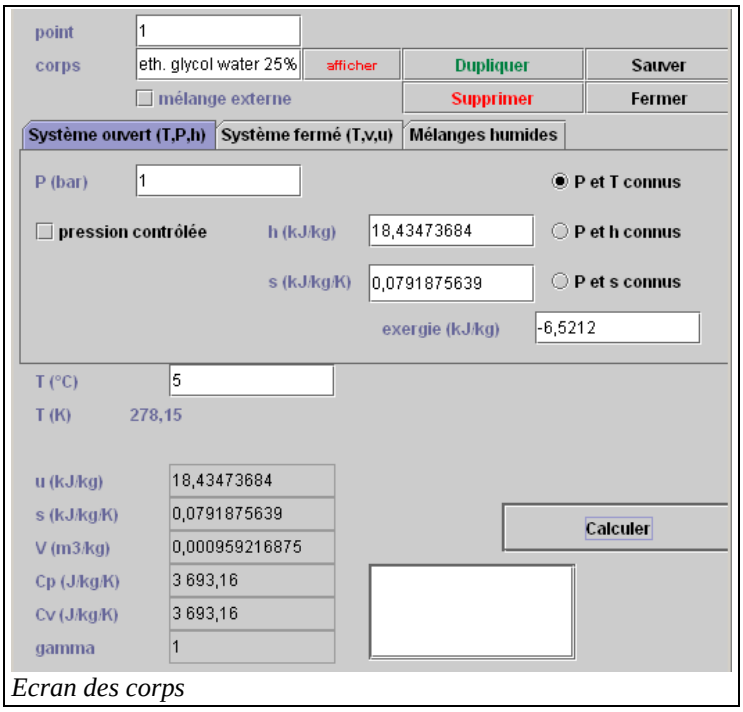
L'interfaçage de Thermoptim avec des **classes** (éléments de code Java) **externes** permet de répondre à cette demande et facilite l'interopérabilité du progiciel avec l'extérieur, notamment avec d'autres applications développées sous Java.

Son intérêt est double :

- pouvoir réaliser des extensions de Thermoptim à partir du noyau commun, en ajoutant des modules externes reconnus par le progiciel, qui définissent des éléments qui apparaissent automatiquement dans ses écrans de manière transparente pour l'utilisateur. C'est ainsi qu'un utilisateur peut ajouter ses propres corps ou des composants non disponibles dans le noyau ;

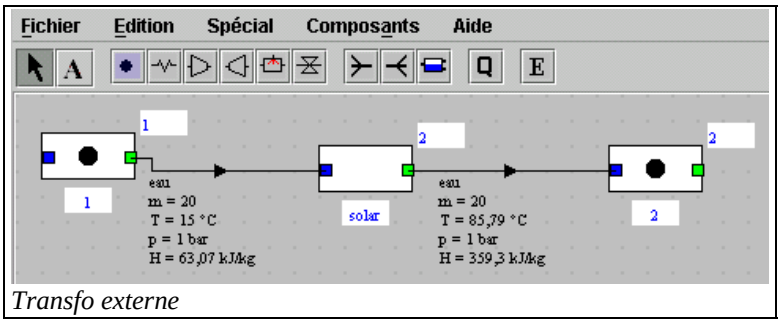
- pouvoir piloter Thermoptim à partir d'une autre application, soit pour guider un utilisateur (tuteur intelligent), soit pour contrôler l'exécution du code (pilotage ou régulation, accès aux bibliothèques thermodynamiques).

Les figures ci-contre montrent comment les corps externes s'ajoutent à la liste des corps de Thermoptim, puis se substituent à un corps interne dans l'écran de point, rendant leur usage aussi facile pour les utilisateurs que s'ils faisaient partie du progiciel.



Ecran des corps

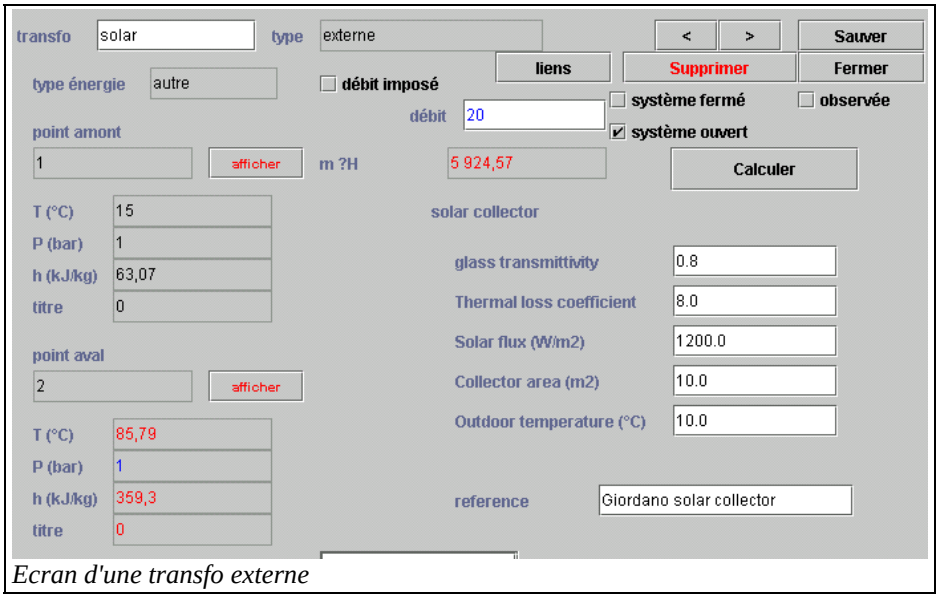
La figure ci-contre montre comment un composant externe représentant un capteur solaire apparaît dans l'éditeur de schémas, et la figure ci-dessous présente l'écran de la transfo correspondante, composée pour partie par du code interne de Thermoptim, et pour partie par du code externe (tiers inférieur droit avec libellés en anglais).



Transfo externe

2.1 Implémentation logicielle

Sur le plan pratique, ajouter une nouvelle transfo externe est très simple : il suffit de créer une classe spécifique, qui hérite de la classe mère abstraite extThopt.ExtProcess, l'interaction avec Thermoptim étant assurée à deux niveaux :



Ecran d'une transfo externe

- par des méthodes générales permettant d'effectuer les calculs requis ;
- par un JPanel qui est incorporé dans l'écran de la transfo externe. De cette manière, le concepteur de la classe peut créer sa propre interface graphique, qui s'insère ensuite dans l'écran dans Thermoptim

2.2 Contraintes informatiques

Pour protéger le code de Thermoptim, ses classes sont "camouflées", c'est-à-dire que l'ensemble de ses éléments est renommé, rendant extrêmement difficile sa décompilation. En revanche, cette opération interdit en pratique de pouvoir accéder de l'extérieur au contenu de Thermoptim. La solution est alors de ne pas camoufler les méthodes que l'on souhaite rendre accessibles. Il y a donc un compromis à trouver entre l'accessibilité et la protection du code, cette dernière conduisant à prévoir un nombre d'accès aussi limité que possible. Pour le moment, seuls deux packages de Thermoptim sur cinq sont ainsi partiellement accessibles : rg.thopt et rg.corps. Le premier contient les classes du simulateur, le second celles des corps.

La solution retenue consiste à ajouter un package non camouflé (extThopt), dans lequel existent des classes (par exemple extThopt.CorpsExterne), reconnues par Thermoptim, qui sont des sous-classes de classes de Thermoptim, par l'intermédiaire de classes d'interface (extThopt.CorpsExterne sous-classe rg.corps.CorpsExt, qui elle-même sous-classe rg.corps.Corps). Ces classes possèdent des méthodes qui établissent la correspondance avec leur équivalent interne camouflé. Nous vous recommandons de vous référer à l'API du package extThopt pour connaître l'ensemble des méthodes disponibles, leur syntaxe et leur fonction. Cette API fait partie de l'environnement de développement des classes externes mis à votre disposition (dans le dossier api_extThopt).

2.3 Mise en œuvre pratique

Dans la version 1.5, quatre types de classes externes ont été implémentés : les corps, les transfos, les diviseurs et les nœuds externes. Dans tous les cas, le mécanisme implémenté est analogue : dans le package externe (extThopt), une classe d'entrée est une simple extension de l'une de celles de Thermoptim (CorpsExterne pour les corps, TransfExterne pour les transfos…). C'est dans cette classe que se trouvent toutes les méthodes de Thermoptim surchargées en externe. Cette classe d'extThopt sert simplement de point d'entrée dans son package, son rôle étant d'instancier les diverses classes représentant les éléments ajoutés à Thermoptim et d'assurer l'interface avec le progiciel grâce aux méthodes surchargées.

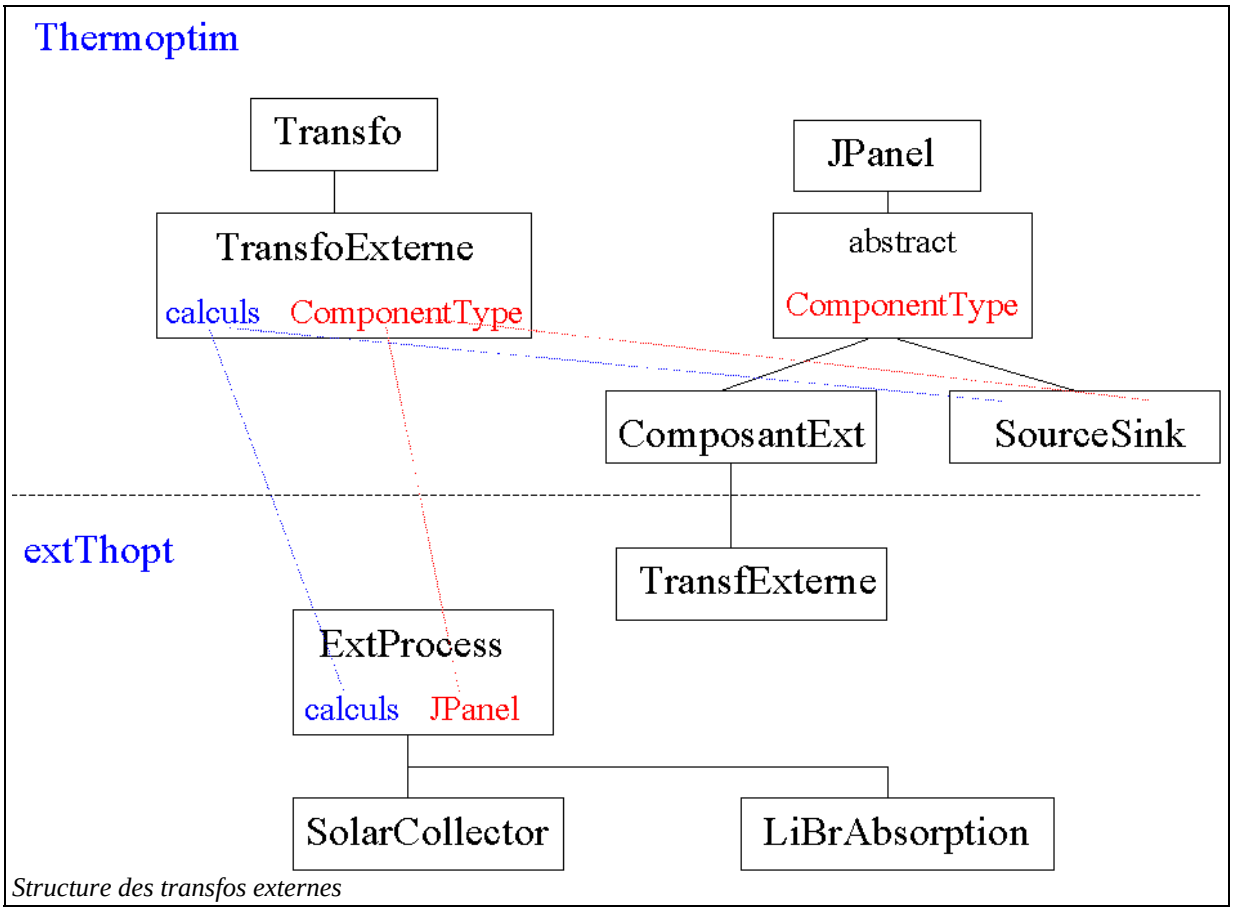
Les classes externes héritent d'une classe mère abstraite : par exemple ExtSubstance pour les corps, ExtProcess pour les composants. En pratique, l'interaction avec Thermoptim est assurée à un ou deux niveaux selon les cas :

- les classes-mères définissent toutes les méthodes générales utiles pour effectuer les calculs requis
- à l'exception de ExtSubstance, elles définissent de plus un JPanel qui est incorporé dans l'écran du composant externe. De cette manière, le concepteur peut créer sa propre interface graphique, qui apparaît ensuite à l'écran dans Thermoptim. Il peut ainsi définir ses propres éléments graphiques (boutons, champs, étiquettes...), et traiter dans ses méthodes des actions spécifiques. Pour les composants, deux méthodes, saveCompParameters() et readCompParameters(String ligne_data), permettent de sauvegarder les paramétrages dans le fichier de projet de Thermoptim, afin de pouvoir les relire lors du chargement du projet.

Pour résumer simplement les choses, des classes externes, comme CorpsExterne ou TransfExterne, héritent directement de celles de Thermoptim, et assurent l'interface avec le progiciel. Elles font appel à des classes abstraites, comme ExtSubstance ou ExtProcess, dont les sous-classes peuvent être définies librement par les utilisateurs, par l'intermédiaire d'une part des méthodes d'interface et d'autre part d'un JPanel.

Le schéma ci-dessous illustre la structure mise en place pour les transfos externes. Pour les corps, le schéma est analogue, mais plus simple : étant donné l'absence d'interface graphique, la partie droite du schéma n'a plus lieu d'être. Le lecteur familier avec la notation UML trouvera en annexe 6 les diagrammes des classes pour les corps et pour les transfos externes.

La partie haute du schéma correspond aux classes présentes dans le noyau de Thermoptim, la partie basse à celles du package extThopt modifiable par le concepteur de classes externes. SourceSink est une implémentation par défaut de ComponentType pour pouvoir instancier une TransfoExterne, même en l'absence de classes externes de ce type, et SolarCollector et LiBrAbsorption des exemples de classes externes. Les liens bleus et rouge en pointillés symbolisent les relais des parties de code relatives aux calculs et à l'interface graphique, effectués depuis TransfoExterne vers les autres classes, soit interne (SourceSink), soit externes (sous-classes de ExtProcess).



Structure des transfos externes

La sélection parmi les classes externes est directement assurée par Thermoptim, de la manière suivante : lors de son lancement, le progiciel analyse les archives extThopt2.zip et extUser2.zip, qui contiennent, entre autres, toutes les classes externes. Il retrouve ainsi toutes les classes ajoutées par défaut et par les utilisateurs, qu'il trie d'après leur classe-mère, et qu'il charge dans différents types de tableaux, ajoutant les éléments externes à ses propres listes pour qu'ils soient accessibles de manière transparente. Ultérieurement, si l'un de ces éléments est sélectionné, Thermoptim transmet sa classe au constructeur approprié, qui l'instancie. La ligne "Visualisateur de classes externes " du menu "Spécial" permet de visualiser les classes externes disponibles (cf. section 3.6).

Une fois un composant externe instancié, il est possible d'en changer en double-cliquant sur son type tel qu'il apparaît à l'écran.

Il n'y a pas pour le moment de test de cohérence sur les noms des classes et les types d'éléments, mais cela sera fait ultérieurement. Pour la mise au point des classes externes, un mécanisme d'émulation permet de lancer Thermoptim à partir d'une application externe, et de charger dynamiquement les classes en cours de développement, en ayant ainsi accès aux outils de débogage de l'environnement habituel de travail (cf. section 6).

Par ailleurs le package extThopt comprend une classe Util qui propose un certain nombre de méthodes utilitaires pour formater les nombres, les relire à l'écran, sauvegarder et relire des valeurs, inverser par dichotomie une méthode (cf. annexes 3)...

2.4 Trois catégories de méthodes d'interface

Les classes mises en jeu dans Thermoptim, CorpsExterne, TransfExterne et autres, définissent les méthodes d'interface, qui relèvent de trois catégories complémentaires :

- les méthodes internes de Thermoptim destinées à être camouflées
- les méthodes internes non camouflées directement utilisées par les classes externes : elles permettent d'exécuter les méthodes de Thermoptim avec des paramétrages externes, et **ne doivent pas être surchargées.** Leur signature doit être scrupuleusement respectée : en particulier, la structure des Vector ou

des Object passés en argument doit parfaitement correspondre à celle attendue par Thermoptim sous peine de générer des ClassCastException.
- les méthodes externes correspondant aux méthodes camouflées de Thermoptim : elles permettent d'exécuter les méthodes externes à partir de Thermoptim, et **doivent être surchargées**

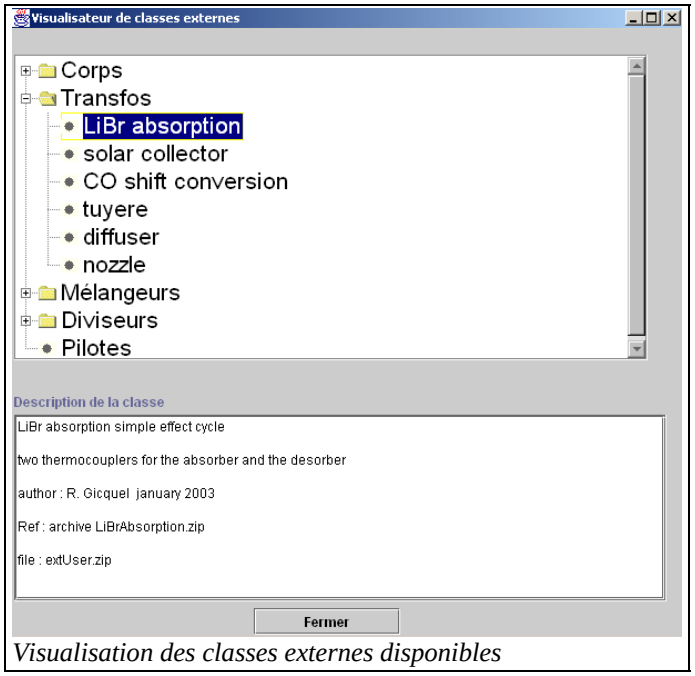
La liste des méthodes non camouflées relevant des deux dernières catégories est donnée en annexe 1, pour l'ensemble des classes accessibles. Pour plus de précisions, le mieux est de se reporter à la documentation html de ces classes donnée dans le dossier api_Thermoptim.

2.5 Utilisation des composants externes

2.5.1 Visualisation des classes externes disponibles

Pour vous aider dans l'utilisation et la gestion des classes externes, la ligne "Visualisateur de classes externes" du menu "Spécial" du simulateur permet de visualiser l'ensemble des classes externes disponibles. Leur liste est affichée, triée par type (corps, transfos, mélangeurs, diviseurs, pilotes) avec un court descriptif de la classe sélectionnée et indication de sa provenance (archives extThopt2.zip et extUser2.zip ainsi que les classes en cours de développement).

Cet écran peut être consulté pendant que vous mettez au point votre modèle.



Visualisation des classes externes disponibles

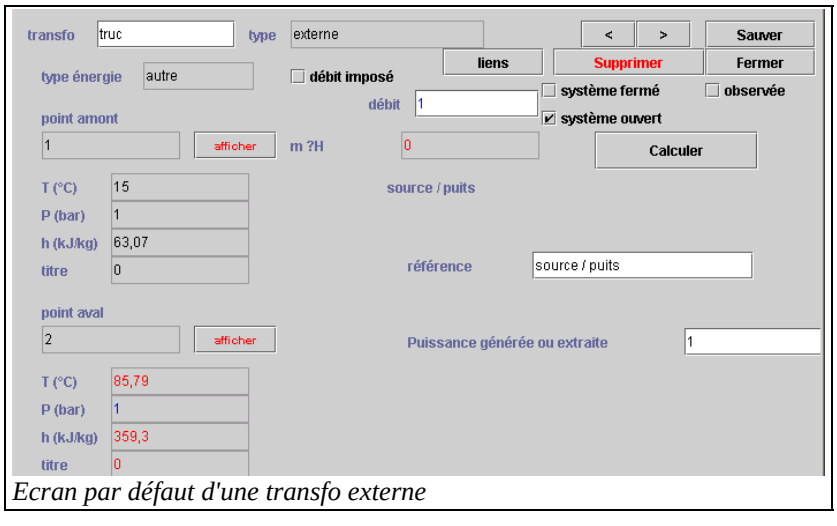
2.5.2 Représentation d'un composant externe dans l'éditeur de schémas

Des icônes spécifiques ont été introduites pour représenter les composants externes (pour les transfos, pour les mélangeurs, et pour les diviseurs). Le composant externe est ensuite sélectionné lors de la mise à jour du simulateur à partir du schéma comme indiqué ci-après.

2.5.3 Chargement d'une classe externe

Pour charger une transfo externe (pour un nœud externe, la procédure est analogue), vous pouvez :
- soit, à partir de l'écran du simulateur, double-cliquer dans le bandeau du tableau des transfos, puis choisir "externe", et enfin sélectionner le type de transfo externe que vous désirez parmi la liste qui vous est proposée ;
- soit, à partir de l'éditeur de schémas, construire le composant externe sous forme graphique, puis mettre à jour le simulateur à partir du schéma. Dans le cas d'une transfo externe, il s'agit par défaut d'un type "source / puits", comme le montre l'écran ci-contre.

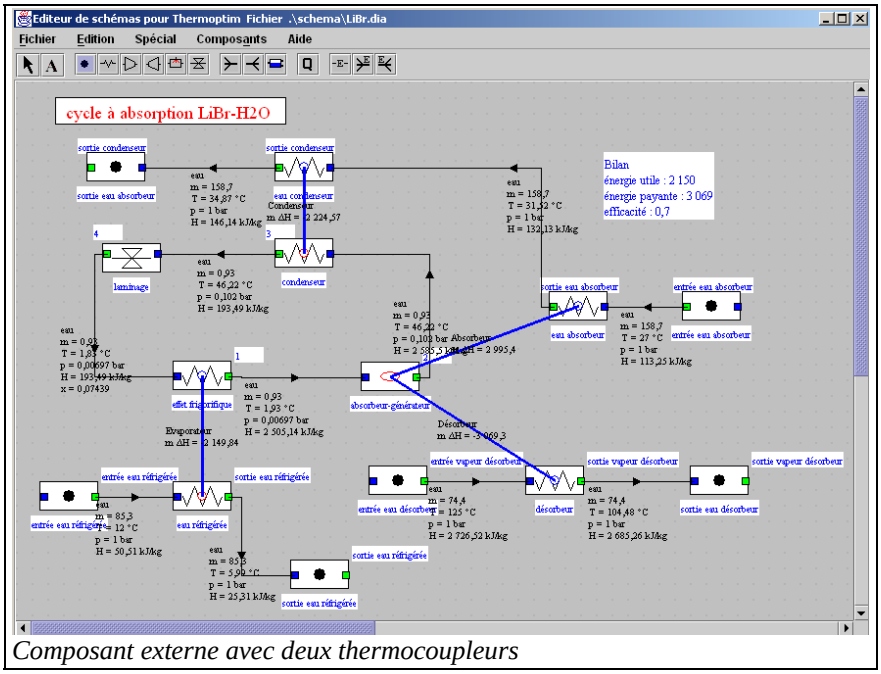
Une fois cette transfo par défaut créée, double-cliquez sur l'intitulé "source / puits", ce qui vous donne accès à la liste de toutes les transfos externes



Ecran par défaut d'une transfo externe

disponibles. Choisissez celle que vous désirez, qui est alors chargée.

2.5.4 Thermocoupleurs

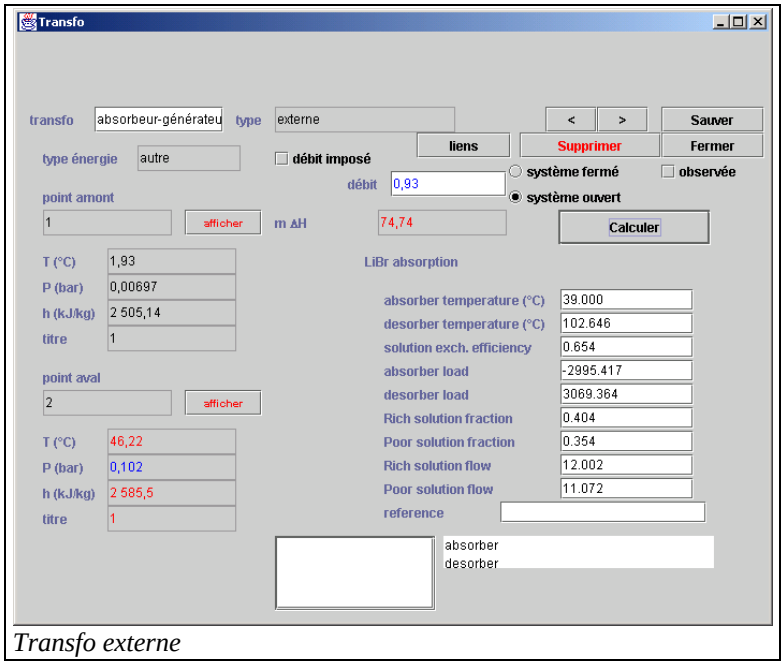


Composant externe avec deux thermocoupleurs

Le mécanisme des thermocoupleurs complète celui des échangeurs de chaleur en permettant à des composants autres que des transfos "échange" de se connecter à une ou plusieurs transfos "échange" pour représenter des couplages thermiques. Il n'englobe pas celui des échangeurs : deux transfos "échange" ne peuvent pas être connectées par un thermocoupleur.

Un tel mécanisme présente un grand intérêt, car il permet de représenter de nombreux couplages thermiques qui ne constituent pas pour autant un échange de chaleur classique, comme par exemple le refroidissement des parois de la chambre de combustion d'un moteur alternatif, une compression refroidie, et surtout des apports ou des extractions de chaleur au niveau de composants externes multifonctionnels.

La figure ci-dessus en est une illustration : un cycle de réfrigération à absorption, dont le système d'absorption-désorption est défini de manière intégrée dans une transfo externe, est traversé par de la vapeur d'eau qui sort de l'évaporateur puis entre dans le condenseur. Ce cycle met en jeu le couple LiBr-H_2O, dont les propriétés sont modélisées soit directement dans la transfo externe soit dans un corps externe, et requiert d'une part un apport de chaleur à haute température au niveau du désorbeur, et d'autre part une extraction de chaleur à moyenne température au niveau de l'absorbeur. La représentation de ces couplages thermiques est ici possible grâce au mécanisme des thermocoupleurs : la transfo externe calcule les énergies thermiques qui doivent être échangées, et le thermocoupleur recalcule la transfo "échange" correspondante qui met à jour son point aval.



Transfo externe

Les types de thermocoupleurs auxquels fait appel un composant externe apparaissent en bas à droite de leur écran. Un double-clic sur l'un des types ouvre l'écran du thermocoupleur correspondant.

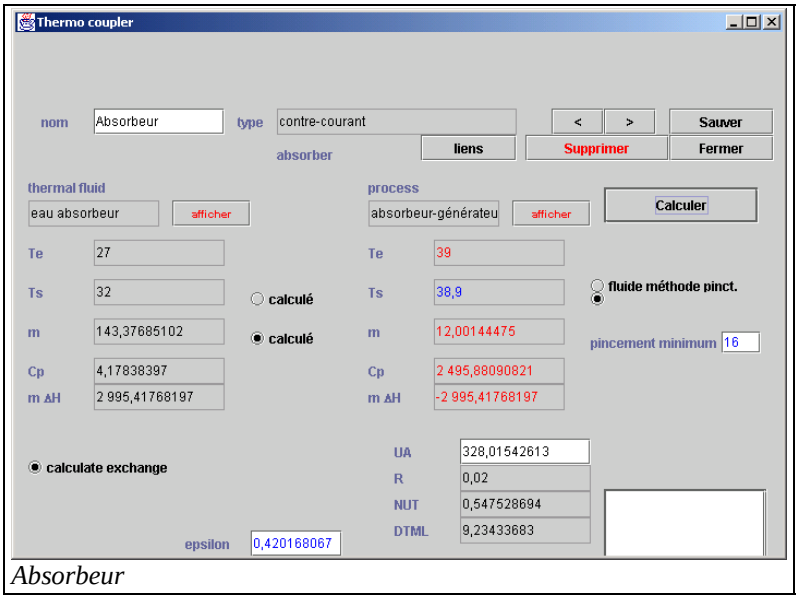
Etant donné que les thermocoupleurs sont des sortes d'échangeurs de chaleur, il est intéressant de les caractériser par des valeurs telles que l'efficacité ε, le UA, le NUT ou la DTML, que l'on peut calculer à partir d'équations analogues. Pour cela, le composant transmet à chacun de ses thermocoupleurs les valeurs d'équivalents débits, température d'entrée et de sortie et énergie thermique transférée qu'ils doivent prendre en compte dans leurs calculs. Des méthodes spécifiques sont prévues dans le code de la classe externe, et ne sont pas modifiables par l'utilisateur.

Il faut cependant être conscient que l'analogie avec les échangeurs comporte quelques limites : par exemple, des croisements de température inadmissibles dans un échangeur peuvent se présenter dans un thermocoupleur, conduisant à une absurdité au niveau des calculs si l'on n'y prend pas garde.

On a donc souvent intérêt à transmettre des valeurs qui ne risquent pas de conduire à ce genre de situation, une solution pouvant être de considérer, pour les calculs des caractéristiques analogues à celles d'un échangeur, que le thermocoupleur est isotherme. Par exemple, une chambre de combustion pourra être supposée être à la température moyenne entre ses points amont et aval lorsqu'on calcule son refroidissement. Une telle hypothèse n'a rien d'absurde et peut éviter un croisement de températures entre le fluide de refroidissement et les gaz qui traversent le composant.

Dans le cas de la machine à absorption présentée plus haut, on a supposé isothermes l'absorbeur et le désorbeur.

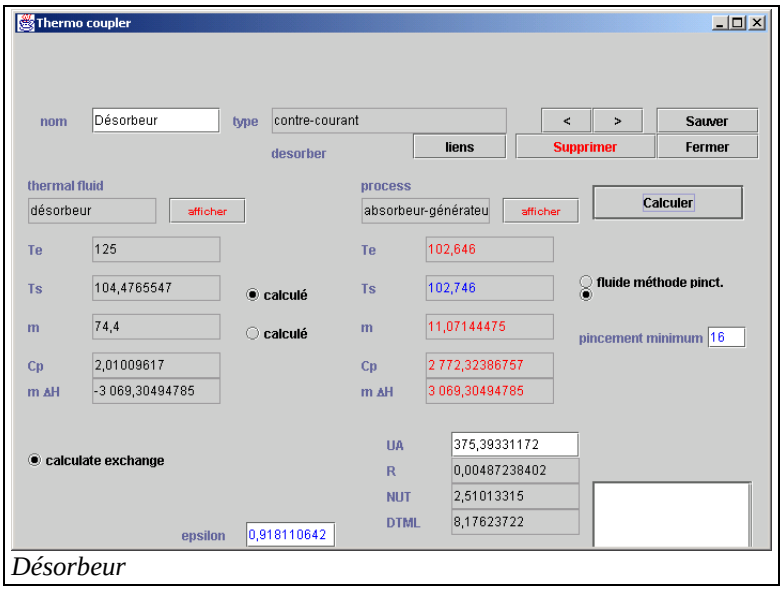
Tous deux conduisent aux écrans ci-dessous. Si l'on n'avait pas pris la température de l'absorbeur comme référence pour les calculs d'échange, en conservant celles de la vapeur d'eau entrant et sortant de la transfo externe, on aurait abouti à un croisement de température.



Absorbeur

Pour les transfos externes admettant plusieurs thermocoupleurs et pour les nœuds externes, la complexité potentielle des calculs à réaliser vient limiter la possibilité de permettre à la transfo "échange" de jouer un rôle autre que passif où elle subit la charge imposée par le composant externe. C'est pour cette raison que le calcul d'un thermocoupleur possède beaucoup moins d'options que celui d'un échangeur de chaleur : l'utilisateur ne peut que choisir entre le calcul de la température de sortie de la transfo échange (à débit donné) et celui du débit, la température étant connue.

On remarquera que c'est sur l'écran du thermocoupleur que le fluide du composant externe peut être sélectionné comme fluide de pincement et qu'une valeur de pincement minimum peut être entrée (cf. méthode d'optimisation, tome 1).



Désorbeur

2.5.5 Nœuds externes

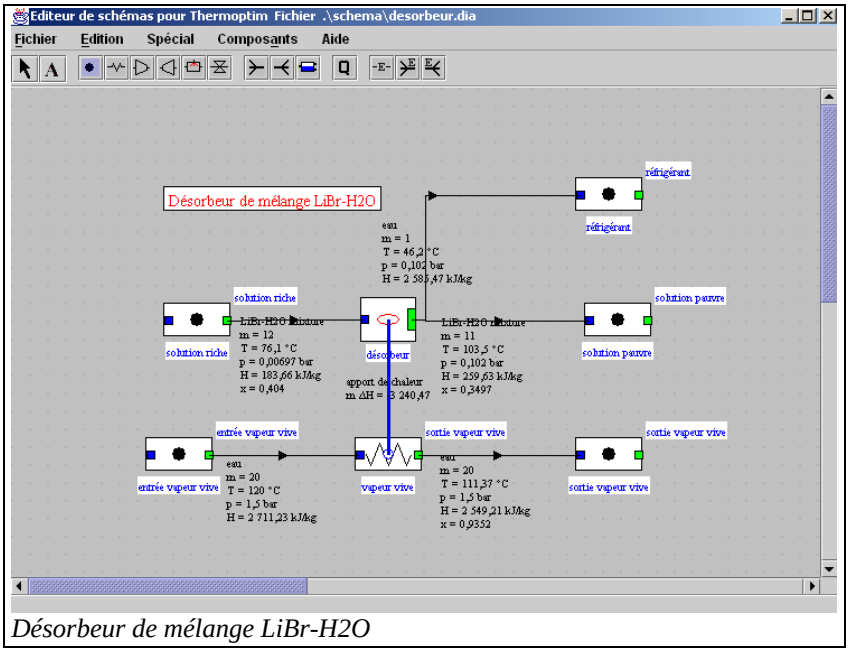
Les nœuds du noyau de Thermoptim sont des composants extrêmement simples utilisés pour compléter la description des circuits fluides. Ils sont toujours adiabatiques, et assurent la conservation du débit-masse et celle de l'enthalpie.

Il existe toutefois un composant un peu particulier, considéré comme une transfo, mais qui est en fait un nœud dans la plupart des cas : la combustion, qui reçoit un comburant et un combustible, et dont sortent les gaz brûlés.

A l'instar de cet exemple, interviennent dans un certain nombre de systèmes énergétiques, des composants de complexité variable, qui peuvent recevoir plusieurs fluides en entrée, et dont peuvent en sortir plusieurs, après des traitements internes variés, avec ou non couplage thermique avec des sources de chaleur externes.

Les nœuds externes ont pour objet de permettre à un utilisateur de définir de tels composants. Seuls des mélangeurs et des diviseurs externes sont définis : aucun composant n'intègre simultanément les deux fonctions (recevoir plusieurs fluides et en émettre plusieurs), mais il suffit de coupler un mélangeur externe à un diviseur externe pour le faire.

A de nombreux égards, les problèmes d'implémentation de ces nœuds externes sont proches de ceux des thermocoupleurs : la complexité potentielle des calculs à réaliser dans le nœud oblige celui-ci à prendre la main et à asservir tant la veine principale que les branches, aucun calcul par défaut n'étant envisageable.



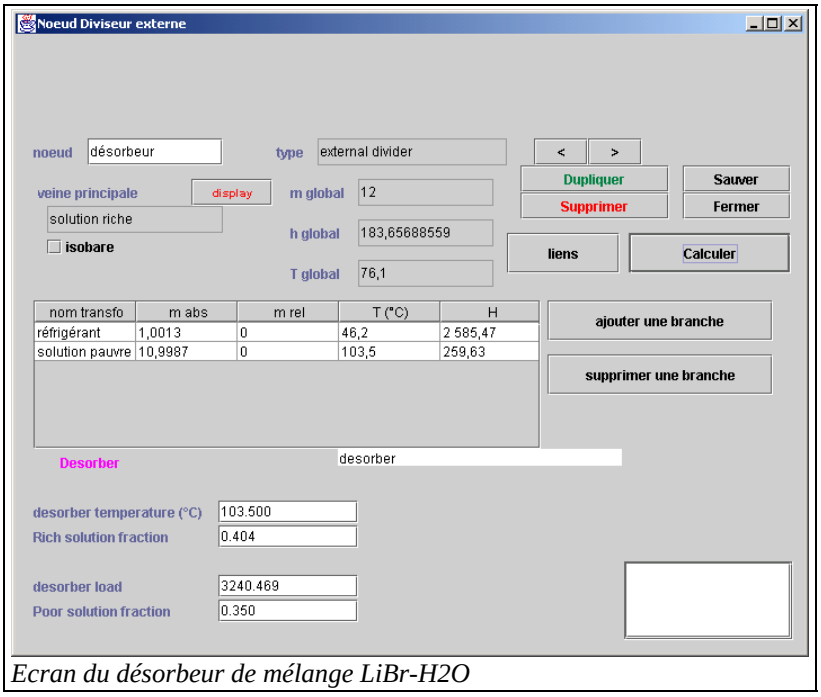
Désorbeur de mélange LiBr-H2O

Les problèmes de vérification de cohérence sont encore plus critiques que ceux des thermocoupleurs : seul le concepteur un tel nœud sait à quelles transfos il doit être connecté, tant en entrée qu'en sortie. Un utilisateur doit donc impérativement se référer à la documentation de la classe pour savoir comment le mettre en œuvre.

La figure ci-dessus montre le schéma d'un désorbeur pour une machine à absorption mettant en jeu le couple LiBr-H_2O, dont les propriétés sont cette fois-ci impérativement fournies par un corps externe.

L'écran du nœud externe est donné ci-contre. Tout comme les transfos externes, il comporte une partie générique définie dans le noyau de Thermoptim, complétée par une partie définie par l'utilisateur (ici dans la zone inférieure gauche). Dans le modèle retenu ici, le seul paramètre défini dans le nœud est la température du désorbeur, les propriétés de la solution riche (fraction massique et débit) et l'état du réfrigérant étant obtenus de leurs transfos respectives.

Le nœud calcule les débits du réfrigérant et de la solution pauvre, l'état de cette dernière, et l'apport calorifique nécessaire.

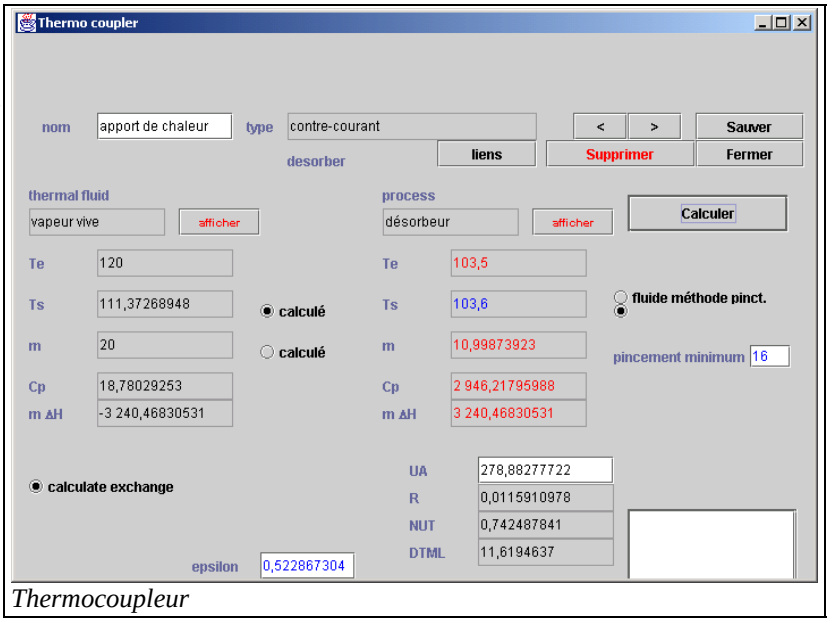


Ecran du désorbeur de mélange LiBr-H2O

Le thermocoupleur calcule ensuite l'état final de la vapeur vive utilisée.

Avant chaque recalcul, le nœud vérifie que sa structure est correcte et charge les valeurs dont il a besoin pour être calculé. Ensuite, il met à jour le thermocoupleur qui peut être recalculé à son tour. On a ici fait l'hypothèse que le désorbeur était isotherme, et on a pris le débit de solution pauvre comme débit de référence.

2.5.6 Remarques diverses



Thermocoupleur

Vous aurez remarqué que les écrans des composants externes représentés ici sont composés pour partie par du code interne de Thermoptim, et pour partie par du code externe (tiers inférieur droit avec libellés en anglais dans les transfos, partie inférieure gauche dans les nœuds). Cela vient de ce que les chaînes de caractères utilisées dans les composants externes n'ont pas été "internationalisées" comme c'est le cas pour le noyau de Thermoptim.

Mais cela ne gêne en rien l'utilisation. En revanche, vous noterez que l'affichage des nombres n'est pas non plus internationalisé, de telle sorte que le séparateur décimal n'est pas le même : il s'agit du point et non pas de la virgule. Il serait bien sûr possible de le modifier, mais ceci n'a pas été fait pour le moment. Il en résulte qu'il faut entrer les chiffres décimaux avec un point dans la partie définie dans le composant externe, et avec une virgule dans le reste de l'écran. Il importe d'y faire attention, faute de quoi des erreurs de formatage des nombres seront détectées par Java.

2.6 Représentation des mélanges de fluides réels dans Thermoptim

Jusqu'en 2005, les seuls corps de Thermoptim dont la composition pouvait être définie par l'utilisateur étaient les gaz composés : il était impossible de représenter des mélanges de fluides réels. Le mécanisme des corps externes a été aménagé pour pouvoir le faire, grâce à l'introduction d'un nouveau type de corps, appelé "**mélange externe**", les anciens corps externes étant rebaptisés "**corps purs externes**".

Un mélange externe est réalisé à partir d'un **système**, c'est-à-dire d'un jeu donné de **corps purs**, et sa composition est spécifiée dans un nouvel éditeur.

Il importe de bien noter que la caractéristique distinctive des mélanges externes est de permettre de générer un ensemble de corps à partir d'un même **système** de corps purs. Ils sont en ce sens analogues aux gaz composés, la différence étant que les interactions entre les fluides réels sont beaucoup plus complexes qu'entre des gaz idéaux, de telle sorte qu'il faut spécifier non seulement les corps purs mis en jeu, mais aussi leurs modèles, les règles de mélange et un ensemble de paramètres additionnels.

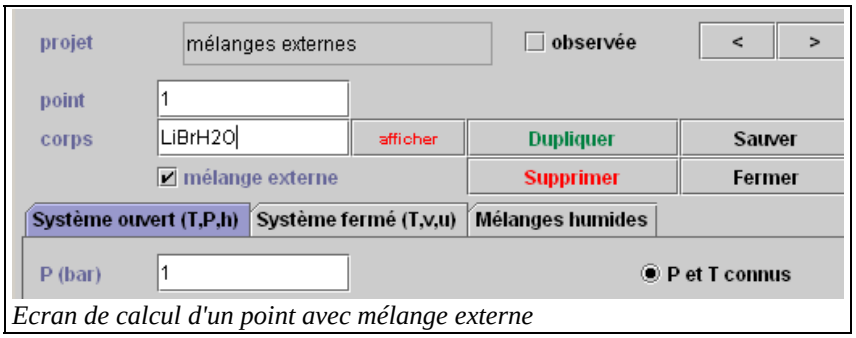
2.6.1 Couplage entre Thermoptim et des serveurs de propriétés thermodynamiques

Le mécanisme qui a été implémenté permet non seulement de définir des mélanges externes en écrivant complètement leurs classes externes, mais aussi de les calculer dans ce que l'on appelle quelquefois des **serveurs de propriétés thermodynamiques (SPT)**, progiciels possédant des bibliothèques spécialisées, comme TEP ThermoSoft développé par le Centre Energétique et Procédés de l'Ecole des Mines de Paris, ou bien encore Simulis Thermodynamics de la société ProSim.

Un tel couplage constitue une très intéressante extension de Thermoptim, puisqu'il permet de mettre en œuvre dans le progiciel de nombreux fluides nouveaux, et ceci avec des modèles d'une très grande précision. La contrepartie est une certaine complexité et surtout d'un temps calcul quelquefois significatif.

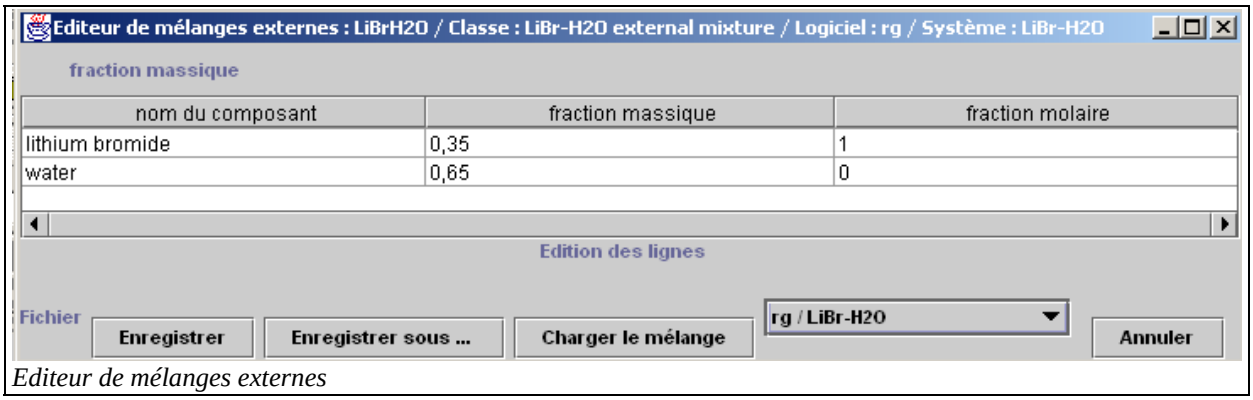
2.6.2 Création d'un mélange externe

L'écran de calcul d'un point a été légèrement modifié : une option "mélange externe" a été rajoutée. Pour créer un nouveau mélange externe, il suffit de sélectionner cette option puis d'entrer un nom de corps qui n'existe pas déjà, et de taper sur Entrée.



Ecran de calcul d'un point avec mélange externe

L'éditeur de mélanges externes peut alors être ouvert en cliquant sur "afficher". Par défaut, le mélange externe créé est du type "LiBr-H2O external mixture", dont la classe est présentée à titre d'exemple dans la documentation de programmation. Si c'est bien le corps désiré, entrez sa composition, puis cliquez sur "Enregistrer". Le type de composition attendu (fraction molaire ou massique) est affiché au dessus de la colonne des noms des composants, et correspond à la première colonne de chiffres à partir de la gauche.



Editeur de mélanges externes

Dans le titre de l'éditeur de mélanges externe apparaissent successivement :

- le nom du mélange (ici LiBrH2O)
- l'identifiant de la classe externe correspondante (ici LiBr-H2O external mixture)
- le nom du logiciel utilisé (soit un SPT, soit comme ici (rg) une famille de classes externes)
- le système utilisé (ici LiBr-H2O)

Ces différents renseignements permettent de caractériser le mélange utilisé, en se référant à la documentation existante. En effet, un même SPT peut proposer plusieurs systèmes, et être éventuellement implémenté dans plusieurs classes externes.

Le menu déroulant situé en bas à droite de l'écran de l'éditeur permet d'afficher l'ensemble des systèmes disponibles. Si celui qui est créé par défaut n'est pas celui que vous désirez, sélectionnez en un dans la liste, puis cliquez sur "charger le mélange". L'éditeur est alors mis à jour, et il ne reste plus qu'à entrer la composition puis à cliquer sur "Enregistrer".

Dans un fichier de projet Thermoptim (.prj), les données relatives aux mélanges externes sont sauvées de manière analogue à celles des gaz composés.

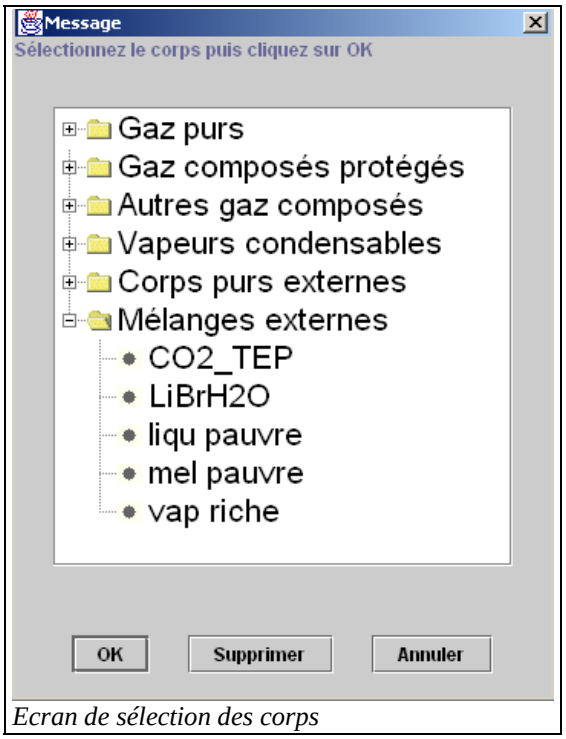
```
CORPS COMPOSES 2
Nom du gaz / Composants       fraction molaire      fraction massique

gaz_brulés     5
CO2     0.0309766336   0.0477375785
H2O     0.0619532671   0.0390824699
O2      0.14154164     0.158596897
N2      0.756807249    0.74238276
Ar      0.0087212103   0.0122002945

LiBrH2O 2       extMixture=true       classe=LiBr-H2O external mixture     syst=LiBr-H2O
lithium bromide       0.35    0
```

```
water   0.65    0
```

extMixture=true permet de savoir qu'il s'agit d'un mélange externe, et le reste de la ligne spécifie la classe et le système choisi. Les lignes suivantes donnent les compositions molaires ou massiques (pour le moment, pour un mélange donné, l'entrée ne peut être faite que de la manière choisie par le concepteur de la classe externe, soit en fractions molaires, soit en fractions massiques).

On prendra garde à ce que l'ordre dans lequel apparaissent les composants n'est pas indifférent : c'est celui qui est défini dans la classe externe correspondante. Si donc on est amené à éditer à la main un fichier de projet, il faut impérativement respecter l'ordre d'apparition des composants du système.

L'écran de sélection des corps a été modifié pour distinguer les deux catégories de corps externes. Dans cet exemple, les trois mélanges externes du bas de la liste correspondent à des compositions différentes du même système.



Ecran de sélection des corps

Une fois un mélange externe défini ou chargé à partir de l'écran de sélection des corps, sa composition peut être modifiée dans l'éditeur, qui permet aussi comme nous l'avons vu de changer de système en en sélectionnant un parmi les bibliothèques disponibles (menu déroulant en bas à droite).

3 PROGRAMMATION DES COMPOSANTS EXTERNES

Dans cette section sont présentées les bases permettant de programmer des composants externes, avec quelques extraits issus des exemples fournis avec l'environnement de développement. Soulignons toutefois avant tout combien il est important de bien préparer la documentation de vos classes. Cette documentation doit comprendre au moins deux volets : le dossier de programmation et la documentation d'utilisation. Une classe mal documentée sera difficile à utiliser ainsi qu'à modifier, et c'est au moment même où l'on crée la classe externe qu'il est le plus facile de la documenter. Ensuite, l'expérience prouve qu'on ne le fait que très rarement…

La classe externe représentant un modèle de composant ayant un sens physique, il est impératif de rédiger une note décrivant ce modèle, tant pour le dossier de programmation que pour la documentation d'utilisation. Le code lui-même devra comprendre suffisamment de commentaires pour que les liens avec le modèle soient aussi clairs que possible (en particulier, il est préférable que les notations soient homogènes). Enfin, la documentation d'utilisation devra indiquer très précisément les contraintes du composant, en spécifiant notamment avec soin d'une part quels sont les fluides qui peuvent lui être connectés, en entrée comme en sortie (nombre, nature…), et d'autre part comment doivent être construits les thermocoupleurs éventuels. La méthode getClassDescription () permet de saisir un bref descriptif de la classe qui est consultable depuis un écran de Thermoptim prévu à cet effet. Il est recommandé que les références de la documentation et les contraintes principales de la classe soient mentionnées dans cette méthode.

Sur le plan pratique, il peut être utile d'opérer comme suit : rassembler dans une archive portant le même nom que la classe les éléments suivants : le code java du composant, son fichier .class, une note de modélisation, une note d'utilisation avec un petit exemple de mise en œuvre, et les fichiers .prj et .dia de cet exemple. Une modélothèque contenant l'ensemble de ces archives peut facilement être constituée et mise à jour.

3.1 Corps externes

3.1.1 Construction des corps purs externes

La structure des classes externes a été définie plus haut : la classe extThopt.CorpsExterne hérite de rg.corps.CorpsExt de Thermoptim et fait l'interface, relayant les calculs au niveau d'une classe abstraite (extThopt.ExtSubstance) définissant les méthodes de base, et sous-classée par les différentes classes introduites (par exemple extThopt.GlycolWater25, extThopt.GlycolWater40…). Nous vous recommandons de vous référer à l'API de extThopt.CorpsExterne et extThopt.ExtSubstance pour connaître l'ensemble des méthodes disponibles, leur syntaxe et leur fonction. Cette API fait partie de l'environnement de développement des classes externes mis à votre disposition.

Comme Thermoptim ne connaît que la classe d'interface externe extThopt.CorpsExterne et la classe abstraite extThopt.ExtSubstance, les instanciations ne peuvent faire appel qu'à elles deux. La nécessaire synchronisation entre les méthodes internes et externes doit être faite avec soin, mais c'est la seule manière de s'assurer que les classes déployées fonctionnent bien.

Nous commencerons par expliquer le mécanisme de construction qui est implémenté dans Thermoptim, puis nous montrerons comment créer en pratique un nouveau corps externe, en sous-classant tout simplement extThopt.ExtSubstance.

3.1.1.1 Construction dans Thermoptim d'un corps pur externe

Le mécanisme de construction est le suivant :

1) l'utilisateur sélectionne un corps externe dans la liste proposée à l'écran, ce qui permet d'en repérer l'indice dans les tableaux de classes externes chargées dans Thermoptim
2) on charge cette classe, que l'on transtype en ExtSubstance, et encapsule dans un Object

```
Class c=(Class)rg.util.Util.substClasses[i];//on charge la classe du corps externe
Constructor ct=c.getConstructor(null);//on l'instancie avec son constructeur sans argument
extThopt.ExtSubstance ec=(extThopt.ExtSubstance)ct.newInstance(null);
Object ob=ec;//on l'entoure dans un Object (sinon, l'argument ne passe pas !)
corps=(Corps)new CorpsExterne(ob);//on instancie le corps externe
```

ce qui est fait par le constructeur suivant :

```
public CorpsExterne(Object obj){
   subst=(ExtSubstance)obj;
   setNom(subst.getType());
   setComment(subst.chemForm);
   initCorpsExt(subst.M, subst.PC, subst.TC, subst.VC, subst.Tmini, subst.Tmaxi,
   subst.Pmini, subst.Pmaxi, subst.typeCorps);
}
```

3.1.1.2 Création pratique d'un corps pur externe

Pour créer un corps externe, il suffit de sous-classer extThopt.ExtSubstance. Examinons le cas de la classe DowThermA.

Le constructeur est le suivant :

```
public DowThermA (){
      super();
      type=getType();//type de corps
      M=166; PC=31.34; TC=752.15; //initialisations (masse molaire, pression et températures critiques)
      Tmini=293.15;  Tmaxi=523.15; //températures minimale et maximale de définition du corps
      chemForm="(C6H5)2O (73.5% vol), (C6H5)2 (26.5% vol)";  //composition chimique du corps
   }
```

La méthode getType() renvoie le type de corps tel qu'il apparaîtra dans les écrans de Thermoptim :

```
public String getType(){
      return "Dowtherm A";
   }
```

La méthode getClassDescription () permet de saisir un bref descriptif de la classe, qui apparaîtra dans l'écran de visualisation des classes externes (cf. section 3.6). Donnez quelques indications sur le modèle retenu, et si possible faites référence à une documentation plus détaillée.

```
public String getClassDescription(){
        return "data from FLUIDFILE software by Dow Chemical\n\nauthor : R. Gicquel   april 2003";
}
```

Les méthodes de calcul des propriétés du corps sont présentées plus loin.

3.1.2 Calculs sur les corps purs externes

3.1.2.1 Principes de correspondance retenus

Pour les calculs, les arguments sont passés par les méthodes non camouflées. On se reportera à l'annexe 1 pour le détail des méthodes existantes.

Par exemple, l'inversion en T de l'enthalpie est calculée dans extThopt.CorpsExterne par :

```
public double getT_from_hP(double hv,double P){
   return subst.getT_from_hP(hv,P);
}
```

La classe d'interface rg.corps.CorpsExt assure la correspondance entre inv_hp_T (camouflée) et getT_from_hP (non camouflée) :

```
public double inv_hp_T(double hv,double P){
   return getT_from_hP(hv, P);
}
```

Ainsi, tout appel à inv_hp_T par une méthode interne à Thermoptim pour un corps externe, est relayé à getT_from_hP par rg.corps.CorpsExt.

Dans l'autre sens, rg.corps.Corps ou rg.corps.CorpsExt contient la méthode suivante :

```
public double getT_from_hP(double hv,double P){
   return inv_hp_T(hv, P);
}
```

Un appel à getT_from_hP par une méthode externe pour un corps interne est ainsi relayé à inv_hp_T.

3.1.2.2 Exemples d'implémentation

La classe DowThermA sert à définir un liquide caloporteur utilisé dans des échangeurs. Le corps reste en permanence à l'état liquide, et les calculs que l'on souhaite effectuer sont simples : par exemple celui de l'enthalpie, fonction uniquement de la température (et de son inversion).

```
public double calcH (double T, double P,double x) {
        double a=0.0028191471215351 9,b=1.51937611940298;

        double TT=T-273.15;
```

```
        return b*TT+a/2.*TT*TT;
}
```

h étant donné par un polynôme du second degré en T, son inversion est explicite :

```
public double getT_from_hP(double $hv,double $P){

        double a=0.00281914712153519,b=1.51937611940298;
        double c=-$hv;
        a=a/2.;
        double delta=b*b-4.*a*c;
        double TT = (-b+Math.pow(delta,0.5))/2./a;

        return TT+273.15;
}
```

Les calculs sont ici très simples. S'il s'agissait de représenter une vapeur condensable, ils seraient beaucoup plus complexes. La classe LiBrH2Omixture constitue un exemple un peu plus compliqué.

3.1.3 Construction des mélanges externes

3.1.3.1 Particularités

Comme les corps purs externes, les mélanges externes sont du type extThopt. ExtSubstance, même si leurs calculs sont effectués dans un SPT. Pour les différencier, les mélanges externes doivent implémenter l'interface extThopt.ExternalMixture qui précise les méthodes qu'ils doivent définir (voir plus loin).

Les noms des corps externes purs apparaissent directement dans la liste de l'écran de sélection des corps où ils servent à instancier les classes .

Pour les mélanges externes, les choses sont nettement plus complexes, car une même classe externe peut servir à définir plusieurs systèmes, et chaque système engendre autant de compositions que l'utilisateur désire, sauf bien sûr s'il s'agit d'un corps pur, auquel cas les fractions molaire et massique valent toutes deux 1. Chaque classe externe définit l'ensemble des systèmes qu'elle propose, qui ne peuvent être modifiés par l'utilisateur depuis Thermoptim.

La classe rg.corps.ExtMixture sert à instancier les mélanges. Elle possède un Corps appelé refExternalSubstance, qui correspond à la classe externe où les calculs sont effectués, et avec laquelle elle échange le nom du système et les compositions sélectionnés, ceci grâce à la méthode de CorpsExterne (qui ne doit bien sûr pas être camouflée) :

```
   public void CalcPropCorps (double T, double p, double x, double fractType, String systType, double[]
fract_mol) {

      //on commence par mettre à jour la composition du système
      subst.updateComp(systType, fractType, fract_mol);
      subst.fractType=fractType;
      double U=20., H=10., S=0.5, V=0.01, Cv=0., Xh=0.;

      if((T>=subst.Tmini)&&(T<=subst.Tmaxi)){
         //on fait appel à la méthode CalcPropCorps du mélange externe
         double[]val=subst.CalcPropCorps (T, p, x);
         //on charge les valeurs calculées
         H=val[0];
         V=val[1];
         Cv=val[2];
         x=val[3];
         U=val[4];
         S=val[5];
```

```
    }
    else{
            String message=resIntTh_fr.getString("mess_116");
            JOptionPane.showMessageDialog(new JFrame(),message);
    }
    //remise à jour des variables du corps (obfusquées)
    setState(p, T, x,//en arguments
        U, H, S, V, Cv, Xh);//à recalculer
}
```

Cette méthode passe le relais au corps externe (subst), en lui indiquant le système sélectionné (systType) et sa composition molaire ou massique (fract_mol), l'unité utilisée (fractType), puis calcule le point.

La nécessité de retourner la valeur du titre ou de la composition à l'équilibre liquide-vapeur a amené à définir la méthode getQuality(), qui retourne un tableau de doubles correspondant à la variable X d'ExtSubstance, qui doit être correctement mise à jour lors des calculs d'inversion.

Tout mélange externe doit retourner trois méthodes définies dans l'interface extThopt.ExternalMixture, utilisées pour initialiser ExtMixture lors de son instanciation :

- public String getSoftware(), qui définit le serveur de propriétés thermodynamiques utilisé
- public Vector getMixtures(), qui définit les noms des différents systèmes pris en compte, et les corps qui les composent
- public boolean isMolarFraction(), qui vaut true si la composition doit être entrée en variables molaires, et false sinon
- public double[] getMolarWeights(), qui retourne la masse molaire globale, les masses molaires des composants, ainsi que les valeurs d'initialisation comme les températures minimale et maximale, la pression maximale…

La structure du Vector vMixtures est la suivante :

```
String[] system={"lithium bromide","water"};
    vMixtures= new Vector();
    Object[]obj=new Object[2];
    obj[0]="LiBr-H2O";
    obj[1]= system;
    vMixtures.addElement(obj);
```

On notera par ailleurs que isMolarFraction() sert uniquement à définir l'affichage de l'éditeur de mélanges : dans tous les cas, la composition transite par le tableau fract_mol, et l'unité choisie par fractType.

3.1.3.2 Coordonnées critiques ou pseudo-critiques : méthode getCriticalParameters()

Les coordonnées critiques d'un mélange dépendent de sa composition et ne peuvent donc être initialisées une fois pour toutes. Pour les faire passer à Thermoptim, le mélange externe renvoie getCriticalParameters(), dont l'implémentation pour CTPLib (il s'agit en fait des coordonnées pseudo-critiques) est donnée par :

```
public double[] getCriticalParameters(){
        double[]props=new double[4];
        if(nbComponents==1){
                props[0] = compProp[0].Tc;
                props[1] = compProp[0].Pc/1.e5;
                props[2] = compProp[0].Vc;
                props[3] = compProp[0].M;
        }
        else{
           props[0] = CalcTcMix();
           props[1] = CalcPcMix()/1.e5;
           props[2] = CalcVcMix();
           props[3] = CalcMMix();
        }
```

```
        return props;
    }
```

Le passage de valeurs se fait lors du calcul du corps.

Pour un mélange externe, le calcul de l'état du corps depuis Thermoptim est relayé à void rg.corps.ExtMixture.etat_complet(double TT, double pp, double x), qui effectue le calcul depuis la classe par :

refExternalSubstance.CalcPropCorps (TT, pp, x, fractType, systType, fract_mol);//modRG 23/01

puis récupère les valeurs par Vector v=refExternalSubstance.getSubstProperties();

et actualise toutes les valeurs, y compris le coordonnées critiques

Lors du calcul de refExternalSubstance.CalcPropCorps, les valeurs critiques sont déterminées par :

```
double[]prop=getCriticalParameters();
subst.TC=prop[0];
subst.PC=prop[1];
subst.VC=prop[2];
subst.M=prop[3];
```

getCriticalParameters() renvoyant les bonnes valeurs.

3.1.3.3 Fichiers associés aux mélanges externes

A l'instar des gaz composés, la liste des mélanges externes est sauvegardée dans un fichier appelé "mel_ext.txt", qui est mis à jour chaque fois qu'un nouveau mélange est créé, et sert à initialiser l'écran de sélection des corps :

```
fichier de mélanges externes
CO2_TEP TEPThermoSoft mixtures CO2     CO2;1.0;0.0
eau_ThBd        ThermoBlend mixtures   water   water;1.0;1.0
R407C_  ThermoBlend mixtures   R407c_  R32;0.38111;0.3811     R125;0.17956;0.1795
        R134A;0.43933;0.4394
CO2_ThBd        ThermoBlend mixtures   CO2     CO2;1.0;1.0
liqu pauvre     TEPThermoSoft mixtures NH3-H2O ammonia;0.25;0.0       water;0.75;0.0
vap riche       TEPThermoSoft mixtures NH3-H2O ammonia;0.75;0.0       water;0.25;0.0
R404a   ThermoBlend mixtures   R404A   R125;0.3578;0.3578     R143A;0.6039;0.6039
        R134A;0.0383;0.0383
mel_pauvre      LiBr-H2O external mixture      LiBr-H2O        lithium bromide;1.0;0.3
        water;0.0;0.7
mel pauvre      TEPThermoSoft mixtures NH3-H2O ammonia;0.1;0.0        water;0.9;0.0
LiBrH2O LiBr-H2O mixture       LiBr-H2O        lithium bromide;0.8;0.0       water;0.2;0.0
R407C   ThermoBlend mixtures   R407C   R143a;0.23;1.0 R125;0.25;0.0  R134a;0.52;0.0
R410A   ThermoBlend mixtures   R410A   R143a;0.5;1.0  R125;0.5;0.0
new     ThermoBlend mixtures   NH3-H2O Ammoniac;0.3;0.3       water;0.7;0.7
Fin
Fin
```

3.1.3.3 Exemple de mélange externe : le système LiBr-H2O

Le système LiBr-H2O constitue un exemple simple de définition d'un mélange externe. Nous donnerons plus loin un exemple plus complexe relatif au couplage avec le SPT TEP ThermoSoft, car sa présentation ne peut se faire qu'en prenant en compte les caractéristiques de ce SPT. Un autre exemple relatif au couplage avec le SPT TEP Lib est détaillé sur le portail Thermoptim UNIT[1]. Comme ce dernier est lui-même écrit en Java, le couplage est particulièrement simple.

Attention : ce système est modélisé dans les classes externes de Thermoptim de deux manières : sous le nom de H2OLiBrMixture en tant que mélange externe, et sous le nom de LiBrH2OMixture en tant que corps externe simple. Dans ce cas, la fraction massique du mélange en LiBr X utilise le champ habituellement dévolu au titre x.

[1] http://www.thermoptim.org/sections/logiciels/thermoptim/ressources/serveur-proprietes/

Dans les machines utilisant le mélange LiBr-H_2O, la différence de tension de vapeur du solvant (LiBr) et du soluté (H_2O) est telle que l'on peut négliger la fraction massique du solvant en phase vapeur, ce qui permet de simplifier les calculs. Notons en passant que l'usage est de paramétrer le diagramme du couple LiBr-H_2O en fonction du titre massique en solvant (LiBr) et non en soluté. L'eau étant susceptible de cristalliser à basse température, on fait souvent apparaître sur le diagramme la courbe de cristallisation du mélange, qui correspond à une limite inférieure de fonctionnement des machines.

Pour ce couple, l'ASHRAE[2] propose les équations (1) et (2), établies en généralisant au mélange la loi de tension saturante du réfrigérant (l'eau), dans laquelle la température de l'eau t' (°C) est remplacée par une fonction linéaire de la température de la solution t (°C). P, exprimée à partir du logarithme décimal, est la pression en kPa, et X la fraction massique du mélange en LiBr. Ces équations sont valables dans les intervalles de valeurs suivantes : -15 < t' < 110 °C, 5 < t < 175 °C, 45 < X < 70 %.

$$\log(P) = C + \frac{D}{t' + 273,15} + \frac{E}{(t' + 273,15)^2} \qquad (1)$$

$$t' = \frac{t - \sum_{i=0}^{3} B_i X^i}{\sum_{i=0}^{3} A_i X^i} \qquad (2)$$

TABLEAU 1 COEFFICIENTS DES EQUATIONS 1 ET 2

A0	-2,00755	B0	124,937	C	7,05
A1	0,16976	B1	-7,71649	D	-1596,49
A2	-3,13E-03	B2	0,152286	E	-104095,5
A3	1,98E-05	B3	-7,95E-04		

Le constructeur initialise les bornes de validité de la classe et le Vector vMixtures, qui ne comprend ici qu'un seul système :

```
   public H2OLiBrMixture (){
      super();
      type=getType();
      M=29;PC=10;TC=350; //Attention : initialisations sans sens physique
      Tmini=278.15;  Tmaxi=448.15;
      chemForm="LiBr-H2O mixture";
      typeCorps=6;//corps externe de type isMixture
      vMixtures= new Vector();
      Object[]obj=new Object[2];
      obj[0]="LiBr-H2O";
      obj[1]=system;
      vMixtures.addElement(obj);
      mel_externes= new Hashtable();
      Vector liste_composes = new Vector();
       liste_composes.addElement("");
       liste_composes.addElement(Util.aff_i(2));
       liste_composes.addElement(system[0]);
       liste_composes.addElement(system[1]);
      mel_externes.put("LiBr-H2O",liste_composes);//chargement dans la HashTable des références du modèle
}
   String[] system={"lithium bromide","water"};
```

[2] ASHRAE, Fundamentals Handbook (SI), Thermophysical properties of refrigerants, 2001.

Les méthodes suivantes permettent de définir la description de la classe, le logiciel utilisé (comme il ne s'agit pas d'un SPT, le nom n'a ici guère d'importance), l'unité (ici massique) à utiliser pour définir la composition, le système proposé, et l'identifiant de la classe.

```
        public String getClassDescription(){
           return "external mixture class\nWatch out! the LiBr composition is to be expressed as mass
fractions";
        }

   public String getSoftware(){
      return "rg";
   }

   public boolean isMolarFraction(){
      return false;
   }
   public Vector getMixtures(){
      return vMixtures;
   }

   public String getType(){
      return "LiBr-H2O external mixture";
   }
```

La mise à jour de la composition se fait à partir de la méthode suivante, qui utilise la variable intermédiaire x, égale à X/100.

```
   public void updateComp(String systType, double[] fract_mass){
   //attention : x représente bien la valeur massique, même s'il passe par fract_mol dans ExtMixture
         selectedSyst=systType;
         x=fract_mass[0];
   }
```

Les autres méthodes (non présentées ici) définissent les calculs à effectuer, correspondant notamment à la résolution des équations (1) et (2).

3.1.4 Instanciation depuis les classes externes

Il est possible d'instancier directement un mélange externe depuis les classes externes, par exemple dans un pilote. La syntaxe est présentée ci-dessous. Il faut simplement indiquer le système choisi (ici "LiBr-H2O", et faire une mise à jour de la composition (ici particulièrement simple). Les fonctions de calcul et d'inversion sont alors directement utilisables, et les résultats peuvent être écrits, par exemple dans le fichier de sortie.

```
        H2OLiBrMixture monCorps=new H2OLiBrMixture ();//instanciation du corps
        double[]fractmass={0.35,0.65};
        monCorps.updateComp("LiBr-H2O",fractmass);
        System.out.println("Enthalpie du mélange externe LiBr-H2O pour la composition massique :
"+fractmass[0]+" LiBr");
        for(int i=0;i<10;i++){
           double T=280+10*i;
                 double[] val=monCorps.CalcPropCorps(T, 5, 0);
                 System.out.println("T :\t"+T+"    h : \t"+val[0]);
        }
```

ce qui donne :

```
Enthalpie du mélange externe LiBr-H2O pour la composition massique : 0.35
LiBr
T :   280.0    h :      -5.2873148226108775
T :   290.0    h :      21.108948694345447
```

```
T :   300.0    h :      47.56170686221427
T :   310.0    h :      74.07095968099561
T :   320.0    h :      100.63670715068943
T :   330.0    h :      127.25894927129576
T :   340.0    h :      153.9376860428146
T :   350.0    h :      180.67291746524594
T :   360.0    h :      207.46464353858977
T :   370.0    h :      234.3128642628461
```

3.1.5 Classe de couplage avec le serveur de propriétés TEP ThermoSoft

Le couplage avec un serveur de propriétés thermodynamiques (SPT) comme TEP ThermoSoft est nettement plus complexe, d'une part parce que ce SPT est constitué de modules de calcul indépendants de Thermoptim, qui plus est développés dans un autre langage, et d'autre part parce que les systèmes thermodynamiques calculables peuvent être très variables selon les cas. Nous illustrerons la procédure à suivre autour de l'exemple de la classe TEPThermoSoft.java.

3.1.5.1 Interfaçage entre Java et Delphi

La méthode suivante est utilisée pour charger la bibliothèque "TEPThermoSoftJava.dll" qui constitue l'interface avec l'environnement Delphi dans lequel a été développé TEP ThermoSoft :

```
static {
      System.loadLibrary("TEPThermoSoftJava");
   }
```

Il faut ensuite déclarer les méthodes Delphi appelables depuis Java et qui servent à dialoguer entre les deux environnements (leur fonction et leur syntaxe sont détaillées en annexe 4, à laquelle nous vous recommandons de vous référer).

```
public native int InitSession(String RepMODELES);
public native void FermerSession();
public native int ChargerSysteme(String FichierMEL);
public native double Lire(String Symbole);
public native void Ecrire(String Symbole, double valeur);
public native void Calculer(String SymbCalc);
```

3.1.5.2 Définition des mélanges proposés par le SPT

Chaque SPT doit définir une liste des systèmes proposés, qui est placée dans le dossier "mixtures" du répertoire d'installation. Son format standard est le suivant :

```
fichier de mélanges externes
NH3-H2O      NH3-H2O.mel  2      ammonia      water        M=44   PC=200 TC=404.128
      Tmaxi=1100   Tmini=216.7  T0=293.15    P0=1.
CO2   CO2.mel      1      CO2    M=44   PC=73.77     TC=304.128   Tmaxi=1100
      Tmini=216.7  T0=273.15    P0=36.59027
Fin
Fin
```

Chaque ligne comprend :

- le nom du système tel qu'il apparaîtra dans les écrans de choix de Thermoptim (p. ex. `NH3-H2O`)
- le fichier de définition du mélange, au format du serveur de propriétés (p. ex. `NH3-H2O.mel`)
- le nombre de composants
- les noms des composants tel qu'ils apparaîtront dans les écrans de choix de Thermoptim
- la masse molaire, les pression et température critique, les températures maxi et mini pour le fluide, et les valeurs de la pression et de la température de référence (pour lesquelles h=u=s=0)

C'est fondamentalement par le biais du fichier de définition du mélange qu'est faite l'initialisation du serveur de propriétés, Thermoptim se contentant de modifier la composition et les variables d'état.

On remarquera dans cette liste la présence du système "CO2" à un seul composant, c'est-à-dire d'un corps pur. Le mécanisme des mélanges externes permet en effet aussi d'émuler un SPT pour représenter un corps pur par un modèle plus précis que celui qui est implémenté dans Thermoptim. Dans ce cas précis, il s'agit d'une équation dédiée, utilisable au voisinage du point critique[3].

L'arborescence des répertoires du serveur de propriétés est libre, les seules contraintes étant que les dll de liaison soient placées dans le répertoire d'installation, et la liste des systèmes proposés dans le dossier "mixtures". Dans le cas de TEP ThermoSoft, cette liste est contenue dans le fichier "TEPThSoft.mix".

Le constructeur de la classe TEPThermoSoft.java diffère peu de celui de la classe précédemment étudiée :

```
public TEPThermoSoft (){
    super();
    type=getType();
    M=44;PC=73.77;TC=304.128;
    Tmaxi=1100;
    Tmini=216.7;
    chemForm="TEP ThermoSoft mixture properties";
    vMixtures= new Vector();
    mel_externes= new Hashtable();
    getMixtures();
}

public Vector getMixtures(){

    vMixtures= new Vector();
    nomfich= new StringBuffer("TEPThSoft.mix");
    dir_data=System.getProperty("user.dir")+File.separator+"mixtures"+File.separator;
    fich = new File(dir_data,nomfich.toString());
    lect_data(fich);
    return vMixtures;
}
```

Il introduit une Hashtable pour pouvoir référencer facilement les mélanges externes existants, et exécute la méthode getMixtures() qui construit le Vector des mélanges existants et la Hashtable, en analysant le fichier de mélanges "TEPThSoft.mix" grâce à la méthode lect_data(). La même classe peut ainsi être utilisée avec des jeux variés de mélanges en fonction des applications.

Les trois méthodes de l'interface ExternalMixture sont ensuite définies, ce qui ne pose aucun problème particulier.

La méthode updateComp doit être capable à la fois d'initialiser le système correctement et de mettre à jour sa composition. Pour ce faire, il fait appel à la Hashtable créée par le constructeur.

```
public void updateComp(String systType, double[] fractmol){

        selectedSyst=systType;
        //récupération des informations sur le système considéré par la Hashtable
        Vector liste_comp = (Vector)(mel_externes.get(selectedSyst));

        selectedFile=(String)liste_comp.elementAt(0);//fichier de mélange du système (XXX.mel)
        int nbComp=Util.lit_i((String)liste_comp.elementAt(1));//nombre de composants du système
        fract_mol = fractmol;
        components=new String[nbComp];
        for(int i=0;i<fract_mol.length;i++){
           components[i]=(String)liste_comp.elementAt(i+2);//noms des composants
        }
        //coordonnées critiques (pour les corps purs)
        PC=Util.lit_d((String)liste_comp.elementAt(nbComp+2));
        TC=Util.lit_d((String)liste_comp.elementAt(nbComp+3));
        //intervalle de définition des modèles utilisés
        Tmaxi=Util.lit_d((String)liste_comp.elementAt(nbComp+4));
        Tmini=Util.lit_d((String)liste_comp.elementAt(nbComp+5));
        //Pression et température des enthalpie, entropie et énergie interne de référence
        P0=Util.lit_d((String)liste_comp.elementAt(nbComp+6));
        T0=Util.lit_d((String)liste_comp.elementAt(nbComp+7));
}
```

[3] W. Wagner, R. Span, A new equation of state for carbon dioxide covering the fluid region from the triple-point temperature to 1100 K at pressures up to 800 MPa. J. Phys. Chem. Ref. Data, 25(6):1509, 1996.

Un exemple de calcul est donné ci-dessous pour l'obtention de la température de saturation (bulle si x=0, rosée si x=1). Il illustre bien l'enchaînement des appels aux méthodes de TEP ThermoSoft documentées en annexe 4. Schématiquement, on commence par initialiser la session et on charge le système considéré, puis on initialise la pression, convertie de bar en Pascal, et la composition du mélange. Selon qu'il s'agit d'un corps pur ou d'un véritable mélange, l'exécution du calcul diffère légèrement. La session de travail est fermée pour libérer les ressources, et la température de saturation est renvoyée à Thermoptim.

Afin d'accélérer les calculs, il est préférable d'éviter d'ouvrir et de refermer une session lorsqu'on peut l'éviter. Pour cela, des variantes de certaines méthodes utilisables uniquement lorsqu'une session est ouverte ont été implémentées. Elles diffèrent des autres par le terme de Session ajouté à leur nom.

Dans ce cas, on a créé deux méthodes sœurs : getSatTemperature() et getSatTemperatureSession(), la première gérant l'ouverture et la fermeture des sessions, et la seconde faisant les calculs. Ceci permet de faire des appels directs à la seconde sans avoir à réinitialiser le système.

```
public double getSatTemperature(double P, double x){
    /* Initialisation du répertoire contenant les fichiers .dll */
    InitSession(System.getProperty("user.dir")+File.separator+"TEPThermoSoft_DLL"+File.separator);
    ChargerSysteme(System.getProperty("user.dir")+File.separator+"mixtures"+File.separator+"TEPThermoSoft_MEL"+File

    double Tsat;
    Tsat=getSatTemperatureSession(P, x);

    /* Libère la mémoire */
    FermerSession();

    return Tsat;
}

public double getSatTemperatureSession(double P, double x){
    double Tsat;
 // Paramètres de calcul
    Ecrire("P", P*1e5);
    if(fract_mol.length==1)Calculer("P#");
    else {
        if(x==0){
            for(int i=0;i<fract_mol.length;i++){
                String ref="x("+Util.aff_i(i+1)+")";
                Ecrire(ref,fract_mol[i]);
            }
            // Lance le Calcul
            Calculer("Px#");
        }
        if(x==1){
            for(int i=0;i<fract_mol.length;i++){
                String ref="y("+Util.aff_i(i+1)+")";
                Ecrire(ref,fract_mol[i]);
            }
            // Lance le Calcul
            Calculer("Py#");
        }
   }
    Tsat=Lire("T"); //calcS(T,p,x);

    return Tsat;
}
```

Les autres méthodes de calcul implémentées sont plus complexes, notamment celles qui font appel aux fonctions d'inversion par dichotomie, mais leur principe est le même. On se reportera aux commentaires documentant la classe pour plus de précisions.

3.1.5.3 Inversion des fonctions

Un exemple est donné ci-dessous pour la méthode d'inversion en T de l'enthalpie, la pression étant connue.

La procédure d'ensemble est analogue : lors des initialisations, une session est ouverte, et le système est chargé. Le calcul d'inversion peut alors être effectué dans cette session, la fonction f_dicho faisant appel à la version de CalcPropCorps qui n'ouvre pas sa propre session. Lorsque les calculs sont terminés, la session est fermée.

```
public double getT_from_hP(double hv,double P){
    /* Initialisation du répertoire contenant les fichiers .dll */
    InitSession(System.getProperty("user.dir")+File.separator+"TEPThermoSoft_DLL"+File.separator);
    ChargerSysteme(System.getProperty("user.dir")+File.separator+"mixtures"+File.separator+"TEPThermoSoft_ME

    double Tcalc;
    this.hv=hv;
    xx=0;
    if(fract_mol.length==1)X=new double[fract_mol.length]; //X est redimensionné en fonction du système cons
    else X=new double[2*fract_mol.length+1];

    Tcalc=Util.dicho_T(this, hv, P, "getT_from_hP", Tmini, Tmaxi, 0.0001);

    /* Libère la mémoire */
    FermerSession();

    return Tcalc;
}

public double f_dicho(double T, double P,String fonc){
    double[]result=new double[10];

    if (fonc.equals("getT_from_hP")){
        result=CalcPropCorpsSession(T,P,xx);//les méthodes appelées utilisent la session déjà ouverte
        if((result[7]>hv)&&(result[6]<hv)){//si ELV et hv compris entre les hlsat et hvsat
            xx=(hv-result[6])/(result[7]-result[6]);//on calcule alors le titre
            result=CalcPropCorpsSession(T,P,xx);//on refait une dernière passe maintenant que x est calculé
            X[0]=xx;//on charge le titre dans X(0)
        }
        return result[0];
    }
```

On notera une particularité de cette méthode d'inversion : lors d'un équilibre liquide-vapeur, pour un corps pur, le renvoi de la température ne suffit pas ; il faut aussi déterminer le titre en vapeur x. Celui-ci est calculé dans f_dicho, puis chargé dans X[0], auquel Thermoptim a accès grâce à la méthode getQuality() d'ExtSubstance.

Pour le moment, seul le titre des corps purs est exploité par les classes internes de Thermoptim, mais, comme le montre la section suivante, le mécanisme implémenté dans TEPThermosoft.java est générique et permet aux classes externes d'accéder à la composition des mélanges multicomposants.

3.1.5.4 Exemple de calcul de mélange externe (NH3-H2O)

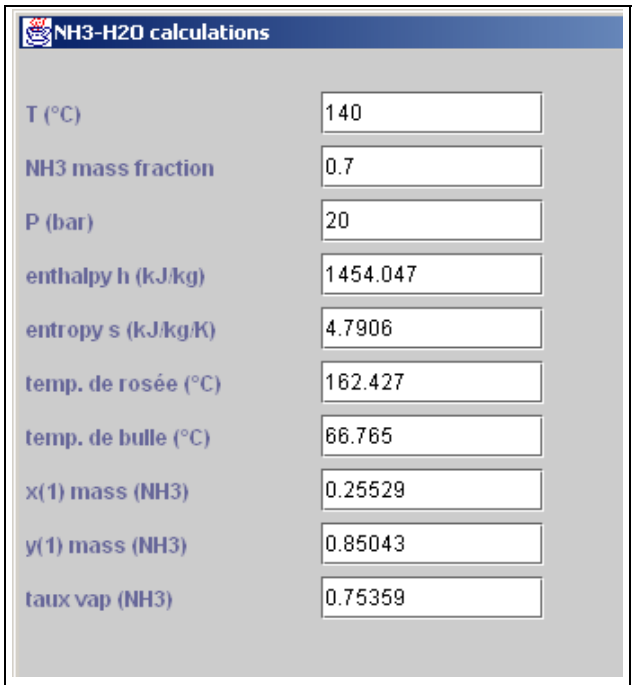
TEP Thermosoft propose le couple NH3-H2O comme mélange. Afin d'illustrer les calculs qu'il permet, nous avons créé une petit classe appelée NH3_Absorption.java, qui permet d'afficher un écran des principales propriétés utiles pour le calcul des systèmes à absorption mettant en jeu ce mélange.

Comme c'est l'usage pour cette application, la composition est entrée en variables massiques.

Dans l'exemple ci-contre, à 140 °C et pour une fraction massique de NH3 égale à 0,7, le point se situe à l'intérieure de la lentille isobare à 20 bars, les températures de bulle et de rosée étant respectivement de 66,8 °C et 162,4 °C.

TEP Thermosoft renvoie donc les fractions molaires liquide et vapeur de NH3, ainsi que le taux de vaporisation moyen. Leurs valeurs sont transmises par le tableau double[] X, qui, dans le cas d'un corps pur, ne contient que le titre en vapeur.

Le code permettant d'effectuer les calculs dans la classe NH3_Absorption.java illustre l'utilisation du tableau X par une classe externe :



Calculateur de propriétés du couple NH3-H2O

```
void bCalcul_actionPerformed(java.awt.event.ActionEvent event)
{
     TEPThermoSoft monCorps=new TEPThermoSoft();//instanciation du corps
     double x1Mass=Util.lit_d(fractmassNH3.getText());
     double x1Mol=18.0153*x1Mass/(17.0306+x1Mass);//fraction molaire
     double[]fractmol={x1Mol,1.-x1Mol};
     monCorps.updateComp("NH3-H2O",fractmol);//choix du système et mise à jour de la composition

     double T=Util.lit_d(T_value_C.getText())+273.15;
     double P=Util.lit_d(P_value.getText());
     System.out.println("\n\nt : "+T_value_C.getText()+"  P : "+P+"  x1(mol) : "+x1Mol+"  x1(mass) : "+x1Mass);
     double Trosee=monCorps.getSatTemperature(P, 1)-273.15;
     Trosee_value.setText(Util.aff_d(Trosee,3));

     double Tbulle=monCorps.getSatTemperature(P, 0)-273.15;
     Tbulle_value.setText(Util.aff_d(Tbulle,3));

     double[]result=monCorps.CalcPropCorps(T,P,1);
     h_value.setText(Util.aff_d(result[0],3));
     s_value.setText(Util.aff_d(result[5],4));
     x1_value.setText(Util.aff_d((17.0306*monCorps.X[1])/(18.0153-monCorps.X[1]),5));
     y1_value.setText(Util.aff_d((17.0306*monCorps.X[3])/(18.0153-monCorps.X[3]),5));
     x_value.setText(Util.aff_d(result[3],5));
}
```

3.1.6 Diagrammes thermodynamiques des mélanges externes

Les mélanges externes n'étant pas inclus dans Thermoptim, le progiciel ne dispose pas de ses diagrammes thermodynamiques. Pour pallier cette limitation, nous avons ajouté un nouveau type de diagramme, appelé diagramme de mélange externe, qui permet d'utiliser des diagrammes entropiques et des frigoristes simplifiés.

La préparation des fonds de diagramme fait appel à une classe externe particulière, appelée CreateMixtureCharts.java. On se réfèrera à sa documentation pour sa présentation.

Les nouveaux diagrammes sont simplifiés par rapport aux autres en ce sens qu'ils ne présentent que les courbes de bulle et de rosée, ainsi qu'un seul jeu d'isovaleurs, les isobares pour le diagramme entropique, et les isothermes pour le diagramme des frigoristes.

Ces diagrammes étant une variante des diagrammes des vapeurs, leur utilisation est expliquée dans la documentation de ces derniers

3.2 Transfos externes

La structure des classes externes a été définie plus haut : la classe extThopt.TransfExterne hérite de rg.corps.TransfoExterne de Thermoptim et fait l'interface, relayant les calculs au niveau d'une classe abstraite (extThopt.ExtProcess) définissant les méthodes de base, et sous-classée par les différentes classes introduites (par exemple extThopt.SolarCollector). Nous vous recommandons de vous référer à l'API de extThopt.TransfExterne et extThopt. ExtProcess pour connaître l'ensemble des méthodes disponibles, leur syntaxe et leur fonction. Cette API fait partie de l'environnement de développement des classes externes mis à votre disposition (dans le dossier api_extThopt).

Nous commencerons par expliquer le mécanisme de construction qui est implémenté dans Thermoptim, puis nous montrerons comment créer en pratique un nouveau composant externe, en sous-classant tout simplement extThopt. ExtProcess.

3.2.1 Construction

3.2.1.1 Construction dans Thermoptim d'une transfo externe

Le mécanisme de construction est le suivant :

1) l'utilisateur sélectionne une transfo externe dans la liste proposée à l'écran, ce qui permet d'en repérer l'indice dans les tableaux de classes externes chargées dans Thermoptim
2) on charge cette classe, que l'on transtype en ExtProcess, et encapsule dans un Object

Class c=(Class)rg.util.Util.componentClasses[i];//on charge la classe du composant externe
Constructor ct=c.getConstructor(null);//on l'instancie avec son constructeur sans argument
extThopt.ExtProcess ec=(extThopt.ExtProcess)ct.newInstance(null);

```
Object ob=ec;//on l'entoure dans un Object
```

3) on instancie alors la classe TransfExterne, qui hérite de ComposantExt, c'est-à-dire est en fait un JPanel contenant l'interface utilisateur définie dans la transfo externe

```
JPanel1.remove(cType);
cType=new TransfExterne((Projet)Proj,ob, this);//on instancie le composant externe
setupCType();//met en place l'interface utilisateur externe
```

ce qui est fait par le constructeur suivant :

```
public TransfExterne(Projet proj, Object obj, TransfoExterne te){
   this.proj=proj;
   this.te=te;//on passe ainsi la référence à la TransfoExterne interne à Thermoptim
   ep= (ExtProcess)obj;//récupère la classe externe instanciée dans TransfoExterne
   ep.proj=proj;//récupère la référence du projet
   Vector vSetUp=new Vector();
   vSetUp.addElement(te);//renvoi de la référence à la TransfoExterne
   vSetUp.addElement(ep.JPanel1);//chargement de l'interface
   vSetUp.addElement(ep.thermoCouplerTypes);//définition des types de thermocoupleurs requis
   setUpFrame(vSetUp);//exécution de setCompFrame();
   ep.tfe=this;
}
```

4) si la construction se fait lors de la lecture d'un fichier de projet, les paramètres de la transfo sont mis à jour

5) lorsqu'un projet est entièrement chargé, Thermoptim fait exécuter dans chaque composant externe une méthode particulière appelée init() qui permet en particulier de procéder à des initialisations faisant référence aux autres composants externes instanciés (cf. section 3.4). Il est ainsi possible de synchroniser leurs méthodes.

3.2.1.2 Création pratique d'une transfo externe

Pour créer une transfo externe, il suffit de sous-classer extThopt.ExtProcess. Examinons le cas de la classe SolarCollector (nous n'avons donné ici qu'une partie de la construction de l'interface graphique).

```
public SolarCollector (){
      super();
                  JPanel1.setLayout(null);//Layout du JPanel
                  JPanel1.setBounds(0,0,400,300);//dimensions du JPanel (a priori standard)
                  JLabel1.setText("glass transmittivity");//définition du premier label
                  JPanel1.add(JLabel1);
                  JLabel1.setBounds(0,0,164,24);
                  JPanel1.add(tau_value);//définition du premier champ de texte éditable pour entrer la
transmittivité du vitrage
                  tau_value.setBounds(164,0,124,24);
                  tau_value.setText("0.8");
                  type=getType();//type de transfo
                  thermoCouplerTypes=new String[0];//aucun thermocoupleur connecté
 }

   public String getType(){
      return "solar collector";
    }

   public String getClassDescription(){
            return "flat plate solar collector (without thermocoupler)\n\nauthor : R. Gicquel  january 2003\n\nRef
         : note MODÉLISATION D'UN CAPTEUR SOLAIRE THERMIQUE";
  }
```

3.2.2 Mise à jour et calcul de la transfo

On se réfère ici à l'exemple de la classe SolarCollector, dont un modèle un peu plus général, valable pour les capteurs à concentration, est présenté dans la note "ModeleCaptSolaireConcentr.pdf" du portail[4].

L'enchaînement des opérations est le suivant :

1) mise à jour du composant avant calcul par chargement des valeurs de la transfo et du point amont
2) lecture des paramètres sur l'écran du composant externe
3) calcul de la puissance mise en jeu et de l'état du point aval
4) calcul des charges thermiques des thermocoupleurs
5) mise à jour de l'écran du composant externe

3.2.2.1 Présentation du code

Examinons maintenant les problèmes rencontrés en pratique à chacune de ces étapes. Quelques extraits du code sont donnés ci-dessous, mais il est recommandé de lire la suite de cette note en ayant sous les yeux l'intégralité de la classe SolarCollector.java.

1) mise à jour du composant par chargement des valeurs de la transfo et du point amont

La petite difficulté est ici qu'un composant externe n'a pas un accès direct aux variables du simulateur : ces grandeurs sont obtenues par des méthodes particulières génériques, qui construisent des Vector de structure différente selon l'objet désiré.

La marche à suivre n'est pas compliquée mais demande à être respectée :

```
String[] args=new String[2]; //tableau des arguments
args[0]="process";//type d'élément désiré (ici une transfo)
args[1]=tfe.getCompName();//nom de la transfo (obtenu par la référence tfe)
Vector vProp=proj.getProperties(args);//méthode de Projet donnée en annexe 2
Double f=(Double)vProp.elementAt(3);
double flow=f.doubleValue();//valeur du débit, propagé automatiquement depuis la transfo amont
String amont=(String)vProp.elementAt(1);//nom du point amont
getPointProperties(amont);//décodage automatique du Vector (méthode de ExtProcess)
Tamont=Tpoint;//ici T1
```

La méthode getPointProperties() d'ExtProcess charge automatiquement l'état d'un point dans des valeurs aisément manipulables, appelées Tpoint, Ppoint, lecorps… Elle est donnée ci-dessous

```
public void getPointProperties(String nom){
        String[] args=new String[2];
        args[0]="point";//type of the element (see method getProperties(String[] args))
        args[1]=nom;//name of the process (see method getProperties(String[] args))
        Vector vProp=proj.getProperties(args);
   lecorps=(Corps)vProp.elementAt(0);
   nomCorps=(String)vProp.elementAt(1);
   Double y=(Double)vProp.elementAt(2);
   Tpoint=y.doubleValue();
   y=(Double)vProp.elementAt(3);
        Ppoint=y.doubleValue();
   y=(Double)vProp.elementAt(4);
        Xpoint=y.doubleValue();
   y=(Double)vProp.elementAt(5);
        Vpoint=y.doubleValue();
   y=(Double)vProp.elementAt(6);
        Upoint=y.doubleValue();
```

[4] http://www.thermoptim.org/sections/logiciels/thermoptim/modelotheque/modele-capt-concentr

```
   y=(Double)vProp.elementAt(7);
         Hpoint=y.doubleValue();
   y=(Double)vProp.elementAt(9);
         DTsatpoint=y.doubleValue();
         String dum=(String)vProp.elementAt(8);
         isTsatSet=Util.lit_b(Util.extr_value(dum));
         dum=(String)vProp.elementAt(10);
         isPsatSet=Util.lit_b(Util.extr_value(dum));
}
```

2) lecture des paramètres sur l'écran du composant externe

Le package extThopt fournit un certain nombre de méthodes simples mais robustes pour convertir en double les String affichés dans les champs JTextField utilisés dans l'interface graphique, et réciproquement pour afficher les double dans ces champs. Ils sont implémentés sous forme de méthodes statiques de la classe extThopt.Util (cf. annexe 3) :

```
P=Util.lit_d(P_value.getText());
A=Util.lit_d(A_value.getText());
tau=Util.lit_d(tau_value.getText());
K=Util.lit_d(K_value.getText());
Tex=Util.lit_d(Tex_value.getText())+273.15;
```

3) calcul de la puissance mise en jeu et de l'état du point aval

On commence par estimer le Cp du fluide caloporteur en faisant un développement limité de la fonction enthalpie, ce qui amène à utiliser la méthode CalcPropCorps() du package Corps, et la méthode getSubstProperties() d'ExtProcess, qui charge automatiquement l'état d'un point dans des valeurs aisément manipulables, appelées Tsubst, Psubst, etc. :

```
public void getSubstProperties(String nom){
         String[] args=new String[2];
         args[0]="subst";//type of the element (see method getProperties(String[] args))
         args[1]=nom;//name of the process (see method getProperties(String[] args))
         Vector vProp=proj.getProperties(args);
   Double y=(Double)vProp.elementAt(0);
   Tsubst=y.doubleValue();//température
   y=(Double)vProp.elementAt(1);
         Psubst=y.doubleValue();//pression
   y=(Double)vProp.elementAt(2);
         Xsubst=y.doubleValue();//titre
   y=(Double)vProp.elementAt(3);
         Vsubst=y.doubleValue();//volume massique
   y=(Double)vProp.elementAt(4);
         Usubst=y.doubleValue();//énergie interne massique
   y=(Double)vProp.elementAt(5);
         Hsubst=y.doubleValue();//enthalpie massique
   y=(Double)vProp.elementAt(6);
         Ssubst=y.doubleValue();entropie massique
y=(Double)vProp.elementAt(7);
      Msubst=y.doubleValue();//masse molaire
      Integer i=(Integer)vProp.elementAt(8);
      typeSubst=i.intValue();//type de corps (1 pour l'eau, 2 pour une vapeur, 3 pour un gaz pur, 4 pour un gaz
composé, 5 pour un corps externe)
      y=(Double)vProp.elementAt(13);
      ChemExerSubst=y.doubleValue();
 }
```

Ceci permet de calculer le Cp comme suit :

```
double H=Hpoint;//enthalpie du point amont
lecorps.CalcPropCorps(Tpoint+1, Ppoint, Xpoint);// recalcul du corps amont par une fonction de Thermoptim
getSubstProperties(nomCorps);//récupération des valeurs du recalcul (méthode de ExtSubstance)
double Cp=(Hsubst-H);//valeur estimée de Cp
```

On calcule alors une valeur estimée de Taval pour pouvoir déterminer la puissance thermique absorbée Qex :

```
double DT0=tau*P/K-Tamont+Tex;
double T=Tamont+(DT0)*(1-Math.exp(-K*A/flow/Cp));
double DT=T-Tpoint;
double Qex=Cp*DT*flow;
```

On détermine la valeur de l'enthalpie massique du point aval, puis on inverse cette fonction pour déterminer la valeur exacte de Taval (méthode de Corps)

```
double hAval=Qex/flow+Hpoint;
Tpoint=lecorps.getT_from_hP(hAval,Ppoint);

getSubstProperties(nomCorps);//récupération des valeurs du recalcul (méthode de ExtProcess)
Xpoint=Xsubst;//mise à jour de la valeur du titre pour le cas où l'état est diphasique
```

4) calcul des charges thermiques des thermocoupleurs

Dans cet exemple simple, le problème ne se pose pas, le composant ne mettant pas en jeu de thermocoupleur. Quelques indications sont données plus loin, ainsi que dans la section relative aux nœuds externes.

5) mise à jour de l'écran du composant externe

La méthode de Thermoptim setupPointAval() permet de mettre à jour le point aval à partir des valeurs chargées dans un Vector construit ici par la méthode getProperties() de ExtProcess :

```
tfe.setupPointAval(getProperties());
```

La valeur du rendement du capteur est ensuite déterminée et affichée.

```
eff_value.setText(Util.aff_d(Qex/P/A, 4));
```

3.2.2.2 Utilisation de classes PointThopt et CorpsThopt

La version 1.7 a introduit deux classes destinées à faciliter l'écriture des opérations sur les points et les corps, en évitant d'avoir systématiquement à extraire les informations des Vector.

Les classes PointThopt et CorpsThopt sont décrites dans l'annexe 3. Leur utilisation dans le cas de la classe SolarCollector conduirait à revoir légèrement le code :

```
String[] args=new String[2]; //tableau des arguments
args[0]="process";//type d'élément désiré (ici une transfo)
args[1]=tfe.getCompName();//nom de la transfo (obtenu par la référence tfe)
Vector vProp=proj.getProperties(args);//méthode de Projet donnée en annexe 2
Double f=(Double)vProp.elementAt(3);
double flow=f.doubleValue();//valeur du débit, propagé automatiquement depuis la transfo amont
String amont=(String)vProp.elementAt(1);//nom du point amont
String aval=(String)vProp.elementAt(2);//nom du point aval
PointThopt amontCapt=new PointThopt(proj,"amont");//instancie le point externe
amontCapt.getProperties();/actualise l'état du point à partir du noyau
Tamont= amontCapt.T;//accès beaucoup plus explicité
```

La mise à jour du point aval est aussi beaucoup plus claire :

```
double hAval=Qex/flow+Hpoint;
```

```
Tpoint=lecorps.getT_from_hP(hAval,Ppoint);

getSubstProperties(nomCorps);//récupération des valeurs du recalcul (méthode de ExtProcess)
Xpoint=Xsubst;//mise à jour de la valeur du titre pour le cas où l'état est diphasique

PointThopt avalCapt=new PointThopt(proj,"aval");//instancie le point externe
avalCapt.T =Tpoint;//modifie T du point externe
avalCapt.X =Xpoint; //modifie X du point externe
avalCapt .update(UPDATE_T, ! UPDATE_P, UPDATE_X); //met à jour le point du noyau
```

Pour cet exemple, les mises à jour avant et après recalcul sont très simples. Dans d'autres cas, il peut être nécessaire d'accéder à d'autres données du simulateur. C'est ce qui est expliqué dans les sections suivantes.

3.2.3 Calculs humides

Il est possible d'effectuer des calculs de gaz humides à partir des classes externes. Généralement, ces calculs sont effectués au niveau d'un point, mais une méthode permet de directement modifier l'humidité d'un gaz. L'humidité du gaz est entrée en indiquant soit son humidité absolue w, soit son humidité relative ε.

Rappelons que l'on appelle **humidité relative ε** le rapport de la pression partielle de la vapeur d'eau à sa pression de vapeur saturante, et que, par définition, l'indice gs correspondant aux valeurs relatives au gaz sec, **l'humidité absolue w** est égale au rapport de la masse d'eau contenue dans un volume donné de mélange humide à la masse de gaz sec contenue dans ce même volume, soit :

$$w = \frac{y_{H2O}}{y_{gs}} = \frac{M_{H2O}\, x_{H2O}}{M_{gs}\, x_{gs}} = \frac{18\, x_{H2O}}{M_{gs}\,(1 - x_{H2O})} = \frac{18\, x_{H2O}}{\Sigma\, M_{i\_sec}\, x_{i\_sec}}$$

Cette relation permet de calculer w lorsqu'on connaît la composition du gaz sec ou du gaz humide, comme le montre notamment l'exemple donné ci-dessous.

Calculs humides concernant les points

La méthode getPointProperties() d'ExtProcess récupère les valeurs des propriétés humides d'un point par les grandeurs suivantes :

Wpoint pour l'humidité absolue w, Epsipoint pour l'humidité relative ε, Qprimepoint pour l'enthalpie spécifique q', Tprimepoint pour la température adiabatique t' (en °C), Trpoint pour la température de rosée tr (en °C), VPrimepoint pour le volume spécifique vs, Condpoint pour les condensats, et M_secpoint pour la masse molaire du gaz sec.

La méthode updatePoint donnée en annexe permet de forcer les calculs humides suivants :

"setW and calculate all", impose w (passé en dernier argument) et calcule toutes les propriétés humides
"setW and calculate q'", impose w (passé en dernier argument) et calcule toutes les propriétés humides sauf t'
"setEpsi", impose ε (passé en dernier argument)

"setEpsi and calculate", impose ε (passé en dernier argument) et calcule toutes les propriétés humides
"calcWsat", calcule wsat et toutes les propriétés humides à la saturation sauf t'
"modHum", modifie la composition du gaz pour que son humidité soit égale à Wpoint
"setGasHum", modifie la composition du gaz pour que son humidité soit égale à w passé en dernier argument

On trouvera divers exemples de mise en œuvre de cette méthode dans les classes DirectCoolingTower et WaterQuench.

Calculs humides concernant les gaz

Afin de pouvoir modifier l'humidité d'un gaz indépendamment de l'état d'un point, la méthode updateGasComp() de GazIdeal, accessible par public void updateGasComposition(Vector vComp) de Corps, a été modifiée : si le

premier élément de vComp est un Integer d'une valeur négative, un traitement particulier est effectué. L'humidité absolue passée en troisième élément de vComp est imposée au gaz.

```
else{//modifications gaz humides
   String task=(String)vComp.elementAt(1);
   String value=(String)vComp.elementAt(2);
   if(task.equals("setGasHum")){//sets the gas humidity
      double w=Util.lit_d(value);
      setGasHum(w);
   }
}
```

L'exemple ci-dessous, issu de la classe BiomassCombustion, montre comment modifier la composition d'un gaz sec pour qu'elle corresponde au gaz humide dont la fraction molaire en eau est fractH2Ofuel :

```
//calcul de l'humidité absolue du gaz
 double inlet_w=18.01528*fractH2OFuel/fuelM/(1-fractH2OFuel);// w = M_H2O x_H2O / M_gs x_gs

//mise en forme du Vector
 Vector vComp=new Vector();
  vComp.addElement(new Integer(-1));
  vComp.addElement("setGasHum");
  vComp.addElement(Util.aff_d(inlet_w));
 //modification de la composition du gaz
  NewFuelSubstance.updateGasComposition(vComp);
```

3.2.4 Calculs des bilans exergétiques

Les composants externes peuvent aussi être représentés dans une structure productive. Leur écran de bilan exergétique peut être paramétré (grâce à String[] getExergyType()) par le créateur de la classe externe, pour faire apparaître pour le moment, à l'instar de celui des transfos "échange", trois options et un champ d'entrée de valeur.

Les choix effectués par l'utilisateur sont ensuite transmis à la classe externe pour évaluation de getExergyBalance(String[] args), défini ci-dessous ; ils sont sauvegardés dans le fichier de structure. Si des paramétrages supplémentaires sont nécessaires, il est toujours possible de les définir dans l'écran du composant du schéma physique.

Autant le calcul du bilan exergétique d'un composant ne pose pas de problème particulier, autant faut-il spécifier avec soin comment ses différentes composantes doivent être prises en compte dans le bilan global, du fait que la frontière du système n'est pas la même que celle d'un composant donné.

Pour calculer son bilan exergétique, chaque composant du simulateur renvoie une méthode double[] getExergyBalance(String[] args), qui comporte les sept valeurs à prendre en compte dans le bilan global (cinq ports plus le rendement exergétique et les irréversibilités).

Pour les éléments du noyau de Thermoptim, les règles permettant de pondérer les valeurs de τ^+ et Δx_q^+ fournis au cycle peuvent être figées une fois pour toutes, mais il n'en va pas de même des composants externes, pour lesquels le concepteur doit toujours préciser quelles valeurs doivent être reprises dans le bilan exergétique global.

Les modèles de composants qui peuvent être implémentés sont tellement divers qu'il n'est pas possible de prévoir tous les cas de figure. C'est pourquoi les composants externes renvoient, en complément des sept valeurs précédentes, deux coefficients compris entre 0 et 1 (tauPlusFactor et deltaXhPlusFactor) qui permettent de pondérer la fraction du travail utile et des exergies chaleurs positives fournis au cycle.

La classe ExtProcess comporte une implémentation par défaut de String[] getExergyType() et de double[] getExergyBalance(String[] args), qui doit être sous-classée dans les composants externes. Les propriétés

exergétiques des fluides doivent être calculées à partir de l'enthalpie, de l'entropie et de T_0 fournies par getSubstProperties().

L'échange de getExergyBalance entre Thermoptim et les composants externes se fait par public double[] getCompExergyBalance(String[]args).

Pour le calcul des exergies des flux entrants et sortants, une valeur de référence est prise pour T_0 et 1 bar.

Dans PointCorpsDemo, son implémentation, sans argument, est la suivante :

```
//calcul de l'exergie de référence, l'exergie chimique ne devant pas être comptabilisée
public double getExergyReference (){//modRG oct04//modRG exerg
        lecorps.CalcPropCorps(Util.T0Exer,1,1);
        return lecorps.H-Util.T0Exer*lecorps.S;
}
```

Dans ExtProcess, c'est, avec deux arguments :

```
        /**
        * returns exergy reference value and initializes T0Exer
        */
  public double getExergyReference(Corps corps,String nomCorps){
    String[] args=new String[2];
    args[0]="project";//type of the element (see method getProperties(String[] args))
    args[1]="";//name of the process (see method getProperties(String[] args))
    Vector vProp=proj.getProperties(args);
    Double f=(Double)vProp.elementAt(2);
    T0Exer=f.doubleValue();
    corps.CalcPropCorps(T0Exer,1.0,0);
    getSubstProperties(nomCorps);
    return Hsubst-T0Exer*Ssubst;
  }
```

Les implémentations par défaut pour les composants externes sont données ci-dessous :

La classe ExtProcess comporte une implémentation par défaut de getExergyType() et getExergyBalance (String[] args), qui doivent être sous-classées dans les composants externes. Les propriétés exergétiques des fluides passent par getSubstProperties().

```
double tauPlus,deltaXhPlus,xqPlus,tauProduct,deltaXhProduct,etaExer,deltaXhi;

        void setExergyExtensor(boolean isExtensor){
          if(isExtensor)exergyType="extensor";
          else exergyType="reductor";
        }
  public String[] getExergyType(){//modRG exerg
    String[] type=new String[9];
    type[0]=exergyType;//extensor ou reductor
    type[1]="false";//affiche JCheckExtSource
    type[2]="External source";//label JCheckExtSource
    type[3]="false";//affiche JCheckIntExchange
    type[4]="Internal exchange";//label JCheckIntExchange
    type[5]="false";//affiche JCheckValuableExergy
    type[6]="Valuable exergy";//label JCheckValuableExergy
    type[7]="false";//affiche champ d'entrée de valeur
    type[8]="Source T (°C)";//label sourceT_value
    return type;
  }

  public double[] getExergyBalance(String[]args){//modRG exerg
```

```
    double[] exergyBalance=new double[9];
    exergyBalance[0]=tauPlus;
    exergyBalance[1]=deltaXhPlus;
    exergyBalance[2]=xqPlus;
    exergyBalance[3]=tauProduct;
    exergyBalance[4]=deltaXhProduct;
    exergyBalance[5]=etaExer;
    exergyBalance[6]=deltaXhi;
    exergyBalance[7]=tauPlusFactor;
    exergyBalance[8]=deltaXhPlusFactor;
    return exergyBalance;
  }
```

3.2.4.1 Classe SolarConcentratorCC

Un capteur solaire convertit le flux solaire reçu en chaleur transmise au fluide qui le parcourt. Il s'agit donc d'un extenseur d'exergie, et, dans le constructeur de la classe SolarConcentratorCC, le type d'UPD est fixé par : setExergyExtensor(true);

Les calculs exergétiques peuvent être effectués de la manière suivante :

```
double Xh0=getExergyReference(refrig);
tauPlus=0;
deltaXhPlus=0;
tauProduct=0;
xqPlus=P/1000*Sc*(1-T0Exer/5800);//le soleil est une source à 5800 K
deltaXhProduct=(Haval-Hamont-T0Exer*(Saval-Samont))*flow;
deltaXhi=xqPlus-deltaXhProduct;
etaExer=deltaXhProduct/xqPlus;
```

3.2.4.2 Classe FluidEjector

Un éjecteur est modélisé comme un mélangeur externe. Selon que l'on s'intéresse au flux moteur ou au flux entraîné, il s'agit d'un réducteur ou d'un extenseur d'exergie. Toutefois, on peut aussi plus simplement le considérer comme un simple mélangeur, ce qui évite de lui associer une jonction ou un embranchement.

Pour s'assurer que l'exergie entrante n'est pas considérée comme un apport externe au cycle, les deux lignes suivantes sont insérées dans le constructeur :

```
deltaXhPlusFactor=0;
tauPlusFactor=0;
```

Les calculs exergétiques peuvent être effectués de la manière suivante :

```
double Xhmi=Hmi-T0Exer*Smi-Xh0-getExergyReference(refrig,nomCorps);
double Xhmsi=Hsi-T0Exer*Ssi-Xh0-getExergyReference(refrig,nomCorps);
double Xhsr=Hd_is-T0Exer*Smix-Xh0-getExergyReference(refrig,nomCorps);
deltaXhProduct=msr*Xhsr;
deltaXhPlus=msi*Xhmsi+mi*Xhmi;
etaExer=deltaXhProduct/deltaXhPlus;
deltaXhi=deltaXhPlus-deltaXhProduct;
```

3.2.4.3 Classe SOFCH2outlet

Une pile à combustible convertit de l'hydrogène en électricité. Elle se comporte donc comme un quadripôle recevant deux fluides en entrée, et dont en sortent deux autres. Le quadripôle est formé en associant un mélangeur en entrée et un diviseur en sortie, les deux étant reliés par une transfo-point qui joue un rôle purement passif. Les calculs sont effectués par le diviseur de sortie.

Les calculs exergétiques peuvent être effectués de la manière suivante :

```
double DH0=-241830;//kJ/kmol H2 vapeur
```

```
double DH0_vap=-285830;//kJ/kmol H2 liquide
double DG0=-237160;//kJ/kmol H2
double elecPower=tau*DG0*epsi*molFlowH2;
double Qlib=-tau*DH0*molFlowH2+elecPower;
tauProduct=-elecPower;
etaExer=elecPower/DH0/molFlowH2;
deltaXhi=(1-etaExer)*tauProduct;
```

3.2.5 Gestion des types d'énergie

Il peut tout d'abord être nécessaire de gérer l'affectation des types d'énergie de manière plus complexe que dans les transfos du noyau de Thermoptim, qui sont essentiellement mono-fonctionnelles. Pour elles, les énergies mises en jeu sont soit payante, soit utile, soit autre. Dans une transfo externe, il peut en être autrement, avec par exemple une charge thermique en énergie payante et une puissance mise en jeu en énergie utile.

La méthode updateProcess() de ComposantExt permet d'affecter les valeurs que l'on désire aux différents types d'énergie. Elle est facilement mise en œuvre avec la méthode setEnergyTypes d'ExtThopt :

```
tfe.updateProcess(setEnergyTypes(tfe.getCompName(),useful,purchased,other));

public Vector setEnergyTypes(String process, double useful,double purchased, double other){
        Vector vEner=new Vector();
        vEner.addElement(process);//process name
        vEner.addElement(new Double(useful));//useful energy
        vEner.addElement(new Double(purchased));//purchased energy
        vEner.addElement(new Double(other));//other energy
        return vEner;
}
```

3.2.6 Accès aux autres éléments du simulateur

Accès aux transfos amont et aval

Les noms des transfos amont et aval sont accessibles par les éléments 9 et 10 du Vector des propriétés de la transfo externe.

```
String[] args=new String[2];
args[0]="process";//type of the element (see method getProperties(String[] args))
args[1]=tfe.getCompName();//name of the process (see method getProperties(String[] args))
Vector vProp=proj.getProperties(args);
String transfoAmont=(String)vProp.elementAt(9);//nom de la transfo amont (ou "null" s'il n'y en a pas)
String transfoAval=(String)vProp.elementAt(10);//nom de la transfo aval (ou "null" s'il n'y en a pas)
```

Une fois ce nom obtenu, on accède à ses propriétés en le passant en args[1] dans proj.getProperties(args). On peut ainsi récursivement parcourir les transfos amont et aval directement connectées à une transfo.

C'est ce mécanisme qui est utilisé pour effectuer des mises à jour des transfos amont et aval des nœuds externes dans la méthode public void updateStraightlyConnectedProcess(String startProcess, String name, boolean downstream, boolean inletPoint, boolean outletPoint, boolean updateT, double T, boolean updateP, double P, boolean updateX, double x) de ExtNode.

L'exemple traité dans la note intitulée "CycleLiBrH2O.doc" et partiellement repris comme illustration des nœuds externes montre comment utiliser ces mécanismes.

Signalons en passant que vProp=proj.getProperties(args) donne aussi accès à certaines propriétés globales du projet si args[0]="project" (cf. annexe 2), et notamment à l'unité de débit. Il y a là un moyen de vérifier que les unités de débit et de puissance implicitement ou explicitement employées la classe externe sont compatibles avec celles du projet, et de prévoir un message si ce n'est pas le cas.

Thermocoupleurs

Etant donné que les thermocoupleurs sont des sortes d'échangeurs de chaleur, il est intéressant de les caractériser par des valeurs telles que l'efficacité ε, le UA, le NUT ou la DTML, que l'on peut calculer à partir d'équations analogues.

Pour cela, il faut que le composant externe transmette à chacun de ses thermocoupleurs les valeurs des équivalents débits, température d'entrée et de sortie et énergie thermique transférée qu'ils doivent prendre en compte dans leurs calculs. Des méthodes spécifiques ont été placées dans l'interface à cet effet.

Il faut cependant être conscient que l'analogie avec les échangeurs comporte quelques limites : par exemple, des croisements de température inadmissibles dans un échangeur peuvent se présenter dans un thermocoupleur, conduisant à une absurdité au niveau des calculs si l'on n'y prend pas garde.

On aura donc souvent intérêt à transmettre des valeurs qui ne risquent pas de conduire à ce genre de situation, une solution pouvant être de considérer, pour les calculs des caractéristiques analogues à celles d'un échangeur, que le thermocoupleur est isotherme, conformément au modèle retenu pour l'absorbeur et le désorbeur de l'exemple présenté dans la note "TrigenMicroTAG.doc".

Pour pouvoir accepter des couplages multiples, tous les heatConnectable définissent un tableau des types de thermocouplers acceptables (dans l'exemple ci-dessus : String[]thermoCouplerTypes={"absorber","desorber"}). Les appels se font ensuite en utilisant comme identifiant le type de thermocoupleur, comme dans :
public double getInletTemperature(String thermoCouplerType);

Lorsque ce tableau est de dimension 1 et que le heatConnectable est une transfo, la gestion peut être simplifiée (voir plus loin). Sinon, il faut que le heatConnectable puisse distinguer quel thermocoupler l'appelle, afin de savoir quoi lui renvoyer par les méthodes d'interface. Il doit donc s'enregistrer chez lui lors de la création, ce qui suppose une vérification de cohérence lors de la construction initiale, et un enregistrement mutuel pour la reconstruction ultérieure.

Pour tout composant externe existent de surcroît un certain nombre de difficultés supplémentaires, étant donné que son accès aux grandeurs internes est limité (les cas mentionnés ici sont relatifs aux transfos, mais les problèmes se posent de manière analogue pour les noeuds) :

- la dimension du tableau des types de thermocouplers acceptables et de ceux qui lui sont associés n'est pas connue a priori, ce qui conduit à l'initialiser de manière spécifique dans la méthode setCompFrame(Object obj) de ComposantExt, qui est la classe-pivot où sont réalisés en pratique une bonne partie des échanges entre la partie interne du code et les composants externes.
- les mises à jour des différents thermocoupleurs se font par la méthode non camouflée de TransfoExterne updateThermoCouplers(Vector vTC)
- les échanges d'informations entre le heatConnectable et ses thermocouplers passent par des Vector, et il doit pouvoir distinguer les différentes valeurs en fonction du rôle de chacun d'entre eux.

La méthode updateThermoCoupler() d'ExtProcess permet d'effectuer les mises à jour :

updateThermoCoupler(String type, double Tin, double Tout, double Q, double flow)

Elle transmet au thermocoupleur les éléments de mise à jour par l'intermédiaire d'un Vector construit par la méthode d'ExtProcess :

```
protected Vector getThermoCouplerVector(String type, double Tin, double Tout, double Q, double flow){
   Vector vTC=new Vector();
   vTC.addElement(type);
   Double d=new Double(Tin);
   vTC.addElement(d);
   d=new Double(Tout);
   vTC.addElement(d);
   d=new Double(Q);
   vTC.addElement(d);
   d=new Double(flow);
```

```
    vTC.addElement(d);
    return vTC;
}
```

Quoique cela ne soit pas recommandé, on peut éventuellement se passer d'utiliser la méthode updateThermoCoupler() pour une transfo externe simple, avec un seul thermocoupleur et pour laquelle les valeurs des températures d'entrée et de sortie, de la charge thermique et du débit peuvent directement être obtenues à partir de celles de la transfo. En revanche, s'il y a plusieurs thermocoupleurs connectés à la même transfo, chacun d'entre eux doit être mis à jour à chaque recalcul.

On notera que l'initialisation d'un thermocoupleur ne peut être faite que par le composant externe dont il dépend. Lors du chargement d'un projet, l'état des thermocoupleurs n'est mis à jour que lorsque les composants externes sont calculés. Pour éviter des difficultés lors du premier recalcul automatique d'un projet, il est recommandé d'effectuer un calcul des composants externes munis de thermocoupleurs en utilisant la méthode init() prévue à cet effet (cf. section 3.2.1.1).

Nœuds

Pour identifier les nœuds auxquels une transfo peut être connectée, il faut parcourir les nœuds du projet, acessibles par la méthode proj.getNodeList(), récupérer la structure de chacun d'entre eux par un appel à proj.getProperties() avec les bons arguments, et regarder si le nom de la transfo apparaît dans les noms de branches ou de la transfo principale du nœud. C'est un peu lourd, mais, même de l'intérieur de Thermoptim, il faut opérer ainsi.

Accès à l'éditeur de schémas

Il est aussi possible d'accéder au contenu de l'éditeur de schémas par les méthodes suivantes de Projet, assez difficiles à employer en pratique, notamment la seconde :

getEditorComponentList() : fournit la liste des composants présents dans l'éditeur de schémas, sous la forme name="nomComposant"+tab+type="typeComposant". Cette liste peut être décomposée simplement avec Util.extr_value("name") et Util.extr_value("type").

getConnectionCode(String[] args) : renvoie un code permettant de savoir si deux composants de l'éditeur sont connectés ou non : 1 si le composant 1 est en amont du composant 2, -1 dans le cas contraire, et 0 s'ils ne sont pas connectés. La structure des arguments est la suivante :

```
String[] args=new String[4];
args[0]="Expansion";//type du composant 1
args[1]="turbine";//nom du composant 1
args[2]="Exchange";//type du composant 2
args[3]="régen gaz";//nom du composant 2
```

Les codes des types sont ceux qui sont utilisés dans les fichiers de sauvegarde de projet.

3.2.7 Sauvegarde et chargement des paramètres du modèle

Il est possible de sauvegarder dans les fichiers de projet habituels de Thermoptim (et ensuite de relire) les paramétrages des composants externes en utilisant deux méthodes spécifiques :

```
public String saveCompParameters()
public void readCompParameters(String ligne_data)
```

La seule contrainte est que l'ensemble du paramétrage du composant externe doit tenir sur une ligne, avec un formatage compatible avec celui qui est utilisé dans le noyau du progiciel : les différents champs de sauvegarde sont séparés par des tabulations.

ExtThopt.Util fournit une méthode générique pour pouvoir associer à la valeur d'un paramètre un code permettant de le repérer :

```
public static String extr_value(String ligne_data, String search)
```

La sauvegarde est effectuée sous la forme "paramètre= valeur", et la recherche se fait sous la forme :

valeur=Util.lit_d(Util.extr_value(ligne_data, "paramètre"));

Si le paramétrage du composant est trop complexe pour être sauvegardé de la sorte, rien n'empêche un utilisateur d'utiliser ce mécanisme pour sauvegarder le nom d'un fichier spécifique de paramétrage, et ensuite de relire comme il le désire ce fichier.

3.3 Nœuds externes

Nous vous recommandons de vous référer à l'API de extThopt.MixerExterne, extThopt.DividerExterne, extThopt.ExtMixer, extThopt.ExtDivider et extThopt.ExtNode pour connaître l'ensemble des méthodes disponibles, leur syntaxe et leur fonction. Cette API fait partie de l'environnement de développement des classes externes mis à votre disposition (dans le dossier api_extThopt).

3.3.1 Construction

La construction d'un nœud externe est en tout point semblable à celle d'une transfo externe.

3.3.2 Mise à jour et calcul du nœud

On se réfère ici à l'exemple de la classe Absorber, dont le modèle est présenté dans la note spécifique intitulée la note intitulée "CycleLiBrH2O.doc" présentant la modélisation interne d'un cycle à absorption.

L'enchaînement des opérations est le suivant :

1) vérification de la cohérence et mise à jour du nœud avant calcul
2) lecture des paramètres sur l'écran du composant externe
3) mise à jour des transfos connectées au nœud externe
4) mise à jour et calcul des thermocoupleurs associés
5) mise à jour de l'écran du composant externe
6) sauvegarde et chargement des paramètres du modèle

Examinons maintenant les problèmes rencontrés en pratique à chacune de ces étapes. Quelques extraits du code sont donnés ci-dessous, mais il est recommandé de lire la suite de cette note en ayant sous les yeux l'intégralité de la classe Absorber.java.

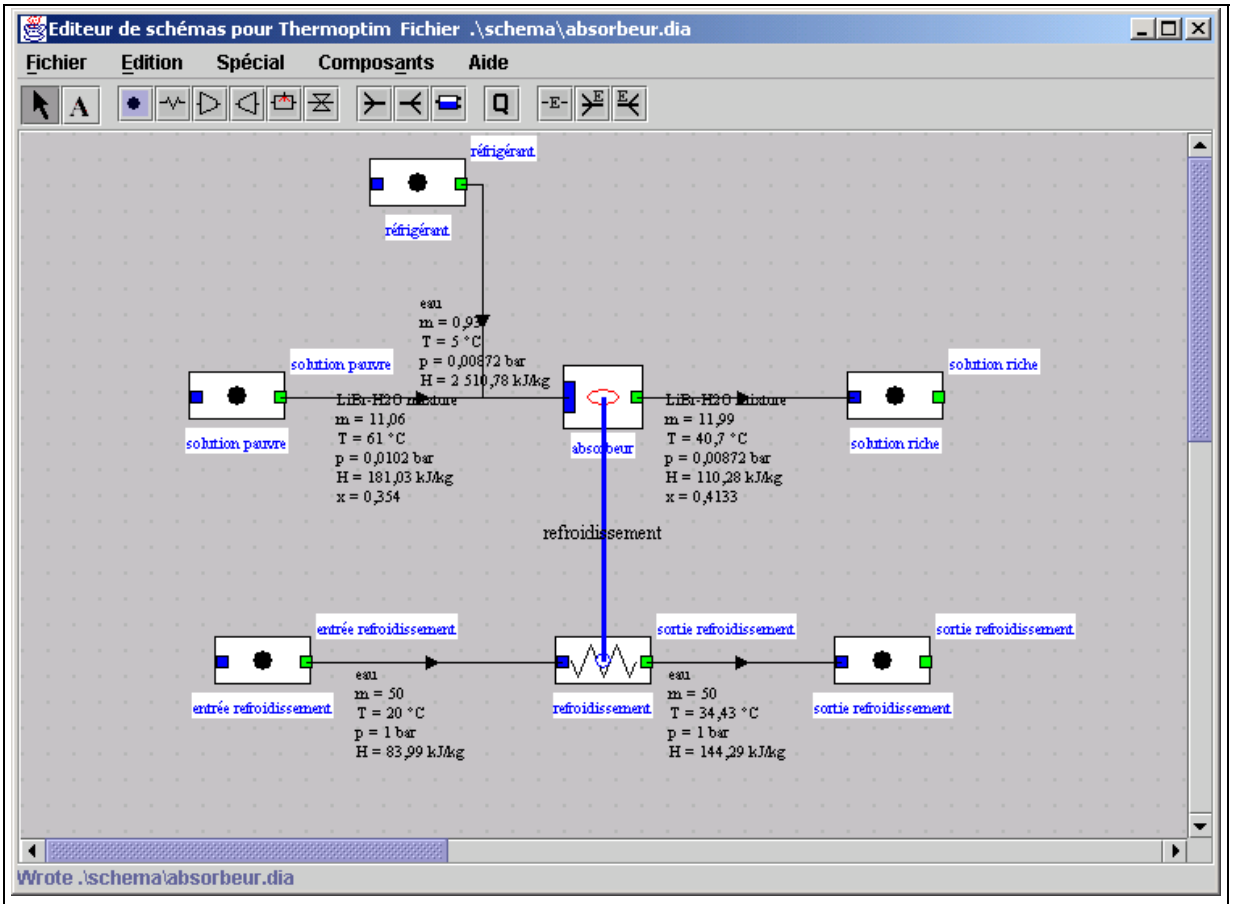
1) vérification de la cohérence et mise à jour du nœud avant calcul

Plusieurs difficultés existent ici :

- en premier lieu, il est fondamental de s'assurer que le nœud a été bien construit, c'est-à-dire que les branches et la transfo principale sont bien conformes à ce que le concepteur du modèle attend. En effet, aucun test de cohérence n'étant possible à ce niveau dans l'éditeur de schémas, un utilisateur peut très bien se tromper lors de la description du schéma ;
- ensuite, un composant externe n'a pas un accès direct aux variables du simulateur : ces grandeurs sont obtenues par des méthodes particulières génériques, qui construisent des Vector de structure différente selon l'objet désiré.

La première étape consiste donc à **vérifier la cohérence de la construction du nœud**. Pour cela, le concepteur dispose d'une méthode d'ExtNode, getNodeStructure(), qui décode la structure du nœud, en chargeant dans un String appelé mainProcess le nom de la transfo principale, et dans un tableau de String appelé nomBranches[] les noms des transfos des différentes branches, dont le nombre correspond à la dimension du tableau.

Le concepteur doit programmer une méthode (appelée checkConsistency() dans notre exemple) dans laquelle il parcourt les diverses transfos du nœud en effectuant tous les tests qu'il désire, et en en profitant pour effectuer les mises à jour des paramètres d'entrée de son modèle.



Mélangeur externe représentant un absorbeur, avec ses connexions

Par exemple, pour l'absorbeur, la méthode est la suivante (pour faciliter le traitement ultérieur, on a décidé de charger les noms des transfos dans des String faciles à identifier appelés richSolutionProcess, refrigerantProcess et poorSolutionProcess, et ceux des points dans richSolutionPoint, refrigerantPoint et poorSolutionPoint) :

```
private void checkConsistency(){
      String[] args=new String[2];
      Vector[] vBranch=new Vector[2];
      isBuilt=true;
      poorSolutionProcess="";
      refrigerantProcess="";
      if(nBranches!=2){//premier test pour vérifier le nombre de branches
                String msg = "Error on the number of branches which is "+nBranches+" instead of 2";
                JOptionPane.showMessageDialog(me, msg);
                isBuilt=false;
      }
      else{
        for(int j=0;j<nBranches;j++){
          args[0]="process";//type of the element (see method getProperties(String[] args))
          args[1]=nomBranche[j];//name of the process (see method getProperties(String[] args))
          vBranch[j]=proj.getProperties(args);//chargement de la transfo correspondante
           String aval=(String)vBranch[j].elementAt(2);//gets the downstream point name
          getPointProperties(aval);//direct parsing of point property vector
            String nom=nomCorps;
            //Check the substance at inlet //vérification du nom du corps
            System.out.println(" ligne "+j+" nomCorps "+nomCorps);
                  if(nom.equals("LiBr-H2O mixture")){//s'il s'agit du mélange, on a détecté la solution pauvre
                     poorSolutionProcess=nomBranche[j];
                     poorSolutionPoint=aval;
```

```
                    //on initialise alors les variables d'entrée
                      Tsp=Tpoint;
                      Psp=Ppoint;
                      Hsp=Hpoint;
                      Xsp=Xpoint;
                      Double f=(Double)vBranch[j].elementAt(3);
                      msp=f.doubleValue();
                    }
                    if(nom.equals("eau")){// s'il s'agit de l'eau, on a détecté le réfrigérant
                      refrigerantProcess=nomBranche[j];
                      refrigerantPoint=aval;
                     //on initialise alors les variables d'entrée
                      Prefr=Ppoint;
                      Hrefr=Hpoint;
                      Trefr=Tpoint;
                      Double f=(Double)vBranch[j].elementAt(3);
                      mr=f.doubleValue();
                    }
                  }
                  if((refrigerantProcess.equals(""))||(poorSolutionProcess.equals(""))){//sinon il y une erreur
                    String msg = "Error on at least one of the branch substances";
                    JOptionPane.showMessageDialog(me, msg);
                    isBuilt=false;
                  }
        }
        richSolutionProcess=mainProcess;
              args[0]="process";//type of the element (see method getProperties(String[] args))
              args[1]=richSolutionProcess;//name of the process (see method getProperties(String[] args))
              Vector vPropMain=proj.getProperties(args);
              Double f=(Double)vPropMain.elementAt(3);
              msr=f.doubleValue();
              String amont=(String)vPropMain.elementAt(1);//gets the upstream point name
              getPointProperties(amont);//direct parsing of point property vector
              richSolutionPoint=amont;
              Tsr=Tpoint;
              Psr=Ppoint;
              Hsr=Hpoint;
              Xsr=Xpoint;
        String nom=nomCorps;

        //Check the substance at inlet
              if(!(nom.equals("LiBr-H2O mixture"))){//on vérifie aussi que la transfo principale a le bon corps
                String msg = "Error on main process substance,which is "+nomCorps+" instead of LiBr-H2O
                     mixture";
                JOptionPane.showMessageDialog(me, msg);
                isBuilt=false;
              }
}
```

2) lecture des paramètres sur l'écran du composant externe

Le package extThopt fournit un certain nombre de méthodes simples mais robustes pour convertir en double les String affichés dans les champs JTextField utilisés dans l'interface graphique, et réciproquement pour afficher les double dans ces champs. Ils sont implémentés sous forme de méthodes statiques de la classe extThopt.Util :

```
Tabs=Util.lit_d(Tabs_value.getText())+273.15;
```

3) mise à jour des transfos connectées au nœud externe

La mise à jour des transfos associées au nœud est faite dans Thermoptim par les méthodes updateMixer(Vector vTC) et updateDivider(Vector vTC), qui, pour pouvoir s'adapter à tous les cas possibles, actualisent d'une part le nœud, d'autre part les débits des transfos, et enfin les points amont des transfos sortantes et les points aval des transfos entrantes.

Le concepteur du nœud externe doit donc configurer convenablement le Vector qui passe en argument. Il dispose pour cela de deux méthodes génériques d'ExtMixer et ExtDivider :

- updateMixer(Vector[]vTransfo,Vector[]vPoints, double TGlobal, double hGlobal)
- updateDivider(Vector[]vTransfo,Vector[]vPoints, double TGlobal, double hGlobal)

auxquelles il doit fournir des tableaux de Vector pour les transfos et les points de chaque branche et de la veine principale, ainsi que les valeurs globales de la température et de l'enthalpie.

Pour préparer les Vector, ExtNode dispose d'une méthode générique :

void setupVector(String process, String point, int i, double m, double T, double P, double X)

i est l'indice des deux Vector, process et point sont les noms de la transfo et du point, m le débit de la transfo, T la température, P la pression et X le titre du point de la transfo le plus proche du nœud.

Dans notre exemple, la solution retenue est d'utiliser trois méthodes intermédiaires pour préparer les Vector :
- setupRichSolution(double m, double T, double P, double X)
- setupPoorSolution(double m, double T, double P, double X)
- setupRefrigerant(double m, double T, double P, double X)

Il suffit alors de dimensionner les tableaux de Vector définis dans ExtMixer, et d'exécuter updateMixer() :

```
vTransfo= new Vector[nBranches+1];
vPoints= new Vector[nBranches+1];
setupRichSolution(msr,Tabs,Psr,Xsr);
setupPoorSolution(msp,Tsp,Psp,Xsp);
setupRefrigerant(mr,Trefr,Prefr,1);
updateMixer(vTransfo,vPoints,Tsr,Hsr);
```

Pour les cas courants, une fois que les débits et les points amont ou aval des transfos reliées au nœud ont été mis à jour, Thermoptim prend en charge la propagation du recalcul des différents éléments grâce au moteur de recalcul automatique. Il est cependant possible que cela ne suffise pas et que le concepteur du nœud doive assurer lui-même la propagation des mises à jour. C'est notamment le cas dans l'exemple qui nous intéresse pour la propagation de la valeur de la concentration en réfrigérant du mélange LiBr-H_2O. En effet, cette valeur est stockée dans ce qui est normalement le titre en vapeur du point, qui n'est jamais propagé automatiquement.

Dans ce cas, le concepteur doit chercher tous les points en aval ou en amont de celui dont la composition a été modifiée, pour s'assurer qu'ils sont correctement mis à jour eux aussi. Il dispose pour cela de la méthode updateStraightlyConnectedProcess() d'ExtNode, qui parcourt récursivement les transfos connectées en amont ou en aval du nœud :

```
updateStraightlyConnectedProcess(richSolutionProcess, richSolutionProcess,
            false,//boolean downstream,
    true,//boolean inletPoint,
    true,//boolean outletPoint,
    false,//boolean updateT,
    0,//double T,
    false,//boolean updateP,
    0,//double P,
    true,//boolean updateX,
    Xsr);
```

4) mise à jour et calcul des thermocoupleurs associés

On se réfèrera à la section 3.2.3 relative aux transfos externes pour les explications générales sur la mise à jour des thermocoupleurs.

L'objectif est que le composant externe transmette à chacun de ses thermocoupleurs les valeurs d'équivalents débits, température d'entrée et de sortie et énergie thermique transférée qu'ils doivent prendre en compte dans leurs calculs.

La méthode updateThermoCoupler() d'ExtMixer ou ExtDivider permet d'effectuer cette mise à jour :

updateThermoCoupler("absorber", Tabs, Tabs, Qabs, msr);

Par exemple, on a ici supposé que l'absorbeur est à la température T_{abs}, qu'il reçoit la charge thermique Q_{abs} et qu'il est traversé par le débit m_{sr}. Si l'on n'avait pas pris la température de l'absorbeur comme référence pour les calculs d'échange, en conservant celles de la vapeur d'eau entrant et sortant de la transfo externe, on aurait abouti à un croisement de température inadmissible conduisant à un diagnostic d'erreur par Thermoptim.

A la différence des transfos externes, la mise à jour doit toujours être effectuée avec la méthode updateThermoCoupler(), même pour les nœuds externes ne mettant en jeu qu'un seul thermocoupleur.

5) mise à jour de l'écran du composant externe

On se réfèrera à la section sur les transfos externes.

3.3.3 Gestion des types d'énergie

Tout comme pour les transfos externes, il peut être nécessaire de gérer l'affectation des types d'énergie d'un nœud externe, alors que ce problème ne se pose pas pour les nœuds du noyau de Thermoptim.

Pour éviter d'alourdir le mécanisme existant, la solution qui a été retenue est d'affecter artificiellement les valeurs que l'on désire aux différents types d'énergie de la veine principale du nœud externe (cf. section 3.2.3).

La méthode updateProcess() de ComposantExt permet de le faire, à condition que la veine principale soit une transfo-point, faute de quoi les valeurs des affectations des énergies qu'elle met en jeu seront réinitialisées à chaque recalcul.

Par exemple, la classe Desorber pourrait déclarer comme énergie payante la charge de son thermocoupleur par :

de.updateProcess(setEnergyTypes(richSolutionProcess,0,Qgen,0));

3.3.4 Sauvegarde et chargement des paramètres du modèle

On se référera à la section sur les transfos externes.

3.3.5 Accès aux calculs de combustion à partir des classes externes

Afin que les classes externes puissent accéder aux calculs de combustion disponibles dans Thermoptim, diverses modifications ont été apportées au progiciel. En particulier, une classe appelée ExternalCombustion, qui hérite directement de Combustion, a été ajoutée. Accessible à partir de MixerExterne, elle permet d'effectuer des calculs de combustion de la manière suivante :

- on commence par construire un Vector vSettings, qui comprend les trois gaz idéaux mis en jeu (comburant, combustible, gaz brûlés), ainsi que l'ensemble des paramétrages nécessaires pour spécifier les calculs à effectuer (la structure de ce Vector est donnée en annexe) ;
- la méthode public void calcExternalCombustion(Vector vSettings) de MixerExterne initialise la combustion puis effectue les calculs demandés
- les résultats sont ensuite récupérés par la méthode public Vector getExternalCombustionResults() de MixerExterne, qui renvoie un Vector comprenant la composition des gaz brûlés, la température de fin de combustion, lambda, les énergies mises en jeu (la structure de ce Vector est donnée en annexe 2) …

Depuis les classes externes (dans des mélangeurs externes uniquement, une chambre de combustion étant structurellement analogue à un mélangeur), l'accès se fait grâce aux méthodes de même nom d'ExtMixer, par des appels du type :

```
calcExternalCombustion(vSettings);
Vector results=getExternalCombustionResults();
```

Le paramétrage de la combustion est en tout point le même que celui qui est requis pour les combustions réalisées dans le noyau de Thermoptim, à la réserve près qu'il est possible d'imposer une charge thermique à prendre en compte pour le calcul de la température de fin de combustion.

On pourra se référer à l'exemple de la classe BiomassCombustion qui illustre bien comment faire appel à ces fonctionnalités.

3.3.6 Accès aux calculs sur les gaz humides à partir des classes externes

On appelle mélange humide, et par abus de langage **gaz humide**, un mélange de gaz qui ne se condensent pas, que nous appellerons le **gaz sec**, et d'**eau**, susceptible de se condenser.

Depuis l'origine, Thermoptim dispose de fonctions de calcul des propriétés humides des gaz et des points, mais celles-ci ne sont généralement utilisées que pour effectuer des calculs particuliers, découplés des calculs de cycles thermodynamiques standard, comme par exemple pour du traitement de l'air ou de la climatisation (cf. manuel de référence tome 2). C'est d'ailleurs pour cela que les transfos humides ne disposent pas de pictogramme dans l'éditeur de schémas.

Nous supposerons dans ce qui suit que le lecteur est suffisamment familier à la fois de l'utilisation des classes externes et de la thermodynamique des gaz humides. Les définitions, notations et équations sont celles du tome 1 du livre Systèmes Energétiques.

Par commodité de langage nous utiliserons le qualificatif de **standard** lorsque nous ferons référence à l'environnement de Thermoptim hors calculs humides.

L'environnement de calcul des cycles standard de Thermoptim et celui des calculs humides ne sont pas directement compatibles pour les deux raisons suivantes :

- suivant l'usage en matière de calculs humides, les valeurs des fonctions thermodynamiques sont généralement ramenées au gaz sec, dont la composition est invariante, alors que les calculs standard effectués dans Thermoptim le sont par rapport à la composition réelle des gaz. Les valeurs auxquelles elles conduisent sont ainsi appelées spécifiques, pour les distinguer des autres ;
- de plus, les températures et pressions de référence n'étant pas les mêmes dans les deux environnements, il est nécessaire de faire des conversions lorsque l'on passe de l'un à l'autre.

Pourtant, un certain nombre de cycles thermodynamiques mettent en jeu des variations de l'humidité des gaz, et il était regrettable de ne pouvoir les modéliser facilement dans Thermoptim. C'est pour cela que les fonctions de calcul des propriétés humides des gaz et des points de Thermoptim ont été rendues accessibles depuis les classes externes. L'annexe 5 explique comment les utiliser.

3.4 Accès aux instances des transfos et nœuds externes

Les sections précédentes et l'annexe 1 expliquent comment accéder aux différentes instances non protégées de Thermoptim. Les procédures (relativement lourdes) sont limitées compte tenu des contraintes de cohérence interne du progiciel.

En revanche, un développeur de composants externes peut souhaiter accéder comme il l'entend aux différentes classes qu'il crée, ceci afin de synchroniser les calculs entre elles. Pour cela, il dispose de la méthode de Projet getExternalClassInstances(), qui construit un Vector donné en annexe 2, contenant les divers transfos et nœuds externes instanciés, avec une description permettant de les identifier.

A titre d'exemple, le nœud externe de veine principale "maTransfo" et de type "monType" peut être obtenu par le code suivant :

```
MaClasse monInstance ;
Vector vExt=proj.getExternalClassInstances();
      for(int i=0;i<vExt.size();i++){
         Object[] obj=new Object[6];
                obj=(Object[])vExt.elementAt(i);
                String type=(String)obj[0];
                if(type.equals("node")){
                   String main=(String)obj[4];
                   String nodeType=(String)obj[3];
                   if((main.equals(maTransfo))&&((nodeType.equals("monType")))){
                      monInstance=(MaClasse)obj[1];
                     monInstanceName=(String)obj[2];
                   }
                }
      }
```

Une fois l'instance retrouvée, l'accès à ses membres se fait selon les règles du langage Java, en fonction de leurs modificateurs d'accès (public, protected…).

3.5 Pilotage externe de Thermoptim

Piloter Thermoptim de l'extérieur a trois principales applications :
- d'une part faciliter la mise au point des classes externes en les testant au fur et à mesure qu'elles sont définies ;
- d'autre part permettre d'émuler Thermoptim à partir d'autres applications, en le faisant par exemple fonctionner en mode client-serveur ;
- enfin, donner accès à l'ensemble des bibliothèques non protégées pour différents usages (effectuer des traitements externes entre deux recalculs, guider un utilisateur dans un module de formation…).

Dans ce tome, nous nous intéresserons essentiellement aux deux premières applications, le tome 4 étant plus particulièrement dédié au dimensionnement technologique et à l'étude du régime non-nominal, pour laquelle les pilotes se révèlent des outils particulièrement intéressants, car ils permettent de coordonner les recalculs entre les différents composants de Thermoptim, et notamment de résoudre les équations de couplage qui apparaissent pour ce type d'étude.

Pour rendre possible le pilotage externe, la classe principale du simulateur, Projet, peut être sous-classée, et un certain nombre de méthodes non protégées peuvent être surchargées pour définir des traitements complémentaires de ceux du noyau. Le détail de ces méthodes est donné en annexe. Dans les exemples qui suivent, la classe ProjetThopt hérite de Projet, tandis que la classe Pilot est utilisée pour gérer le pilote.

Pour éviter toute confusion, ces deux classes ont été définies dans un paquet distinct de "extThopt", appelé "pilot".

3.5.1 Faciliter la mise au point des classes externes

Le principe consiste à instancier Thermoptim à partir de la classe externe Pilot, puis à le lancer en utilisant les méthodes appropriées de ProjetThopt. Les utilisateurs de Windows pourront utilement se référer à la section 6 intitulée "Environnement de développement des classes externes" qui explique comment utiliser le logiciel freeware Eclipse pour disposer d'un petit environnement de travail convivial.

Dans l'exemple donné, Pilot instancie Thermoptim par la séquence suivante, dont seule les deux premières méthodes sont nécessaires, les autres étant données pour illustrer certaines possibilités existantes:

```
runThopt();
loadProject();
listProcesses();
```

```
printProperties();
recalcProperties();
```

Les méthodes nécessaires sont :

```
public void runThopt(){
   proj=new ProjetThopt();//instanciation de Thermoptim
   proj.show();//affichage de l'écran du simulateur
   proj.openEditor();//affichage de l'éditeur de schémas
}

public void loadProject(){
   String[] args=new String[4];
   args[0]="complet.prj";//project file name
   args[1]="complet.dia";//diagram file name
   args[2]=".";//working directory
   args[3]="noTest";//pas de test de sauvegarde si un projet est déjà ouvert
   proj.loadSelection(args);//chargement de l'exemple défini dans args[]
}
```

Pour que les classes externes en cours de développement soient prises en compte dans Thermoptim, il faut les déclarer dans la méthode getInternalClasses() de la classe ProjetThopt. La figure ci-dessous vous indique comment opérer.

```
public Vector getInternalClasses(){
    Vector vClasses=new Vector();
    try{
        Class c=Class.forName("extThopt.LiBrAbsorption");//recompose le nom complet de
        vClasses.addElement(c);
        c=Class.forName("extThopt.Desorber");//recompose le nom complet de la classe "
        vClasses.addElement(c);
        c=Class.forName("extThopt.Absorber");//recompose le nom complet de la classe "
        vClasses.addElement(c);
        c=Class.forName("extThopt.LiBrH2OMixture");//recompose le nom complet de la cl
        vClasses.addElement(c);
    }
    catch(Exception e){//à compléter par un traitement détaillé des exceptions,
                       //pour informer le créateur de l'archive de ses erreurs de con
        e.printStackTrace();
    }
    return vClasses;
```

Méthode getInternalClasses() pour charger dans Thermoptim les classes testées

Rappelons que le package extThopt comprend une classe Util qui propose un certain nombre de méthodes utilitaires pour formater les nombres, les relire à l'écran, sauvegarder et relire des valeurs, inverser par dichotomie une méthode (cf. annexe 3)...

Une fois les classes définies, compilez-les, puis lancez le pilote en cliquant sur la touche F5. L'exemple défini dans la méthode loadProject() de la classe Pilot sera chargé quand vous cliquerez sur le bouton "Run Thermoptim and load example" de l'écran du pilote.

3.5.2 Donner accès aux bibliothèques non protégées

Il y a deux moyens de piloter Thermoptim : soit totalement à partir d'une application externe qui instancie le progiciel et le paramètre, soit partiellement, pour un projet donné.

Dans le premier cas, le principe consiste à instancier Thermoptim à partir de la classe externe Pilot, puis à accéder à ses bibliothèques en utilisant les méthodes appropriées de ProjetThopt. Les utilisateurs de Windows pourront utilement se référer à la section 6 intitulée "Environnement de développement des classes externes" qui explique comment utiliser un logiciel en freeware pour disposer d'un environnement de travail convivial.

Il est ainsi possible par exemple d'instancier un corps pour en calculer les propriétés thermodynamiques, ou bien d'afficher un diagramme thermodynamique et d'y tracer un cycle pré-enregistré.

Le pilote se présente ainsi comme une application indépendante, capable de lancer Thermoptim, qu'il considère comme une bibliothèque de classes externes, et de construire des projets en instanciant des classes Thermoptim. Connaissant ces instances, il peut à tout moment connaître leur état. Le pilote n'est pour le moment pas connu des instances de Thermoptim.

Le lancement du pilote ouvre une fenêtre avec un bouton de lancement de Thermoptim et de chargement d'un exemple. La classe correspondante (Pilot) possède quelques méthodes de base pour :

- lancer Thermoptim : runThopt()
- charger un exemple : loadProject()
- accéder aux propriétés des éléments du simulateur : proj.getProperties(args)
- simplifier le parsing des vecteurs de propriétés : getSubstProperties(String nom), getPointProperties(String nom), extr_value(String s), lit_b(String s)

Dans le second cas, on associe une classe de pilotage spécifique à un projet donné, et cette classe est instanciée lors du chargement du projet. Elle permet alors de coordonner les recalculs de ce projet selon des règles particulières. La classe de pilotage est une extension de extThopt.ExtPilot, qui dérive de rg.thopt.PilotFrame. Pour pouvoir l'associer au projet, il faut commencer par en faire une classe externe pour qu'elle soit chargée dans Thermoptim lors de son lancement. Une fois le projet ouvert, il faut sélectionner la classe de pilotage qui lui est associée par la ligne "Pilot frame" du menu Spécial. Si le projet est sauvé, le nom de la classe de pilotage est sauvegardé pour qu'elle puisse être instanciée lors du chargement du projet. Il est possible de sauvegarder les données du pilote au même titre que celles d'un composant externe, ce qui permet éventuellement d'enregistrer des résultats de simulation qui ne peuvent l'être dans les éléments habituels de Thermoptim.

Pilotage de Thermoptim

- calcThopt() : lance un recalcul dans Thermoptim à partir du tuteur
- setupThopt() : instanciation de l'éditeur de schémas et désactivation de certaines fonctions de Thermoptim
- openEditor() instanciation de l'éditeur de schémas
- setControls() : permet au tuteur de reprendre la main entre deux recalculs, pour modifier certains paramètres de Thermoptim par exemple
- loadSelection(String[] args) : chargement des exemples

Accès au simulateur

- getHxList () : liste des échangeurs du projet
- getNodeList () : liste des nœuds du projet
- getPointList () : liste des points du projet
- getProcessList() : liste des transfos du projet
- getSubstanceList () : liste des corps du projet
- getProperties(String[] args) : vecteur de propriétés. Sa structure, qui dépend du type d'élément considéré, est donnée en annexe.
- notifyElementCalculated(String[] element) : cette méthode est exécutée dans Thermoptim chaque fois qu'un élément (point, transfo, noeud, échangeur) est calculé. element est construit comme suit :

```
if(pt instanceof Transfo) element[0]="process";
else if(pt instanceof PointCorps) element[0]="point";
else if(pt instanceof Node) element[0]="node";
else if(pt instanceof HeatEx) element[0]="heatEx";
element[1]=pt.getName();
element[2]=pt.getType();
```

Ainsi, element peut directement être utilisé comme argument dans getProperties() : getProperties(element) fournit l'état de l'élément considéré.

Un autre point d'accès est la classe Corps, qui possède un certain nombre de méthodes publiques accessibles depuis le pilote, et notamment :

- CalcPropCorps (double T, double p, double x) : calcul de l'état complet d'un corps
- getSubstProperties() : vecteur des propriétés
- toute une série de méthodes permettant d'inverser les fonctions d'état.

Accès à l'éditeur de schémas

- getEditorComponentList() : fournit la liste des composants présents dans l'éditeur de schémas, sous la forme name="nomComposant"+tab+type="typeComposant". Cette liste peut être décomposée simplement avec extr_value("name") et extr_value("type").
- getConnectionCode(String[] args) : renvoie un code permettant de savoir si deux composants de l'éditeur sont connectés ou non : 1 si le composant 1 est en amont du composant 2, -1 dans le cas contraire, et 0 s'ils ne sont pas connectés. La structure des arguments est la suivante :

```
String[] args=new String[4];
args[0]="Expansion";//type du composant 1
args[1]="turbine";//nom du composant 1
args[2]="Exchange";//type du composant 2
args[3]="régen gaz";//nom du composant 2
```

3.5.3 Fonctionnement en mode client-serveur

Pour différentes raisons, il peut être intéressant d'utiliser Thermoptim à travers un réseau, à des fins didactiques, ou bien pour associer à un contrôleur (une régulation) un modèle thermodynamique détaillé, par exemple un modèle de pompe à chaleur couplé à un régulateur de chauffage.

La solution a été de développer une petite application basée sur la classe sti.ThoptEmulator, instanciée par sti.ThoptLauncher, qui permet de lancer un serveur géré par la classe sti.ThoptServer qui gère les échanges avec l'extérieur.

Le lancement du serveur se fait par EmulatorServerExec.jar, qui fait appel à EmulatorServer.zip, qui contient les bibliothèques de base. Ces deux fichiers doivent être placés dans le répertoire d'installation de Thermoptim.

L'échange des données se fait par des BufferedReader et PrintWriter, sous forme de String de plusieurs lignes, qui sont décodées par les classes client et serveur. Le client est une classe Java appelée Emulator.java, qui doit bien sûr être adaptée en fonction des besoins.

La solution proposée consiste à écrire un pilote externe, qui doit implémenter l'interface Emulable, qui définit les deux méthodes d'envoi String[] getValues() et de réception setParameters(String[] params) des données du client. Les String[] params de setParameters() et getValues() comprennent en première ligne le nombre de lignes suivantes comportant des enregistrements de valeurs. ThoptServer se contente de transmettre du client au serveur et réciproquement ces deux String[], sans les modifier.

Il est aussi possible que l'échange de requêtes se fasse en utilisant le protocole ModBus, qui, quoique plus complexe, présente l'avantage de permettre de communiquer avec des applications n'utilisant pas les codages Java qui sont souvent utilisées dans le monde du contrôle-commande.

3.6 Intégration de classes externes dans la bibliothèque extUser2.zip

Une fois que des classes externes sont mises au point, il faut les intégrer dans la bibliothèque extUser2.zip pour qu'elles puissent être automatiquement reconnues dans la version purement exécutable de Thermoptim.

3.6.1 Utilisation du gestionnaire de classes externes

Une fonctionnalité de Thermoptim permet d'effectuer cette opération : le **Gestionnaire de classes externes** est accessible depuis le menu Spécial du simulateur. Pour qu'il puisse fonctionner, il est nécessaire d'ajouter à l'installation quelques fichiers contenant les bibliothèques dont il a besoin[5].

Il faut que soient présents d'une part le fichier Extlib.ini, qui contient une seule ligne donnant le nom du chemin d'accès au répertoire (par défaut "externalClassLibrary2") où se trouve la bibliothèque (avec impérativement un sous-répertoire extThopt), et d'autre part l'archive externalClassLibrary2.zip dans laquelle sont temporairement stockées par Thermoptim les classes de la bibliothèque. Sinon, seul peut être affiché le contenu de extUser2.zip.

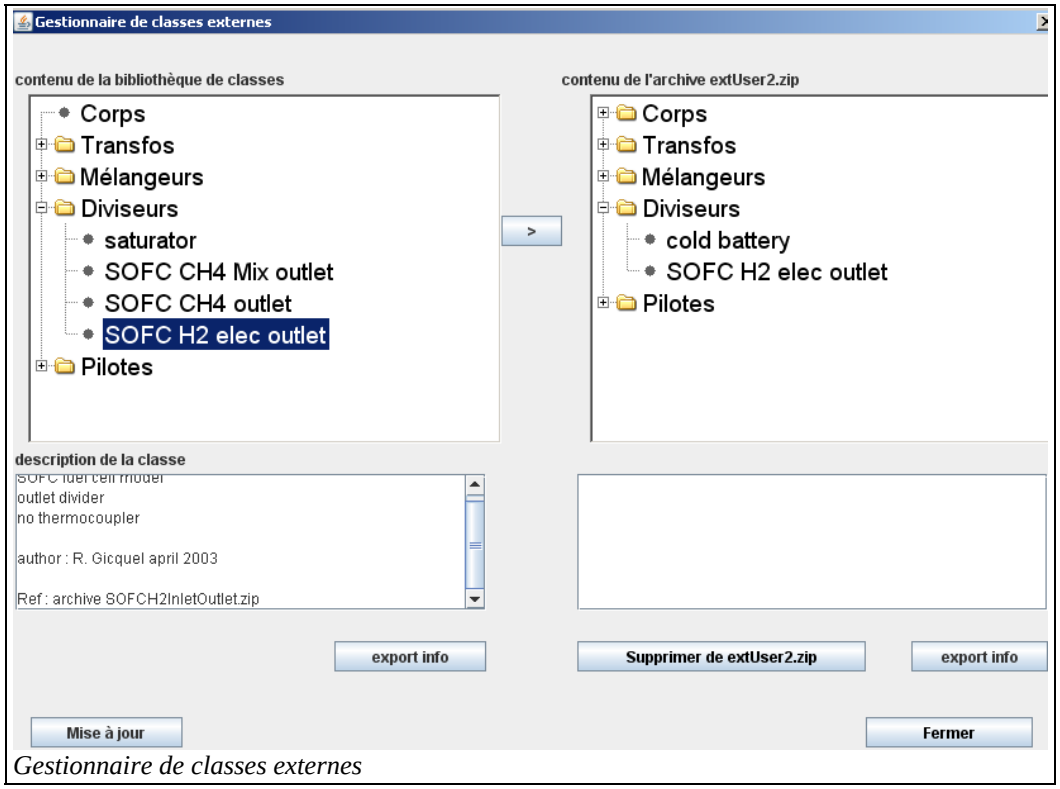
Placez les nouvelles classes non incluses dans l'archive dans le répertoire \externalClassLibrary2\extThopt.

[5] http://www.thermoptim.org/sections/logiciels/thermoptim/ressources/gestionnaire-classes

Si des bibliothèques externes sont chargées par les classes, copiez-les elles aussi à l'endroit où elles doivent l'être par rapport au répertoire \externalClassLibrary2.

Attention : il est recommandé de commencer par faire une copie du fichier extUser2.zip, pour pouvoir le retrouver en cas d'erreur. Toutefois, comme le gestionnaire commence par dupliquer extUser2.zip dans le fichier extUser2_copy.zip, il est possible en cas de problème de repartir de ce fichier en le renommant.

Lancez le Gestionnaire depuis le menu Spécial du simulateur. Si des conflits existent entre les noms de code attribués aux classes, des messages sont affichés, et écrits dans le fichier output.txt. L'écran ci-dessous s'affiche ensuite.



Gestionnaire de classes externes

Il présente, dans sa partie gauche, l'ensemble des classes disponibles dans le répertoire de la bibliothèque des classes, dont le chemin est indiqué dans le fichier "ExtLib.ini" (par défaut le répertoire "externalClassLibrary2"). Ces classes sont présentées par type (corps, transfos externes, mélangeurs externes, diviseurs externes, pilotes). Dans la partie droite, le contenu de l'archive extUser2.zip est affiché selon le même principe. Il est ainsi possible de facilement comparer les deux séries de classes.

Si vous sélectionnez une seule classe, dans l'une ou l'autre des listes, son descriptif apparaît dans la fenêtre située en dessous. Mais vous pouvez aussi faire une sélection multiple, auquel cas rien n'est affiché.

Vous pouvez transférer une sélection simple ou multiple depuis la bibliothèque des classes (à gauche), vers l'archive extUser2.zip, en cliquant sur la petite flèche centrale. 

Si une classe choisie dans la sélection existe déjà dans l'archive, un message vous en avertit, en rappelant les dates de création des deux classes, et vous demande si vous souhaitez remplacer l'existante, ou bien la conserver.

Vous pouvez supprimer de l'archive extUser2.zip une sélection simple ou multiple de la fenêtre de droite, en cliquant sur le bouton "Supprimer d'extUser2.zip". Un message vous demande, pour chaque classe sélectionnée, de confirmer la suppression, avec l'option de sauvegarder la classe dans la bibliothèque des classes. Dans ce dernier cas, si la classe existe déjà dans la bibliothèque, un message vous en avertit, en rappelant les dates de création des deux classes, et vous demande si vous souhaitez remplacer l'existante, ou bien la conserver.

Deux boutons intitulés "export info" vous permettent chacun de créer un fichier texte appelé "zipList.txt" ou "libraryList.txt" qui contient, pour chaque classe, son nom, son type et son descriptif.

Lors du chargement des fenêtres de gauche et de droite, un test est fait pour vérifier si des classes n'ont pas le même type, ce qui conduirait à des risques d'erreur dans Thermoptim. Si c'est le cas, il faut impérativement les différencier et recompiler les classes modifiées.

Enfin, le bouton "Mise à jour" permet de mettre à jour les deux fenêtres une fois un transfert ou des suppressions effectués, cette opération n'étant pas automatique.

Si vous avez modifié le fichier extUser2.zip, il est vivement recommandé de fermer Thermoptim puis de le réouvrir, car les bibliothèques ont été modifiées et Thermoptim peut avoir besoin d'être réinitialisé.

Attention aux classes externes liées (classes mères notamment) : il faut impérativement les ajouter ou les retirer ensemble, faute de quoi Thermoptim risque de ne pouvoir démarrer correctement (le problème serait bien sûr identique en utilisant un compacteur standard comme indiqué plus bas). Toutefois, si vous faites appel à des classes autres que celles que reconnaît Thermoptim, vous devrez les inclure à la main dans extUser2.zip, car ce processus ne peut être automatisé.

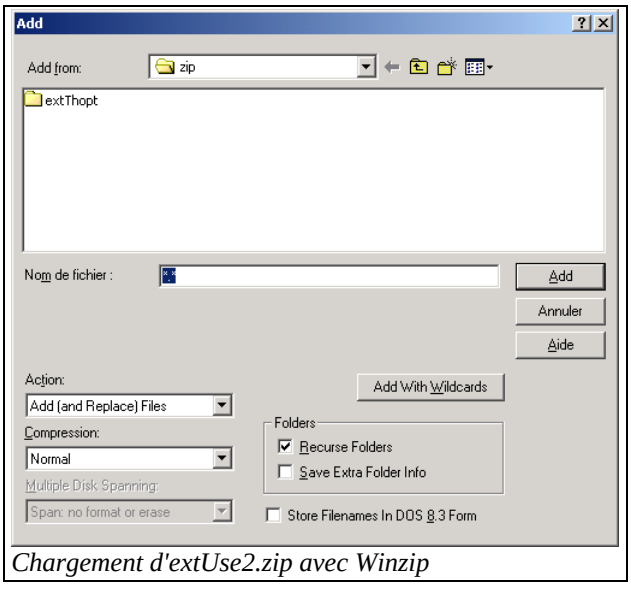
Si, après une modification de extUser2.zip, Thermoptim ne redémarre pas, ouvrez le fichier error.txt, qui contient généralement des indications sur les problèmes rencontrés.

Signalons enfin que le visualisateur de classes externes analyse à la fois extThopt2.zip et extUser2.zip, alors que le gestionnaire n'analyse que extUser2.zip.

Vous pouvez transférer à la main les classes externes de extThopt2.zip dans la bibliothèque pour les transférer dans extUser2.zip, puis les supprimer de extThopt2.zip, mais ce n'est pas recommandé.

3.6.2 Utilisation d'un compacteur standard

Il est aussi possible d'utiliser un compacteur standard comme WinZip, en lui faisant ajouter la nouvelle classe dans l'archive. Toutefois, il nous a été rapporté que des difficultés pouvaient survenir avec certaines versions de WinZip.



Chargement d'extUse2.zip avec Winzip

Avec WinZip, vous pouvez opérer comme suit :

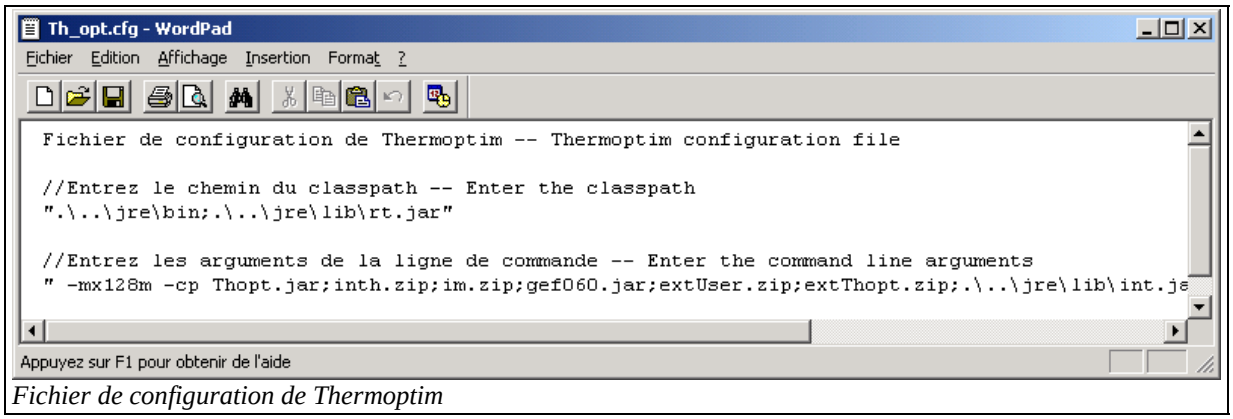
- créez un nouveau répertoire et appelez-le "zip", puis créez dans ce répertoire un sous-répertoire appelé "extThopt" ;
- copiez dans ce répertoire toutes les classes que vous désirez charger dans extUser2.zip (elles se trouvent dans le répertoire de création des classes de votre environnement de développement (/extThopt/) ;
- n'oubliez pas de quitter Thermoptim, faute de quoi l'archive extUser2.zip sera interdite en écriture ;

- ouvrez extUser2.zip, cliquez sur "Add", et placez-vous dans le répertoire "zip". Sélectionnez l'option "Recurse folders", puis cliquez sur "Add with Wildcards", qui ajoutera tout le contenu du sous-répertoire extThopt (il faut impérativement que le chemin d'accès aux classes dans l'archive soit correct, sinon Thermoptim ne pourra pas les charger).

En cas de difficulté, adressez un mél à contact@s4e2.com expliquant votre problème, avec en fichier attaché la classe à intégrer et la version précédente de extUser2.zip. L'archive reconstituée vous sera renvoyée dans les meilleurs délais.

3.6.3 Modification du classpath pour ajout de bibliothèques Java

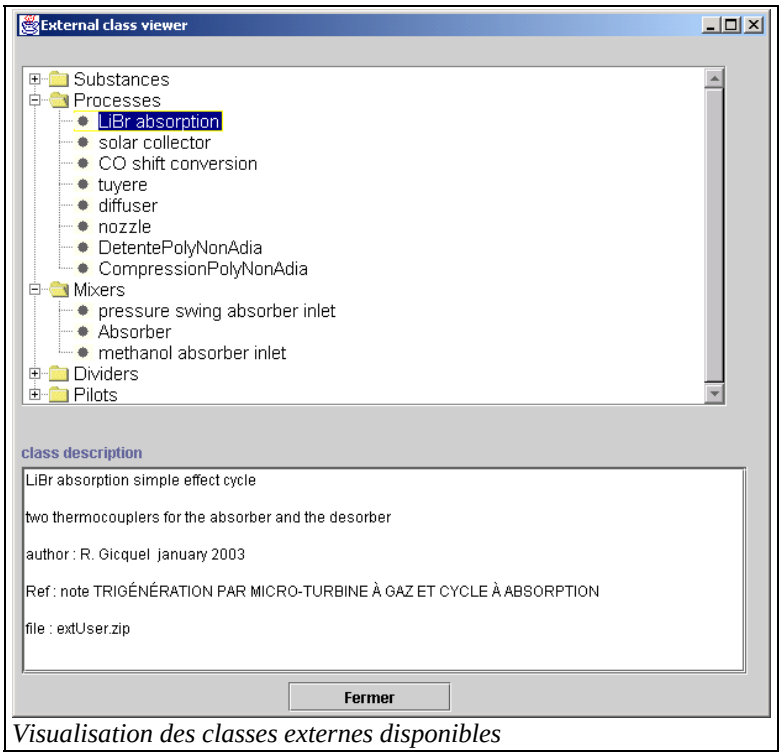
Si vous désirez adjoindre d'autres bibliothèques Java auxquelles vos classes font appel, il est nécessaire que vous indiquiez leur chemin d'accès dans le classpath. Sous Windows, ouvrez pour cela le fichier de configuration de Thermoptim "Thopt.cfg" dans un éditeur de texte, et ajoutez leurs chemins d'accès après "-cp", séparés par des point-virgule. Procédez de manière analogue sous Unix ou MacIntosh.



Fichier de configuration de Thermoptim

3.6.4 Visualisation des classes externes

Pour vous aider dans l'utilisation et la gestion des classes externes, la ligne "Visualisateur de classes externes" du menu "Spécial" permet de visualiser l'ensemble des classes externes disponibles. Leur liste est affichée, triée par type (corps, transfos, mélangeurs, diviseurs, pilotes) avec un court descriptif de la classe sélectionnée et indication de sa provenance (archives extThopt2.zip et extUser2.zip ainsi que classes en cours de développement).



Visualisation des classes externes disponibles

Cet écran peut être consulté pendant que vous mettez au point votre modèle. Notez que plus vous prendrez de soin dans la rédaction du descriptif de vos classes, plus cet écran servira aux utilisateurs. Rappelons aussi l'importance de bien préparer la documentation de vos classes, dont vous pourrez donner la référence dans leur descriptif. Cette documentation doit comprendre deux parties, l'une destinée aux utilisateurs, et l'autre aux concepteurs de classes. Une classe mal documentée sera difficile à utiliser ainsi qu'à modifier.

4 ENVIRONNEMENT DE DEVELOPPEMENT DES CLASSES EXTERNES

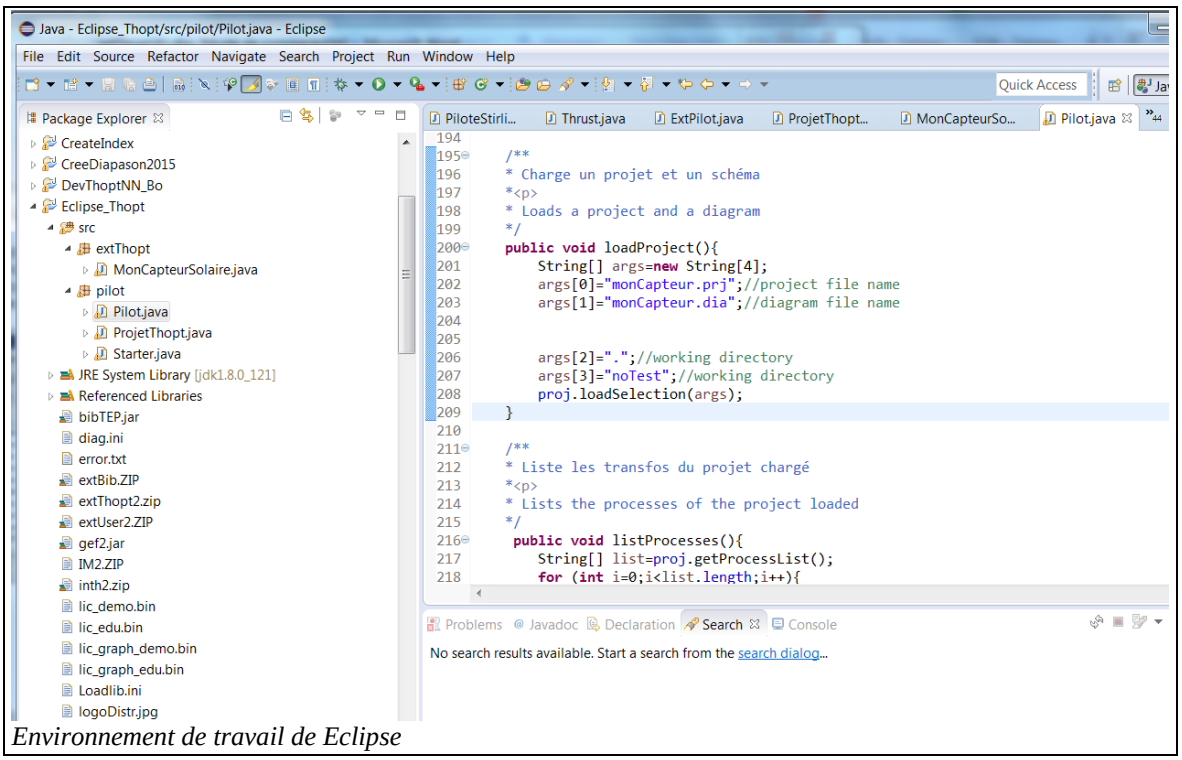
Pour mettre au point les classes externes, si vous ne disposez pas déjà d'un environnement de développement Java et si vous travaillez sous Windows, vous pouvez utiliser un outil disponible en freeware, appelé Eclipse. Développé en open source, Eclipse est téléchargeable sur Internet à l'adresse http://www.Eclipse.org.

Eclipse vous permet d'écrire vos classes en langage Java, de les compiler et de tester directement leur intégration dans Thermoptim. Pour vous faciliter la tâche, un espace de travail ("Workspace") de démarrage vous est proposé. Il s'appelle Eclipse_Thopt.

4.1 Présentation de l'environnement de travail

L'environnement de travail se présente comme suit :

- dans la partie gauche de l'écran apparaît la liste des fichiers Java, avec d'une part une petite classe appelée Starter qui permet de lancer le pilote, et d'autre part (dossier extThopt) les classes en cours de développement ;
- dans la partie droite apparaissent différents onglets contenant le code des classes.
- la fenêtre du bas est celle des messages.



Environnement de travail de Eclipse

4.2 Installation d'Eclipse

Pour procéder à l'installation de l'environnement de développement, connectez-vous sur le site d'Eclipse, et suivez les instructions.

Des explications sont données dans le portail Thermoptim-UNIT[6]. Toutefois nos explications ne peuvent être que succinctes et relatives aux particularités de Thermoptim.

[6] https://direns.mines-paristech.fr/Sites/Thopt/fr/co/environnement-developpement-eclipse.html

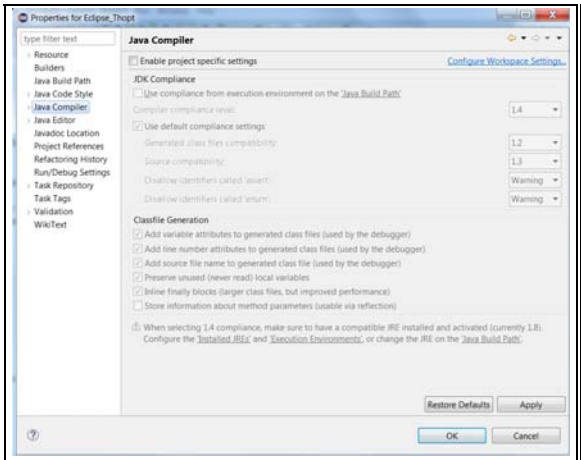
Compte tenu du potentiel d'Eclipse, c'est un environnement très utilisé, et vous trouverez sans difficulté des sites expliquant dans le détail comment procéder à son installation pour divers systèmes d'exploitation.

Dans ce qui suit, nous supposons que le lecteur connaît quelques rudiments du langage Java. Si ce n'est pas le cas, nous lui suggérons de se reporter à un ouvrage d'initiation avant de chercher à utiliser l'environnement de développement.

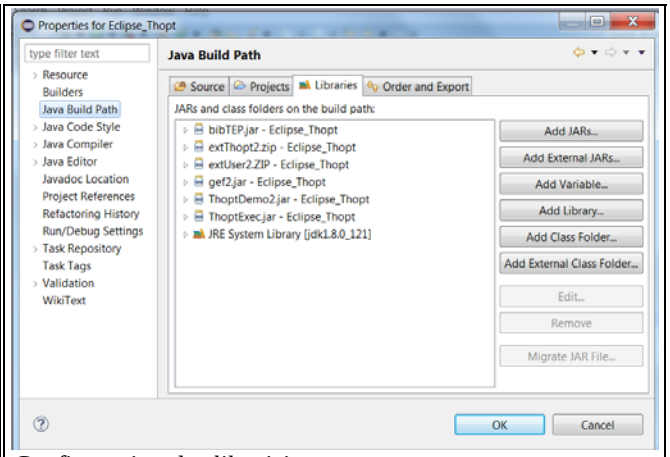
Commencez par installer le JDK et sa documentation, puis Eclipse. Lors de la première exécution d'Eclipse, le logiciel vous propose de lui associer les extensions de certains fichiers. Cliquez sur "OK" sauf si vous avez une bonne raison de ne pas faire ce choix. Choisissez bien ensuite les chemins d'accès aux répertoires contenant le JDK et sa documentation.

Décompactez le fichier Thopt25_EclipseDemo.zip et placez le dossier Eclipse_Thopt dans le répertoire d'installation de Eclipse. Il s'agit d'un Workspace comprenant tous les fichiers vous permettant de travailler avec Thermoptim. Ajoutez éventuellement l'archive ThoptEdu2.jar ou Thopt2.jar de Thermoptim et inth2.zip, ainsi que vos fichiers de licence (tous ces fichiers se trouvent dans le répertoire d'installation de Thermoptim).

Pour que le projet puisse s'exécuter convenablement, il faut qu'il soit complètement configuré. Ouvrez le menu Projects/Properties, puis Java Compiler. Vous obtenez un écran qui ressemble à celui ci-dessous à gauche. Vérifiez quels JDK apparaissent et lequel est coché.



Configuration des JDK



Configuration des librairies

Il vous faut ensuite indiquer les bibliothèques requises par votre application, en l'occurrence l'ensemble de celles dont Thermoptim a besoin pour travailler, plus Thermoptim lui-même. Pour cela, ouvrez le menu Projects/Properties, puis Java Build Path, puis Libraries. Si les bibliothèques de la figure n'apparaissent pas, cliquez sur "Add JARs", et sélectionnez dans le répertoire "Eclipse_Thopt" les fichiers qui apparaissent dans l'écran ci-dessous à droite : ThoptEdu2.jar ou Thopt2.jar, extUser2.zip, im2.zip, inth2.zip, extThopt2.zip, gef.jar. Une fois les archives sélectionnées, cliquez sur "OK". Vous pouvez maintenant commencer à développer vos classes externes comme indiqué section 3.

Le code source des classes de base du paquet extThopt est donné sous forme d'une archive. Ces classes sont déjà contenues au format compilé dans l'archive extThopt2.zip, et ne doivent donc en principe pas être chargées dans l'environnement de développement. Il est possible de les modifier, mais vous risquez de ne plus être compatible avec les versions "officielles" des classes externes de Thermoptim.

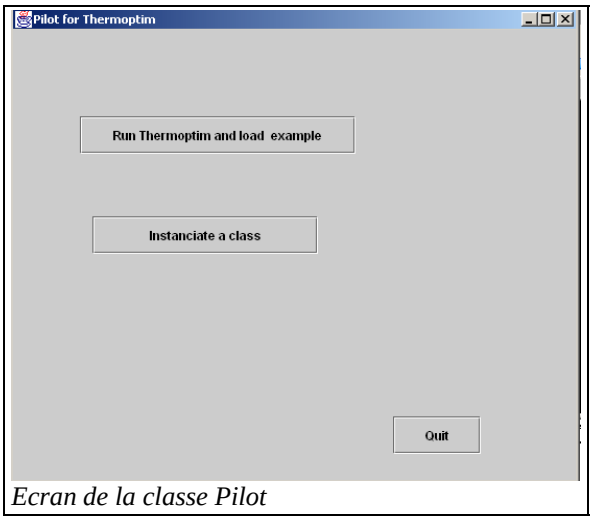
L'annexe 1 présente leur usage et donne un certain nombre d'exemples de mise en œuvre, l'annexe 2 donne le code des méthodes de Thermoptim renvoyant des Vector dont il est souhaitable de connaître le contenu détaillé pour programmer les classes externes, et l'annexe 3 contient un bref descriptif des méthodes de la classe Util.

Enfin, il est recommandé de parcourir le code des classes de base d'extThopt qui contient diverses méthodes qui ont été écrites pour simplifier le travail de développement des classes externes, et dont l'usage est illustré dans les exemples de classes qui sont fournis. Leur API est disponible dans le dossier "api_extThopt".

4.3 Emulation de Thermoptim à partir de Eclipse

Pour pouvoir émuler Thermoptim afin de tester une classe en cours de développement, nous avons implémenté dans Eclipse un espace de travail particulier, dans lequel trois classes que nous présentons ici rapidement jouent un rôle clé :

- Starter est la classe lancée au démarrage de Eclipse. Elle initialise les fichiers de sortie ("output.txt" et "error.txt", et instancie la classe Pilot du paquet pilot ;
- Pilot est la classe qui permet de lancer Thermoptim, en instanciant la classe ProjetThopt du même paquet et en exécutant sa méthode runThopt(). Ensuite, la méthode loadProject() affichée à l'écran de la figure présentant l'environnement de travail permet de charger un projet et un schéma pour pouvoir tester le fonctionnement des classes externes en cours de développement. Dans l'exemple ci-dessus, nous avons entré un seul exemple, mais on peut en prévoir plusieurs, étant entendu que seul le dernier chargé dans le tableau args[] est pris en compte (cette manière de faire permet de changer d'exemple très simplement, en déplaçant les deux lignes appropriées) ;
- ProjetThopt, qui hérite de la classe Projet de Thermoptim, possède une méthode appelée getInternalClasses() dans laquelle sont définies les classes externes à charger dans Thermoptim.



Ecran de la classe Pilot

Une fois les méthodes loadProject() et getInternalClasses() modifiées, il suffit de recompiler le projet et de l'exécuter pour que la fenêtre de chargement de Thermoptim et de son projet apparaisse.

En cliquant sur "Run Thermoptim and load example", vous chargez l'exemple désiré. Vous disposez alors de toutes les fonctionnalités du progiciel, et pouvez notamment modifier votre exemple et le sauvegarder si vous disposez d'une version autre que celle de démonstration. Si un fichier de projet ou de schéma est absent, un message vous en avertit.

De cette manière, vous pouvez développer votre classe et la mettre au point en testant directement son fonctionnement dans Thermoptim, ce qui permet de gagner beaucoup de temps. Une fois que la classe est au point, il vous suffit de l'archiver dans extUser2.zip, pour pouvoir la distribuer afin qu'elle soit utilisée en dehors de l'environnement de développement.

Sur le plan pratique, opérez comme suit :

- en suivant les indications données au chapitre 3, commencez par créer votre classe externe, soit en adaptant une de celles qui sont fournies (les fichiers .java sont contenus dans le répertoire "src" du Workspace), soit à partir de l'assistant fourni par Eclipse (menu Project/New Class), et compilez-la (menu Build/Compile File)
- déclarez votre classe dans la méthode getInternalClasses() de ProjetThopt, en suivant les indications fournies, afin qu'elle soit chargée dans Thermoptim lors du chargement du progiciel, et recompilez ProjetThopt
- pour tester votre classe, dupliquez un exemple de projet et de schéma, entrez les nouveaux noms dans les dernières lignes du tableau args[] dans la méthode loadProject() de Pilot, et recompilez Pilot
- lancez l'exécution (menu Project/Build Project, ou Ctrl+F11), et cliquez sur "Run Thermoptim and load example"

ANNEXE 1 : METHODES DE THERMOPTIM ACCESSIBLES DE L'EXTERIEUR

Dans cette annexe, nous présentons les méthodes de Thermoptim accessibles depuis les classes externes, en indiquant quel est leur usage et en donnant un certain nombre d'exemples de leur mise en œuvre. Leur API est quant à elle définie dans la JavaDoc qui se trouve dans le dossier api_Thermoptim de l'environnement de développement.

package rg.corps

Les noms de deux classes ne sont pas camouflés : rg.corps.Corps, classe mère de tous les corps et rg.corps.CorpsExt, classe mère des corps externes.

Les méthodes suivantes sont accessibles :

Dans Corps

Pour calculer l'état complet du corps
public void CalcPropCorps (double T, double p, double x)

```
  public double[] getState(){//modRG 2008
      double[]res=new double[8];
      res[0]=T;
      res[1]=P;
      res[2]=xx;
      res[3]=V;
      res[4]=U;
      res[5]=H;
      res[6]=S;
      res[7]=Cv;
      res[8]=Xh;
      return res;
   }
```

Pour instancier un corps dont on connaît le nom
public static Object createSubstance(String nomCorps)

exemple :

```
Object obj=Corps.createSubstance("eau");//instanciation du corps, encapsulé dans un Object
Corps leau=(Corps)obj;//transtypage de l'Object
double T=leau.getT_from_hP(3400,50);//inversion en T de l'enthalpie à pression connue
System.out.println("T eau : "+T);
```

Pour obtenir le Cp d'un corps connaissant son Cv
public double getCp(double Cv)

Méthodes d'inversion des fonctions d'état
public double getP_from_hT(double hv,double T)
public double getP_from_sT(double sv,double T)
public double getP_from_sT(double sv,double T, double pmin, double pmax)
public double getP_from_vT(double v,double T)
public double getT_from_hP(double hv,double P)
public double getT_from_sP(double sv,double P)
public double getT_from_sv(double s,double v)
public double getT_from_uv(double u,double v)
public double[]getPTx_from_sh (double s,double h), utilisée pour les inversions du diagramme de Mollier, et qui retourne un tableau de trois valeurs : P, T, et x

public double[] getQuality(), utilisée avec toutes les méthodes précédentes, renvoie la composition à l'équilibre liquide-vapeur. En particulier, pour les corps purs, getQuality()[0] représente le titre en vapeur.

Méthodes de calcul de la pression ou la température de saturation
public double getSatPressure(double T, double x)
public double getSatTemperature(double P, double x)

Ces deux méthodes existent avec des signatures diverses afin de permettre de calculer les mélanges externes.

Pour obtenir l'état d'un corps
public Vector getSubstProperties()

La méthode getSubstProperties() de extProcess fait indirectement appel à cette méthode pour charger les valeurs des double Tsubst,Psubst,Xsubst,Vsubst,Usubst,Hsubst,Ssubst,Msubst,typeSubst;

exemple :

lecorps.CalcPropCorps(Tpoint+1, Ppoint, Xpoint);//calcule l'état du corps
getSubstProperties(nomCorps);//récupère les valeurs calculées et les charge notamment dans Hsubst
double Cp=(Hsubst-H);
typeSubst vaut 1 pour l'eau, 2 pour une vapeur,3 pour un gaz pur, 4 pour un gaz composé, 5 pour un corps externe simple, et 6 pour un corps externe du type Mixture.

Initialisation d'un corps externe
public void initCorpsExt(double M, double PC, double TC, double VC,
double Tmini, double Tmaxi, double Pmini, double Pmaxi, int typeCorps)

public double[] getMolarWeights(), fournit les masses molaires d'un mélange externe

Définit un commentaire pour un corps externe (formule chimique, composition…)
public void setComment(String comment)

Donne un nom à un corps externe
public void setNom(String name)

Charge les valeurs des calculs effectués par les corps externes
public void setState(double P, double T, double xx,
double U, double H, double S, double V, double Cv,
double Xh)

Pour obtenir la composition d'un gaz
public Vector getGasComposition()

Ce Vector, qui fournit aussi les masses molaires et la valeur du PCI, a la structure suivante :

```
   public Vector getGasComp(){
      Vector vComp=new Vector();
      Double[]fracMol= new Double[nb_comp],fracMass= new Double[nb_comp],molarMass= new
Double[nb_comp];
      for(int j=0;j<nb_comp;j++){
                  fracMol[j]=new Double(fract_mol[j]);
                  fracMass[j]=new Double(fract_mass[j]);
                  molarMass[j]=new Double(M*fract_mass[j]/fract_mol[j]);
      }
      vComp.addElement(new Integer(nb_comp));
      vComp.addElement(nom_gaz_comp);
      vComp.addElement(fracMol);
      vComp.addElement(fracMass);
      vComp.addElement(molarMass);
      vComp.addElement(new Double(PCI));
```

```
    return vComp;
  }
```

Exemple d'utilisation :

getPointProperties(cO2Point);//récupère les propriétés du point cO2Point
Vector nouvComp=lecorps.getGasComposition();//récupère la composition du gaz
updateGasComposition(nouvComp, cO2Process);//met à jour la composition du gaz

Pour mettre à jour la composition d'un gaz
public void updateGasComposition (Vector vComp)

Cette méthode sert généralement à modifier la composition d'un gaz, auquel cas le Vector vComp doit comporter les noms des composés et au moins les fractions molaires, selon la syntaxe présentée ci-dessous. Elle permet aussi de modifier l'humidité absolue d'un gaz, si le nombre de composants entré comme premier élément de vComb a une valeur négative (voir ci-dessous ou section 3.2.3).

La méthode à laquelle le relais est passé dans GazIdeal est la suivante :

```
  public void updateGasComp(Vector vComp){
     Integer i=(Integer)vComp.elementAt(0);
     int nComp=i.intValue();
    if(nComp>=0){//cas général de la mise à jour de la composition
        String[] Comp= new String[nComp];
        double[]fractmol= new double[nComp],fractmass= new double[nComp];
        Double[]fracMol= new Double[nComp],fracMass= new Double[nComp];
                Comp=(String[])vComp.elementAt(1);
                fracMol=(Double[])vComp.elementAt(2);
                fracMass=(Double[])vComp.elementAt(3);
        for(int j=0;j<nComp;j++){
                    Double f=(Double)fracMol[j];
                    fractmol[j]=f.doubleValue();
                    f=(Double)fracMass[j];
                    fractmass[j]=f.doubleValue();
        }

        updateSubstComp(Comp,fractmol,fractmass);
     }
     else{//modifications gaz humides //modRG 01/06
        String task=(String)vComp.elementAt(1);
        String value=(String)vComp.elementAt(2);
        if(task.equals("setGasHum")){//sets the gas humidity
           double w=Util.lit_d(value);
           setGasHum(w);
        }
     }
  }
```

Dans CorpsExt

public double getLiquidConductivity(double T)
public double getLiquidViscosity(double T)
public double getVaporConductivity(double T)
public double getVaporViscosity(double T)

package rg.thopt

Les noms des classes suivantes ne sont pas camouflés :

- rg.thopt.Projet, classe du simulateur,
- rg.thopt.TransfoExterne, rg.thopt.DividerExterne, rg.thopt.MixerExterne et rg.thopt.ComposantExt, qui servent à définir les composants externes
- rg.thopt.Compression et rg.thopt.ComprExt, qui servent à définir les compresseurs externes.

Les méthodes suivantes sont accessibles :

Dans Projet

lance un recalcul dans Thermoptim à partir du pilote
public void calcThopt()

renvoie un code permettant au pilote de savoir si deux composants de l'éditeur sont connectés ou non :
public int getConnectionCode(String[] args)

1 si le composant 1 est en amont du composant 2, -1 dans le cas contraire, et 0 s'ils ne sont pas connectés.

La structure des arguments est la suivante :

```
String[] args=new String[4];
args[0]="Expansion";//type du composant 1
args[1]="turbine";//nom du composant 1
args[2]="Exchange";//type du composant 2
args[3]="régen gaz";//nom du composant 2
```

Fournit la liste des composants présents dans l'éditeur de schémas
public String[] getEditorComponentList()

sous la forme name="nomComposant"+tab+type="typeComposant". Cette liste peut être décomposée simplement avec extr_value("name") et extr_value("type")

Listes d'éléments
public String[] getHxList ()
public String[] getNodeList ()
public String[] getPointList ()
public String[] getProcessList()
public String[]getSetPressureList ()
public String[] getSubstanceList ()

Vecteur de propriétés dont la structure dépend du type d'élément considéré
public Vector getProperties(String[] args)

type=args[0]; "subst" / "point" / "process" / "heatEx"
nomType=args[1];

Cette méthode est fondamentale : elle permet d'accéder à l'ensemble des données relatives à un type primitif dès lors qu'on en connaît le type et le nom. Elle est très utilisée par les méthodes génériques du package extThopt.

On trouvera en annexe 2 le code interne de Thermoptim correspondant, qui permet de savoir quelle est la structure du Vector retourné (qui dépend du type du type primitif appelé).

chargement des exemples par le pilote de Thermoptim
public void loadSelection(String[] args)

méthode exécutée dans Thermoptim chaque fois qu'un élément (point, transfo, noeud, échangeur) est calculé
public void notifyElementCalculated(String[] element)

element est construit comme suit :

```
if(pt instanceof Transfo) element[0]="process";
else if(pt instanceof PointCorps) element[0]="point";
else if(pt instanceof Node) element[0]="node";
else if(pt instanceof HeatEx) element[0]="heatEx";
element[1]=pt.getName();
element[2]=pt.getType();
```

Ainsi, il peut directement être utilisé comme argument dans getProperties() : getProperties(element) fournit l'état de l'élément considéré.

permet au pilote de reprendre la main entre deux recalculs, pour modifier certains paramètres de Thermoptim par exemple
public void setControls()

instanciation de l'éditeur de schémas et désactivation de certaines fonctions par le pilote de Thermoptim
public void setupThopt()

instanciation de l'éditeur de schémas par le pilote de Thermoptim
public void openEditor()

fermeture de l'éditeur de schémas par le pilote de Thermoptim
public void closeEditor ()

chargement de classes externes par le pilote de Thermoptim
public Vector getInternalClasses()

opérations sur les diagrammes à partir du pilote de Thermoptim
public void setupChart (Vector v)

récupération de l'instance du pilote de Thermoptim
public PilotFrame getPilotFrame()

exemple d'ouverture de diagramme, de choix du corps et de superposition de deux cycles

```
if(proj!=null)proj.calcThopt();//si nécessaire instanciation de Thermoptim
Vector v=new Vector();
v.addElement("openDiag");//ouverture d'un diagrammes
v.addElement("1");//type de diagramme (1 pour vapeurs, 2 pour gaz idéaux, 3 pour psychrométriques)
proj.setupChart(v);
v=new Vector();
v.addElement("selectSubstance");//sélection d'un corps
v.addElement("R134a");//nom du corps
proj.setupChart(v);

v=new Vector();
v.addElement("readCycle");//chargement d'un cycle
v.addElement("frigoR134aFin.txt");//nom du fichier
proj.setupChart(v);
v=new Vector();
v.addElement("readCycle");//chargement d'un cycle
v.addElement("frigoR134a.txt");//nom du fichier
proj.setupChart(v);
```

mise à jour d'un point, d'une transfo ou d'un échangeur
public void updatePoint(Vector v)
public void updateProcess(Vector v)
public void updateNode(Vector v)
public void updateHX(Vector v)

Le code de ces méthodes est donné en annexe

exemple la méthode updateStraightlyConnectedProcess() d'ExtNode contient la séquence suivante :

```
Vector vPoint=new Vector();
vPoint.addElement(outletPointName);
vPoint.addElement(Util.aff_b(updateT));
vPoint.addElement(Util.aff_d(T));
vPoint.addElement(Util.aff_b(updateP));
vPoint.addElement(Util.aff_d(P));
vPoint.addElement(Util.aff_b(updateX));
vPoint.addElement(Util.aff_d(x));
proj.updatePoint(vPoint);
```

Dans PilotFrame

Pour connaître le Projet appelant
public void getProjet ()

Pour relire les paramètres du pilote sauvegardés dans le fichier de projet
public void readCompParameters(String ligne_data)

Pour sauvegarder les paramètres du pilote dans le fichier de projet
public String saveCompParameters()

Dans ComposantExt

Pour calculer le composant
public void calcProcess()

Pour obtenir le nom du composant
public String getCompName()

Pour obtenir le type du composant
public String getCompType()

Pour obtenir les 7 valeurs prises en compte dans le bilan exergétique d'un composant
public double[] getCompExergyBalance(String[]args)

Pour relire les paramètres du composant sauvegardés dans le fichier de projet
public void readCompParameters(String ligne_data)

Pour sauvegarder les paramètres du composant dans le fichier de projet
public String saveCompParameters()

Pour initialiser l'écran du composant ainsi que sa configuration logique interne
public void setCompFrame(Object obj)
public void setDivFrame (Object obj)
public void setMixFrame (Object obj)

Pour mettre à jour la transfo (débit, point amont, point aval, types d'énergie)
public void setupFlow(double flow)
public void setupPointAmont(Vector vProp)
public void setupPointAmont(PointThopt pt)
public void setupPointAval (Vector vProp)
public void setupPointAval (PointThopt pt)
public void updateProcess(Vector vEner)

Pour construire le Vector vProp, ExtProcess dispose d'une méthode générique, qui encapsule comme il le faut l'état du point :

```
public Vector getProperties(){
        Vector vProp=new Vector();
        vProp.addElement(lecorps);//Corps
        vProp.addElement(nomCorps);//Corps
        vProp.addElement(new Double(Tpoint));
        vProp.addElement(new Double(Ppoint));
        vProp.addElement(new Double(Xpoint));
        vProp.addElement(new Double(Vpoint));
        vProp.addElement(new Double(Upoint));
        vProp.addElement(new Double(Hpoint));
        return vProp;
}
```

exemple :

```
tfe.setupPointAval(getProperties());
```

La méthode updateProcess() de ComposantExt permet d'affecter les valeurs que l'on désire aux différents types d'énergie. Elle est facilement mise en œuvre avec la méthode setEnergyTypes d'ExtThopt (cf. section 3.2.3).

exemple :

```
tfe.updateProcess(setEnergyTypes(useful,purchased,other));
```

Dans TransfoExterne

Pour mettre à jour un thermocoupleur
public void updateThermoCouplers(Vector vTC)

Pour construire le Vector vTC, ExtProcess dispose d'une méthode générique, qui encapsule comme il le faut les valeurs à transmettre :

```
protected void updateThermoCoupler(String type, double Tin, double Tout, double Q, double flow){
   Vector vTC=new Vector();
   vTC.addElement(type);//type du thermocoupleur considéré
   Double d=new Double(Tin);
   vTC.addElement(d);
   d=new Double(Tout);
   vTC.addElement(d);
   d=new Double(Q);
   vTC.addElement(d);
   d=new Double(flow);
   vTC.addElement(d);
   tfe.te.updateThermoCouplers(vTC);
}
```

Fournit des valeurs du thermocoupleur du type appelé
public Vector getThermoCouplerData(String thermoCouplerType)

Le Vector retourné ne contient pour le moment que le nom du thermocoupleur, mais cela évoluera sans doute.

Dans DividerExterne

Met à jour le diviseur

public void updateDivider (Vector vTC)

Pour construire le Vector vTC, ExtNode dispose d'une méthode générique, qui encapsule comme il le faut les valeurs à transmettre :

```
protected Vector getUpdateVector (Vector[]vTransfo,Vector[]vPoints, double TGlobal, double hGlobal){
   Vector vTC=new Vector();
   for(int j=0;j<vTransfo.length;j++){
      vTC.addElement(vTransfo[j]);
   }
   for(int j=0;j<vPoints.length;j++){
      vTC.addElement(vPoints[j]);
   }
   Vector vGlobal=new Vector();
   vGlobal.addElement(new Double(TGlobal));
   vGlobal.addElement(new Double(hGlobal));
   vTC.addElement(vGlobal);
   de.de.updateDivider(vTC);
}
```

L'appel se fait alors dans ExtDivider de la manière suivante :

```
protected void updateDivider(Vector[]vTransfo,Vector[]vPoints, double TGlobal, double hGlobal){
   Vector vTC=getUpdateVector(vTransfo,vPoints, TGlobal, hGlobal);
   de.de.updateDivider(vTC);
}
```

Ces méthodes font appel aux tableaux de Vector vTransfo et vPoints représentant la veine principale et les branches, dont il peut être utile d'automatiser la construction, comme dans la classe Desorber, où des méthodes génériques appelées d'un nom ayant un sens physique construisent les Vector :

```
private void setupRichSolution(double m, double T, double P, double X){
   vTransfo[0]=new Vector();
   vPoints[0]=new Vector();
   vTransfo[0].addElement(richSolutionProcess);
   vTransfo[0].addElement(new Double(m));

   vPoints[0].addElement(richSolutionPoint);
   vPoints[0].addElement(new Double(T));
   vPoints[0].addElement(new Double(P));
   vPoints[0].addElement(new Double(X));
}
```

exemple : mise à jour du diviseur externe Desorber après calcul :

```
vTransfo= new Vector[nBranches+1];
vPoints= new Vector[nBranches+1];
setupRichSolution(msr,Tsr,Psr,Xsr);
setupPoorSolution(msp,Tgen,P,Xsp);
setupRefrigerant(mr,Trefr,Prefr,1);
updateDivider(vTransfo,vPoints,Tsr,Hsr);
```

Fournit des valeurs du diviseur
public Vector getDividerData()

Actuellement non utilisé, car faisant double emploi avec getProperties() de Projet exécutée pour un nœud. ExtDivider fournit pour cela la méthode public void getDividerStructure() qui charge la structure du nœud.

Pour mettre à jour un thermocoupleur (voir TransfoExterne)
public void updateThermoCouplers(Vector vTC)

Fournit des valeurs du thermocoupleur du type appelé (voir TransfoExterne)
public Vector getThermoCouplerData(String thermoCouplerType)

Dans MixerExterne

Met à jour le mélangeur (voir DividerExterne)
public void updateMixer (Vector vTC)

Fournit des valeurs du mélangeur (voir DividerExterne)
public Vector getMixerData ()

Pour mettre à jour un thermocoupleur (voir TransfoExterne)
public void updateThermoCouplers(Vector vTC)

Fournit des valeurs du thermocoupleur du type appelé (voir TransfoExterne)
public Vector getThermoCouplerData(String thermoCouplerType)

Pour initialiser la combustion puis effectuer les calculs demandés
public void calcExternalCombustion(Vector vSettings)

La stucture de vSettings est la suivante :

```
Vector vSettings=new Vector();
vSettings.addElement(NewComburSubstance);//oxidizer ideal gas
vSettings.addElement(fuelSubstance) );//fuel ideal gas
vSettings.addElement(new Double(comburFlow));//oxidizer flow rate
vSettings.addElement(new Double(fuelFlow));//fuel flow rate
vSettings.addElement(new Double(lambda));//lambda
vSettings.addElement(new Double(Tfluegas));//T flue gas
vSettings.addElement(new Double(diss));//dissociation rate
vSettings.addElement("true");//dissociation boolean
vSettings.addElement(new Double(Tquench);//Tfigeage
vSettings.addElement(new Double(a));//a
vSettings.addElement(new Double(hf0carb));//hf0carb
vSettings.addElement("false");//comb CHa
vSettings.addElement(new Double(Hcarb));//hc
vSettings.addElement(new Double(Hcarb));//uc
vSettings.addElement("true");//setFuelFlow
vSettings.addElement(new Double(heatLoad));//heat load
vSettings.addElement("true");//open system
vSettings.addElement("true");//IcalcDirect //si true on calcule Tad à partir de lambda
vSettings.addElement(synGasSubstance); //flue gas ideal gas
vSettings.addElement(new Double(Tcombur));//T amont
vSettings.addElement(new Double(Hcombur));//H amont
vSettings.addElement(new Double(eff));//rendement chambre
```

Récupére les résultats d'un calcul de combustion
public Vector getExternalCombustionResults()

La stucture du Vector est la suivante :

```
Vector vResults=new Vector();
vResults.addElement(flueGas); //flue gas ideal gas
vResults.addElement(new Double(Tfluegas));
vResults.addElement(new Double(lambda));
vResults.addElement(new Double(flueGasFlow));
vResults.addElement(new Double(eta_comb));
vResults.addElement(new Double(k_CO));
```

```
vResults.addElement(new Double(DeltaH));
vResults.addElement(new Double(DeltaS));
vResults.addElement(new Double(W_moteur));
```

Dans ComprExt

```
public double getComprFlow()
public double getComprIsentropicEfficiency()
public String getComprName()
public double getComprRatio()
public double getComprRelaxValue()
public double getComprRotationSpeed()
public double getComprSweptVolume()
public String getComprType()
public double getComprVolumetricEfficiency()
public double getComprVolumetricFlow()
public double getPumpFanVolumetricFlow()
public Vector getSupplyPointProperties()
public void makeComprDesign()
public void readComprParameters(String line)
public String saveComprParameters()
public void setComprFrame(Object ob)
```

ANNEXE 2 : CODE DE METHODES DE THERMOPTIM

Nous donnons ici le code de certaines méthodes de Projet et Corps dont la structure doit être fidèlement reproduite si l'on veut les utiliser dans les sous-classes.

Méthode getProperties() de Projet

```
        /**
        * Fournit les valeurs thermodynamiques des divers types primitifs
        *<p>
        * Gives thermodynamic values for the various element
        * @param args String[]
        * @param type=args[0];  "project" / "subst" / "point" / "process" / "heatEx"
        * @param nomType=args[1];
        **/
        public Vector getProperties(String[] args){
           String type, nomType;
           type=args[0];
           nomType=args[1];
           Vector vProp=new Vector();

           if(type.equals("project")){
              vProp.addElement(Util.unit_m);//unité des débits [0]
              vProp.addElement(Util.unitT);//unité des températures[1]
              vProp.addElement(new Double(Util.T0Exer)); // température de l'environnement[2]
              vProp.addElement(new Double(getEnergyBalance()[0]));//énergie payante [3]
              vProp.addElement(new Double(getEnergyBalance()[1])); //énergie utile [4]
              vProp.addElement(new Double(getEnergyBalance()[2])); //efficacité [5]
}
           else if(type.equals("subst")){
              Corps pt=getCorps(nomType);
              if(pt!=null){
                 vProp=pt.getSubstProperties();
              }
           }
           else if(type.equals("point")){
              PointCorps pt=getPoint(nomType);
              if(pt!=null){
                 vProp.addElement(pt.lecorps);//Substance [0]
                 vProp.addElement(pt.lecorps.getNom());//Substance name [1]
                 vProp.addElement(new Double(pt.getT()));//Temperature [2]
                 vProp.addElement(new Double(pt.getP()));//Pressure [3]
                 vProp.addElement(new Double(pt.getXx()));//Quality [4]
                 vProp.addElement(new Double(pt.getV()));//Volume [5]
                 vProp.addElement(new Double(pt.getU()));//Internal energy [6]
                 vProp.addElement(new Double(pt.getH()));//Enthalpy [7]
                 vProp.addElement(new Double(pt.getS()));//Entropy [8]
                 String setTsat="set_Tsat="+Util.aff_b(pt.JCheckSetTsat.isSelected());
                 vProp.addElement(setTsat);//setTsat [9]
                 vProp.addElement(new Double(pt.dTsat_value.getValue()));//Dtsat [10]
                 String setpsat="set_psat="+Util.aff_b(pt.JCheckSetPsat.isSelected());
                 vProp.addElement(setpsat);//setpsat [11]

                 //wet gas values
                 vProp.addElement(new Double(pt.w_value.getValue()));//specific humidity [12]
                 vProp.addElement(new Double(pt.epsi_value.getValue()));//relative humidity [13]
                 vProp.addElement(new Double(pt.qprime_value.getValue()));//specific enthalpy [14]
                 vProp.addElement(new Double(pt.tprime_value.getValue()));//adiabatic temperature [15]
                 vProp.addElement(new Double(pt.tr_value.getValue()));//dew point temperature [16]
                 vProp.addElement(new Double(pt.v_spec_value.getValue()));//specific volume [17]
```

```
                vProp.addElement(new Double(pt.cond_value.getValue()));//condensates [18]
                vProp.addElement(new Double(pt.lecorps.M_sec));//Dry gas molar mass [19]
                vProp.addElement("dummy");//réserved if necessary for wet gases [20]
                vProp.addElement("dummy");//réserved if necessary for wet gases [21]

                //pressure setting
                if(pt.JCheckPvalue.isSelected()){
                  PointCorps pt2=(PointCorps)pt;
                    vProp.addElement("pressureSet="+pt2.thePres.getName());//set pressure [22]
                    vProp.addElement(new Double(pt2.pressCorrFact_value.getValue()));//correction factor [23]
                }
                else {
                    vProp.addElement("pressureSet=");// [22]
                    vProp.addElement(new Double(1.));// [23]
                }
            }
        }
        else if(type.equals("process")){
            Transfo tf=getTransfo(nomType);
            vProp.addElement(tf.getType());//[0]
            if(tf!=null){
                vProp.addElement(tf.getPointAmont().getName());// [1]
                vProp.addElement(tf.getPointAval().getName());// [2]
                vProp.addElement(new Double(tf.getFlow()));//flow rate [3]
                vProp.addElement(new Double(tf.DeltaH));//Enthalpy [4]
                vProp.addElement(tf.ener_type_value.getText());//Energy type [5]
                String direct="calcDirect="+Util.aff_b(tf.IcalcDirect);
                vProp.addElement(direct);//true if direct calculation [6]
                String ouvert="openSyst="+Util.aff_b(tf.JCheckOuvert.isSelected());
                vProp.addElement(ouvert);//true for open system [7]
                String setflow="setFlow="+Util.aff_b(tf.JCheckSetFlow.isSelected());
                vProp.addElement(setflow);//true for set flow [8]
                String inletProcess="null";
                if(tf.isInletStrConnected())inletProcess=tf.getTransfoAmontName();//inlet process name (if
exits)
                vProp.addElement(inletProcess);// [9]
                String outletProcess="null";
                if(tf.isOutletStrConnected())outletProcess=tf.getTransfoAvalName();//outlet process name (if
exits)
                vProp.addElement(outletProcess);// [10]
                if(tf instanceof ComprExpan){
                    ComprExpan cb=(ComprExpan)tf;
                    vProp.addElement(new Double(cb.rendIs));// rendement isentropique [11]
                    vProp.addElement(new Double(cb.Q_value.getValue()));// chaleur apportée (si non
adiabatique) [12]
                }
                if(tf instanceof Combustion){
                    Combustion cb=(Combustion)tf;
                    String fuel="null";
                    if(cb.getFuel()!=null)fuel=cb.getFuel().getName();
                    vProp.addElement(fuel);// [11]
                    vProp.addElement(new Double(cb.lambda_value.getValue()));//lambda [12]
                    vProp.addElement(new Double(cb.Tfluegas));//combustion temperature [13]
                    vProp.addElement(new Double(cb.tfig_value.getValue()));//quenching temperature [14]
                    vProp.addElement(new Double(cb.t_diss_value.getValue()));//dissociation rate [15]
                        String calcLambda="calcLambda="+Util.aff_b(cb.JCheckCalcLambda.isSelected());
                    vProp.addElement(calcLambda);//true if calculate lambda set [16]
                        String calcT="calcT="+Util.aff_b(cb.JCheckCalcT.isSelected());
                    vProp.addElement(calcT);//true if calculate T set [17]
                        String setFuelFlow="setFuelFlow="+Util.aff_b(cb.JCheckFuelFlow.isSelected());
```

```
            vProp.addElement(setFuelFlow);//true if calculate with set fuel flow [18]
                String dissoc="setDissoc="+Util.aff_b(cb.Check_dissoc.isSelected());
            vProp.addElement(dissoc);//true if dissociation set [19]
                String premix="setPremix="+Util.aff_b(cb.JCheckPremix.isSelected());
            vProp.addElement(premix);//true if premix set [20]
                String setV="setV="+Util.aff_b(cb.JChecksetV.isSelected());
            vProp.addElement(setV);//true if set volume [21]
                String setP="setP="+Util.aff_b(cb.JChecksetP.isSelected());
            vProp.addElement(setP);//true if set pressure [22]
                String setT="setT="+Util.aff_b(cb.JChecksetT.isSelected());
            vProp.addElement(setT);//true if set temperature [23]
            vProp.addElement(new Double(cb.getUsefulEnergy()));//useful energy [24]
}
        }
     }
     else if(type.equals("node")){
        Node the_node=getNode(nomType);
        if(the_node!=null){
           vProp.addElement(the_node.mainProcess.getName());//main process name
           vProp.addElement(the_node.getClassType());//node type
           vProp.addElement(the_node.getEffNode());//node efficiency
           vProp.addElement(Util.aff_b(the_node.isoBaric));//is node isobaric?
           vProp.addElement(new Integer(the_node.vBranch.size()));//number of branches
for(int j=0;j<the_node.vBranch.size();j++){
   Object[] branch=(Object[])the_node.vBranch.elementAt(j);
              Transfo the_process=(Transfo)branch[0];
              vProp.addElement(the_process.getName());//branch process name
           }
        }
     }
     else if(type.equals("heatEx")){
        HeatExDemo hX=getHX(nomType);

        if(hX!=null){
           String coldFluid="",hotFluid="";
           if(hX instanceof ThermoCoupler){
              ThermoCoupler tc=(ThermoCoupler)hX;
              coldFluid=tc.getHcName();
              hotFluid=tc.getExchange().getName();
           }
           else {
              coldFluid=hX.getColdFluid().getName();
              hotFluid=hX.getHotFluid().getName();
           }
           vProp.addElement(hotFluid);//hot fluid process name
           vProp.addElement(coldFluid);//cold fluid process name
           vProp.addElement(hX.getHxType());//heat exchanger type (counterflow, parallel flow...)
           vProp.addElement(Util.aff_b(hX.JCheckTceImp.isSelected()));//set Tce
           vProp.addElement(Util.aff_b(hX.JCheckTcsImp.isSelected()));//set Tcs
           vProp.addElement(Util.aff_b(hX.JCheckMcImp.isSelected()));//set mc
           vProp.addElement(Util.aff_b(hX.JCheckTfeImp.isSelected()));//set Tfe
           vProp.addElement(Util.aff_b(hX.JCheckTfsImp.isSelected()));//set Tfs
           vProp.addElement(Util.aff_b(hX.JCheckMfImp.isSelected()));//set mf
           vProp.addElement(Util.aff_b(hX.JCheckMinPinch.isSelected()));//set minimum pinch
           vProp.addElement(Util.aff_b(hX.JCheckSetEff.isSelected()));//set efficiency
           vProp.addElement(Util.aff_b(hX.IcalcDirect));//set direct calculation (not used I think)
           vProp.addElement(Util.aff_b(hX.JCheckDesign.isSelected()));//set design mode
           vProp.addElement(new Double(hX.R));//R value
           vProp.addElement(new Double(hX.NUT));//NUT value
           vProp.addElement(new Double(hX.UA));//UA value
```

```
                vProp.addElement(new Double(hX.DTML));//DTML value
                vProp.addElement(new Double(hX.epsi_value.getValue()));//efficiency value
                vProp.addElement(new Double(hX.DTmin_value.getValue()));//pinch value
              }
            }
            return vProp;
}
```

Méthode updatePoint () de Projet

```
          /**
           * met à jour un point, avec recalcul éventuel
           * valable pour calculs humides
           *<p>
           * updates a point, possibly with recalculation
           * valid for moist gas calculations
           */
   public void updatePoint(Vector properties){//profondément modifié en 2018 pour version 2.8
      String nomPoint=(String)properties.elementAt(0);
//       System.out.println("properties.size() "+properties.size());//properties.size()
      PointCorps point=getPoint(nomPoint);
      if(point!=null){
        String test=(String)properties.elementAt(1);
        boolean updateT=Util.lit_b(test);
        String value=(String)properties.elementAt(2);
        double T=Util.lit_d(value);
        test=(String)properties.elementAt(3);
        boolean updateP=Util.lit_b(test);
        value=(String)properties.elementAt(4);
        double P=Util.lit_d(value);
        test=(String)properties.elementAt(5);
        boolean updateX=Util.lit_b(test);
        value=(String)properties.elementAt(6);
        double x=Util.lit_d(value);

       //calculs a effectuer dans le cas general
        if(updateT)point.setT(T);
        if(updateP){
           if(point.open_syst)point.setP(P);
           else {
              point.setV(P);
              point.calcPoint();
           }
        }
        if(updateX)point.setX(x);
        point.CalculeUnPoint();

        //pour melanges humides
        if(properties.size()>7){//modRG 2018
           test=(String)properties.elementAt(7);
           boolean melHum=Util.lit_b(test);
            if(melHum){
                    if(point.lecorps.typeCorps==4){//modRG 01/06
                String task=(String)properties.elementAt(8);
         //           System.out.println("task "+task);
             //      Attention : le fluide chaud ne se refroidit
                value=(String)properties.elementAt(9);
```

```
                if(task.equals("setW and calculate all")){//sets w and calculates moist properties
                   double w=Util.lit_d(value);
                   point.setW(w);
                   point.calcHum();
                }
                if(task.equals("setW and calculate q'")){//sets w and calculates moist properties except t'
                   double w=Util.lit_d(value);
                   point.setW(w);
                   point.calcQprime(false);//modRG 08/06
    //               System.out.println("epsi "+point.epsi_value.getValue()+" q'
"+point.qprime_value.getValue());
                }
                if(task.equals("setEpsi")){//sets epsilon
                   double epsi=Util.lit_d(value);
                   point.setEpsi(epsi);
                   point.impHumRel();
                }
                if(task.equals("setEpsi and calculate")){//sets epsilon and calculates moist properties
                   double epsi=Util.lit_d(value);
                   point.setEpsi(epsi);
                   point.impHumRel();
                   point.calcQprime(false);//modRG 08/06
                }
                if(task.equals("calcWsat")){//calculates saturation properties and moist properties except t'
                   T=Util.lit_d(value);
                   double wsat=point.wsat(T);
                   point.setW(wsat);
                   point.calcQprime(false);//modRG 08/06
                }
                if(task.equals("modHum")){//modifies the gas composition, setting the gas humidity to WPoint
                   point.modGasHum(false);
                }
                if(task.equals("setGasHum")){//sets the gas humidity to w//modRG 01/06
                   double w=Util.lit_d(value);
                   point.setGasHum(w);
                }
             }
             else{//modRG 01/06 (modifier la ressource)
                            String mess="Watch out! point "+point.getName()+"'s substance is not a compound
gas.\nYou cannot use it for wet gas calculations.";
                JOptionPane.showMessageDialog(this,mess);
             }
          }
       }

       if((properties.size()>10)&&(properties.size()<=12)){//mise a jour du facteur de correction d'une pression
imposee
          test=(String)properties.elementAt(10);
          boolean updateCorrFact=Util.lit_b(test);
          if(updateCorrFact){//
             value=(String)properties.elementAt(11);
             double corr=Util.lit_d(value);
             point.pressCorrFact_value.setValue(corr);
          }
       }
       if(properties.size()>12){//mise a jour du sous-refroidissement
          test=(String)properties.elementAt(12);
          boolean updateCorrFact=Util.lit_b(test);
 //          System.out.println("updateCorrFact.P "+test);//properties.size()
          if(updateCorrFact){//
```

```
                value=(String)properties.elementAt(13);
                double corr=Util.lit_d(value);
                point.dTsat_value.setValue(corr);
//                System.out.println("dTsat_value.P "+corr);
            }
        }
    }
}
```

Méthode updateProcess () de Projet

```
    /**
     * met à jour une transfo
     *<p>
     * updates a process
     */
public void updateProcess(Vector properties){
    String nomTransfo=(String)properties.elementAt(0);//getName()
    Transfo trsf=getTransfo(nomTransfo);
    if(trsf!=null){
        String typeTransfo=(String)properties.elementAt(1);//getClassType()
        String test=(String)properties.elementAt(2);
        boolean updateFlow=Util.lit_b(test);
        Double y=(Double)properties.elementAt(3);//getFlow()
        double flowRate=y.doubleValue();
        if(updateFlow)trsf.setFlow(flowRate);
        test=(String)properties.elementAt(4);
        boolean reCalculate=Util.lit_b(test);
        test=(String)properties.elementAt(5);
        trsf.setSetFlow(Util.lit_b(test));
        if(typeTransfo.equals("Compression")){
            Compression ce=(Compression)trsf;
            test=(String)properties.elementAt(6);
            boolean updateEta=Util.lit_b(test);
            y=(Double)properties.elementAt(7);//getIsentropicEfficiency()
            if(updateEta)ce.setIsentropicEfficiency(y.doubleValue());
            if(properties.size()>8){
                test=(String)properties.elementAt(8);
                boolean updatePressRatio=Util.lit_b(test);
                y=(Double)properties.elementAt(9);//getIsentropicEfficiency()
                if(updatePressRatio)ce.setPressureRatio(y.doubleValue());
            }
        }
        if(typeTransfo.equals("Expansion")){
            Expansion ce=(Expansion)trsf;
            test=(String)properties.elementAt(6);
            boolean updateEta=Util.lit_b(test);
            y=(Double)properties.elementAt(7);//getIsentropicEfficiency()
            if(updateEta)ce.setIsentropicEfficiency(y.doubleValue());
            if(properties.size()>8){
                test=(String)properties.elementAt(8);
                boolean updatePressRatio=Util.lit_b(test);
                y=(Double)properties.elementAt(9);//getIsentropicEfficiency()
                if(updatePressRatio)ce.setPressureRatio(y.doubleValue());
            }
        }
        if(typeTransfo.equals("Combustion")){
            Combustion ce=(Combustion)trsf;
            test=(String)properties.elementAt(6);
            boolean updateTout=Util.lit_b(test);
```

```
            y=(Double)properties.elementAt(7);//getOutletTemperature()
            if(updateTout)ce.setOutletTemperature(y.doubleValue());
            if(properties.size()>8){
               test=(String)properties.elementAt(8);
               boolean updateLambda=Util.lit_b(test);
               y=(Double)properties.elementAt(9);//
               if(updateLambda)ce.setLambda(y.doubleValue());
            }
             if(properties.size()>10){
               test=(String)properties.elementAt(10);
               boolean updateEta=Util.lit_b(test);
               y=(Double)properties.elementAt(11);//
               if(updateEta)ce.setThermalEfficiency(y.doubleValue());
            }
        }
         if(reCalculate)trsf.Calculate();
       }
   }
```

Méthode updateNode () de Projet

```
          /**
           * met à jour un noeud
           *<p>
           * updates a node
           */
public void updateNode(Vector properties){//modRG 2018
   String name=(String)properties.elementAt(0);//getName()
   Node nd=getNode(name);
   String test=(String)properties.elementAt(1);
   boolean reCalculate=Util.lit_b(test);
   if(nd instanceof Separator){//on passe la valeur true or false en 3ème argument, et la valeur de l'efficacité de
séchage en 4ème
      Separator sp=(Separator)nd;
      test=(String)properties.elementAt(2);
      boolean updateEpsi=Util.lit_b(test);
      Double y=(Double)properties.elementAt(3);
      double epsi=y.doubleValue();
      if(updateEpsi)sp.setEfficiency(epsi);
   }
   if(nd instanceof Divider){//modifié en 2018  //modRG 2018
           //on passe la valeur true or false pour updateFactor en 3ème argument,
           //le nom de la transfo dont on souhaite changer le facteur de débit en 4ème
           //et la valeur du facteur de débit en 5ème
           Divider div=(Divider)nd;
           if(properties.size()==5){
                   test=(String)properties.elementAt(2);
                   boolean updateFactor=Util.lit_b(test);
                if(updateFactor){
                   reCalculate=false;//le diviseur ne doit alors être recalculé que quand tous les facteurs de débit
ont été mis à jour
                   String branchName=(String)properties.elementAt(3);//le recalcul est effectué par un appel
sans mise à jour des facteurs de débit
                   String flowFactor=(String)properties.elementAt(4);
                   div.setupFlowFactor(branchName, flowFactor);
                }
           }
   }
   if(reCalculate)nd.Calculate();
}
```

Méthode updateHX () de Projet

```
        /**
         * met à jour un échangeur de chaleur
         *<p>
         * updates a heat exchanger
         */
  public void updateHX(Vector properties){
     String name=(String)properties.elementAt(0);//getName()
     HeatExDemo hx=getHX(name);
     String test=(String)properties.elementAt(1);
     boolean reCalculate=Util.lit_b(test);
     test=(String)properties.elementAt(2);
     boolean updateUA=Util.lit_b(test);
     Double y=(Double)properties.elementAt(3);
     double UA=y.doubleValue();
     if(updateUA)hx.setUA(UA);
     test=(String)properties.elementAt(4);
     boolean updateEpsi=Util.lit_b(test);
     y=(Double)properties.elementAt(5);
     if(updateEpsi)hx.setEpsi(y.doubleValue());
     test=(String)properties.elementAt(6);
     boolean updateDTmin=Util.lit_b(test);
     y=(Double)properties.elementAt(7);
     if(updateDTmin)hx.setDTmin(y.doubleValue());
     test=(String)properties.elementAt(8);
     boolean updateCalcMode=Util.lit_b(test);
     test=(String)properties.elementAt(9);
     boolean calcMode=Util.lit_b(test);
     if(updateCalcMode)hx.setCalcMode(calcMode);
     if(reCalculate)hx.Calculate();
}
```

Méthode updateSetPressure() de Projet

```
        /**
         * met à jour une pression imposée et recalcule la pression des points
         * les facteurs de correction de ces derniers sont modifiables par updatePoint()
         *<p>
         * updates a set pressure and recalculates the point pressures
         * their correction factors can be updated by updatePoint()
         */
  public void updateSetPressure(Vector properties){//modRG 2008
      String nomPres=(String)properties.elementAt(0);//getName()
      PresSetValue thePres=getPresSet(nomPres);
      if(thePres!=null){//le false signifie que la pression est changée, mais sans notification, pour éviter un
recalcul complet
        Double y=(Double)properties.elementAt(1);
        double value=y.doubleValue();
        thePres.setValue(value);
        thePres.modifyPressures(false);
       }
   }
```

Méthode getSubstProperties() de Corps

```
/**
   * donne les fonctions d'état du corps
   * <p>
```

```
   * gives the substance state functions
   * @return Vector
         *<p>
         * do not override
   */
   public Vector getSubstProperties(){
            Vector vProp=new Vector();
            vProp.addElement(new Double(T));//Temperature [0]
            vProp.addElement(new Double(P));//Pressure [1]
            vProp.addElement(new Double(xx));//Quality [2]
            vProp.addElement(new Double(V));//Volume [3]
            vProp.addElement(new Double(U));//Internal energy [4]
            vProp.addElement(new Double(H));//Enthalpy [5]
            vProp.addElement(new Double(S));//Entropy [6]
            vProp.addElement(new Double(M));//Molar mass [7]
            vProp.addElement(new Integer(typeCorps));//Substance type [8]
            vProp.addElement(new Double(M_sec));//Dry gas molar mass [9]
            vProp.addElement(new Double(TC));//Critical temperature [10]
            vProp.addElement(new Double(PC));//Critical pressure [11]
            vProp.addElement(new Double(VC));//Critical volume [12]
            vProp.addElement(new Double(getChemicalExergy()));//Chemical exergy [13]
            return vProp;
   }
```

Méthode getExternalClassInstances () de Projet

```
/**
   * Provides the external class instances
   *
   * @return a Vector containing the instances and their type
   */
public Vector getExternalClassInstances(){
      Vector vInstances=new Vector();
            Primtype pt;
            int nbProcess=vProcess.size();
            for(int i=0;i<nbProcess;i++){
               pt=(Primtype)vProcess.elementAt(i);
               if(pt instanceof TransfoExterne){
                  Object[] obj=new Object[6];
                  TransfoExterne te=(TransfoExterne)pt;
                  obj[0]="process";
                  obj[1]=te.cType.externalInstance;//instance de la classe externe
                  obj[2]=te.getName();
                  obj[3]=te.cType.externalInstance.getType();//type de la classe externe
                  obj[4]=te.getPointAmont().getName();
                  obj[5]=te.getPointAval().getName();
                  vInstances.addElement(obj);
               }
                  }
            int nbNode=vNode.size();
            for(int i=0;i<nbNode;i++){
               pt=(Primtype)vNode.elementAt(i);
               if((pt instanceof MixerExterne)||(pt instanceof DividerExterne)){
                  Node nd=(Node)pt;
                  Object[] obj=new Object[6];
                  obj[0]="node";
                  obj[1]=nd.cType.externalInstance;//instance de la classe externe
                  obj[2]=nd.getName();
                  obj[3]=nd.cType.externalInstance.getType();//type de la classe externe
                  obj[4]=nd.getMainProcess().getName();
```

```
                obj[5]=new Integer(nd.getBranches().size());
                vInstances.addElement(obj);
            }
              }
              return vInstances;
  }
```

Méthode setupChart () de Projet

```
public void setupChart(Vector properties){
    String task=(String)properties.elementAt(0);
    String value=(String)properties.elementAt(1);

    if(task.equals("openDiag")){
       int i=Util.lit_i(value);
       setupChart(false);
       cm.setupChart(i);
    }
    else if(task.equals("selectSubstance")&& cm!=null){
       Graph graph=cm.getChart();
       graph.getDiagIni().setSelectedSubstance(value);
       graph.select_vap();
    }
    else if(task.equals("readCycle")&& cm!=null){
       Graph graph=cm.getChart();
       graph.litCycle(value,"");
       graph.setConnectedCycle();
    }
    else if(task.equals("unSelectCycle")&& cm!=null){
       Graph graph=cm.getChart();
       graph.setupCycleManager();
       CycleManager cM=graph.getCycleManager();
       cM.unSelect(value);
       graph.repaint();
    }
    else if(task.equals("removeCycle")&& cm!=null){
       Graph graph=cm.getChart();
       graph.setupCycleManager();
       CycleManager cM=graph.getCycleManager();
       cM.removeCycle(value);
       graph.repaint();
    }
    else if(task.equals("setChartType")&& cm!=null){
       Graph graph=cm.getChart();
       graph.setChartType(value);
    }

  }
```

ANNEXE 3 : METHODES UTILITAIRES DU PAQUET EXTTHOPT

Classe Util

La classe extThopt.Util fournit un certain nombre de méthodes utilitaires pour faciliter la programmation des classes externes :

Méthodes d'affichage

- public static String aff_i(int i) : affiche un entier
- public static String aff_b(boolean b) : affiche un booléen
- public static String aff_d(double d) : affiche un double
- public static String aff_d(double d, int dec) : affiche un double avec dec décimales

Méthodes de lecture

- public static int lit_i(String s) : lit un entier
- public static boolean lit_b(String s) : lit un booléen
- public static double lit_d(String s) : lit un double

Méthodes d'extraction de valeurs d'une chaîne alphanumérique

- public static String extr_value(String s)
- public static String extr_value(String ligne_data, String search)

La première méthode extrait "3.4" de "value=3.4", quel que soit le texte à gauche du signe d'égalité. Elle peut être utilisée avec un StringTokeniser séparant les couples "valeur=xxx".

La seconde combine un StringTokeniser avec comme séparateur la tabulation et la méthode précédente, cherchant le couple "search=xxx".

Ces deux méthodes retournent null en cas d'insuccès.

Méthodes d'inversion de fonction

- public static double dicho_T (Inversable inv, double value,double param, String function, double valMin, double valMax,double epsilon)

Cette méthodee générique permet l'inversion d'une fonction par dichotomie. La classe doit implémenter l'interface Inversable, qui impose de définir la méthode f_dicho :

public double f_dicho(double x, double param, String function)

value est la valeur objectif pour la méthode retournée par f-dicho, la variable x varie entre valMin et valMax, epsilon est le critère de précision, et function est un String permettant de repérer une méthode particulière dans f_dicho :

if (function.equals("P_LiBr"))return getP(x,T);

Au bout de 50 itérations, s'il n'y a pas convergence, dicho_T renvoie 0.

Méthodes de manipulation d'un gaz pur dans un Vector de composition de gaz

- public static double molarComp(Vector vComp,String pureGas) : renvoie la fraction molaire de pureGas
- public static boolean contains(Vector vComp,String pureGas) : renvoie true si pureGas est présent
- public static void updateMolarComp(Vector vComp,String pureGas, double newFractMol) : met à jour la fraction molaire de pureGas

Classe PointThopt

Cette classe permet de créer dans une classe externe des sortes de clones des points du noyau de Thermoptim, afin d'avoir un accès aisé à leurs valeurs, qui ne sont pas directement accessibles. Elle apporte davantage de confort et de lisibilité que ne le fait l'utilisation des méthodes getProperties() et updatePoint() de Projet,

documentées dans le tome 3 manuel de référence. Elle possède des double dans lesquels sont stockés les équivalents des variables d'état du noyau :

```
double W,Epsi,QPrime,Tprime,Tr,VPrime,Cond, M_sec, corrFactor;
 CorpsThopt corps;
```

Lors de la construction, on spécifie la référence au projet et le nom du point dans le simulateur, qui permet à Thermoptim de savoir quel point est concerné. Attention, s'il y a une erreur à ce niveau, le point ne peut être correctement instancié.

```
public PointThopt(Projet proj, String name){
            this.proj=proj;
            this.name=name;
            try{
               getProperties();
            }
            catch(Exception e){
               String msg = "Error constructing the PointCorps name: "+ name;
               JOptionPane.showMessageDialog(new JFrame(), msg);
               e.printStackTrace();
            }
            corps=new CorpsThopt(proj, nomCorps,lecorps);
}
```

PointThopt propose lui aussi des méthodes de mise à jour de son équivalent du simulateur, fonctionnant sur le même principe que celles de ExtPilot, avec des signatures variables selon le nombre de valeurs à modifier.

```
   public void update(boolean updateT, boolean updateP, boolean updateX, boolean updateCorrFactor, boolean
melHum, String task, double value)
```

Une version allégée la plus courante a servi d'exemple à l'utilisation des booléens explicites d'ExtPilot :

```
   public void update(boolean updateT, boolean updateP, boolean updateX){
      update(updateT, updateP, updateX, false, false, "", 0.);
   }
```

Classe CorpsThopt

Cette classe permet de créer dans une classe externe des sortes de clones des corps du noyau de Thermoptim, afin d'avoir un accès aisé à leurs valeurs, qui ne sont pas directement accessibles. Elle apporte davantage de confort et de lisibilité que ne le fait l'utilisation des méthodes getProperties() et updateSubstance() de Projet, documentées dans le tome 3 manuel de référence. Elle possède des double dans lesquels sont stockés les équivalents des variables d'état du noyau :

```
public double T,P,X,V,U,H,S,M,M_sec,TC,PC,VC;
```

Lors de la construction, on spécifie la référence au projet et le nom du point dans le simulateur, qui permet à Thermoptim de savoir quel point est concerné. Attention, s'il y a une erreur à ce niveau, le point ne peut être correctement instancié.

```
         public CorpsThopt(Projet proj, String name, Corps lecorps){
            this.proj=proj;
            this.name=name;
            this.lecorps=lecorps;
         }
```

Un exemple d'instanciation est donné dans le constructeur de PointThopt.

La méthode **getSubstProperties**() permet de mettre à jour les valeurs après un calcul effectué sur la variable lecorps. L'exemple ci-dessous montre l'utilisation d'un CorpsThopt faisant appel au PointThopt amont :

```
CorpsThopt corps=new CorpsThopt(proj,amont.nomCorps, amont.lecorps);
corps.lecorps.CalcPropCorps(amont.T,amont.P,1.);
corps.getSubstProperties();
```

Les fonctions usuelles sont alors directemet accessibles : par exemple, l'enthalpie vaut corps.H.

ANNEXE 4 : TEP THERMOSOFT - INTERFACE JAVA / DELPHI – APPLICATION A THERMOPTIM (PAR F. RIVOLLET)

Ce document explique le principe de passerelle entre TEP ThermoSoft et Thermoptim. Ces deux programmes sont écrits respectivement en Pascal sous environnement Delphi et en Java. L'objectif est le calcul des propriétés thermodynamiques nécessaires sous Thermoptim à partir des modèles développés pour TEP ThermoSoft.

Structure de dialogue entre les deux programmes

Le schéma ci-dessous reprend le principe de dialogue entre les deux programmes et les fichiers nécessaire à un calcul.

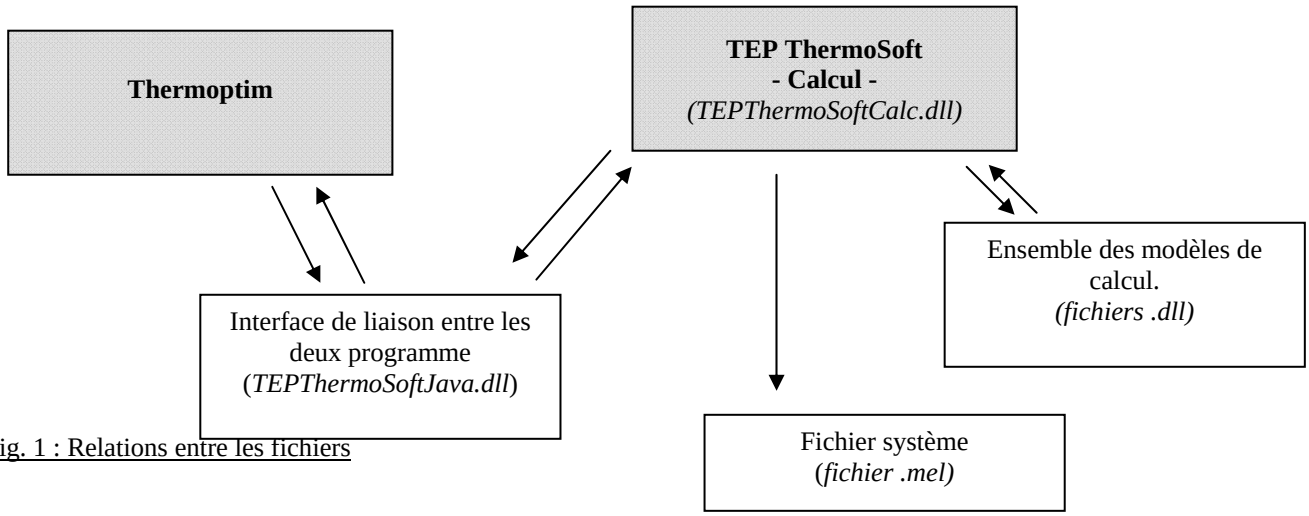


Fig. 1 : Relations entre les fichiers

La principale difficulté est la compatibilité des fonctions appelées du Java vers le Pascal. Pour ce faire, une librairie spécifique (*TEPThermoSoftJava.dll*) permet d'assurer le passage des variables entre les deux entités.

La définition et l'exécution des calculs au sein de TEP ThermoSoft nécessitent :

- La routine principale (*TEPThermoSoftCalc.dll*)
- un fichier système (*fichier .mel*) qui rassemble l'ensemble des propriétés de calcul.
- les modèles thermodynamiques écrits sous forme de librairies (*.dll*) et contenus dans un répertoire spécifique.

Remarque : Les deux fichiers TEPThermoSoftCalc.dll et TEPThermoSoftJava.dll doivent se situer dans le même répertoire.

Exécuter un calcul en java

Définition des méthodes de dialogue avec TEP ThermoSoft

Les méthodes suivantes doivent être définies en Java comme faisant référence à la librairie externe « Delphi2Java.dll ». Leur fonction et leur syntaxe sont expliquées dans les paragraphes 2.2 à 2.5 et un exemple est donné en 2.6.

```
public native int InitSession(String RepMODELES);
public native void FermerSession();
public native int ChargerSysteme (String FichierMEL);
public native double Lire(String Symbole);
public native void Ecrire(String Symbole, double valeur);
```

public native void Calculer(String code);

Chargement/Libération des modèles thermodynamiques en mémoire

Avant tout calcul, la méthode « InitSession » doit être appelée afin d'initialiser l'ensemble des paramètres et de spécifier le répertoire contenant les modèles thermodynamiques « .dll ».

InitSession(System.getProperty("user.dir")+File.separator+"TEPThermoSoft_DLL"+File.separator);

Cette fonction retourne un entier qui détermine un code d'état :

0	Pas d'erreur
-1	Erreur inconnue

Une fois que les fonctions de calcul ne sont plus nécessaires, il est souhaitable de libérer la mémoire en appelant « FermerSession »

FermerSession() ;

Une utilisation répétée de ces deux méthodes peut ralentir les calculs. Il est ainsi préférable de regrouper les calculs quand cela est possible.

Définition d'un système

Un « système » représente soit un corps pur, soit un mélange de plusieurs composés. Toutes les informations relatives au calcul de ses propriétés (Modèles thermodynamiques, valeurs des paramètres de calcul, …), sont regroupées dans un fichier unique dont l'extension est « .mel ». Ce fichier peut être ouvert avec un simple éditeur de texte. Sa structure reprend celle d'un fichier « .ini » sous Windows. Cela signifie que des catégories sont définies entre crochets :

```
[GENERAL]
NOM = Melange
DATE = 25/04/2005
HEURE = 08:17:56
```

Propriétés générales sur le système

```
[CPS]
994    AMMONIA
1000   WATER
```

Informations sur les composés

```
[ELV]
S      CAlpha  AlphaMC.dll
S      CCPGP CPGP107.dll
// … //
P      a(1)
P      a(2)
P      ALPHA(1)
// .. //
P      CP107(2)[0] 33363
P      CP107(2)[1] 26790
P      P           Pa
P      Pc(1) 11280005.625      Pa
P      Pc(2) 22055007.45 Pa
P      PHI|l(1)
// .. //
```

Modèles thermodynamiques à utiliser

Symboles et Valeurs des paramètres de calcul

Au sein de TEP ThermoSoft, tout calcul se réfère à un système donné. Pour cela, il faut, à partir de l'application Java, charger les paramètres de calcul en mémoire à l'aide de la fonction « *ChargerSysteme* » en spécifiant le fichier système voulu :

ChargerCalcul(System.getProperty("user.dir")+File.separator+"mixtures"+File.separator+"TEPThermoSoft_MEL"+File.separator+selectedFile);

Cette méthode retourne un entier qui spécifie le numéro du mélange chargé en mémoire.

Ce numéro sera utile lors de l'utilisation de plusieurs mélanges en parallèle. Dans un premier temps un seul mélange sera pris en considération.

De plus, ce numéro, s'il est négatif, indique une erreur lors du chargement du fichier. Pour le moment les codes d'erreur sont les suivants :

-1 Erreur inconnue
-2 Fichier introuvable
-3 Aucun composé défini dans le fichier.

Modifier / Lire des variables de TEP ThermoSoft

Une fois un fichier de mélange chargé en mémoire, deux méthodes ont été écrites pour permettre de modifier et de lire les valeurs des variables se référant au système. Toute variable intervenant dans les calculs peut être modifiée ou lue. Pour cela il suffit de respecter la typographie suivante :

SYMB(cps)[vecteur]|phase

SYMB	symbole de la variable à modifier (*Ex. T : Température – Cf. 0*)
cps	composé ou interaction entre composés (Ex. *(1) , (2), (1_2)*). La numérotation des composés commence à 1.
vecteur	numéro du vecteur (ex. *[0], [1],* …). La numérotation des vecteurs commence à 0.
phase	lors de données multiphasiques, la barre | suivi d'une lettre (l ou v) permet de définir la phase recherchée (ex. |l ou |v).

Par exemple, la lecture de la température peut se faire par la méthode :

double temperature = Lire("T");

De même la modification du premier paramètre de la fonction Alpha de Mathias-Copeman pour le composé 2 s'écrit :

Ecrire("MC(2)[0]", 0.152);

Lancer un calcul

Pour lancer un calcul, il suffit d'écrire la valeur des paramètres de calcul au moyen de la méthode « Ecrire » vue précédemment, puis de lancer le calcul par la méthode « Calculer » :

Calculer("Tx");

La méthode « Calculer » attend un paramètre qui définit le type de calcul souhaité (cf. 0) Dans l'exemple ci-dessus, il s'agit d'un calcul d'équilibre LV en mélange en spécifiant la température et la composition liquide. Ainsi, avant d'exécuter cette fonction il est nécessaire de bien définir les variables T et x comme par exemple pour un binaire :

Ecrire("T",280);
Ecrire("x(1)",0.1);

Ecrire("x(2)",0.9);

Exemple d'écriture d'un calcul complet

Voici un exemple de lignes de commande permettant un calcul à l'équilibre LV du système « NH3-H2O» défini dans le fichier système « NH3-H2O.mel » :

```
InitSession(System.getProperty("user.dir")+File.separator+"TEPThermoSoft_DLL"+File.separator);
//
ChargerSysteme(System.getProperty("user.dir")+File.separator+"mixtures"+File.separator+"TEPThermoSoft_MEL"+File.separator+"NH3-H2O.mel");
//
Ecrire("T",280);
Ecrire("x(1)",0.1);
Ecrire("x(2)",0.9);
//
Calculer("Tx");
//
P  = Lire("P");
y[0]  = Lire("y(1)");
y[1]  = Lire("y(2)");
//
FermerSession() ;
```

Variables et méthodes de calcul disponibles

Variables classiques

Voici une liste des symboles les plus utilisés dans les méthodes « Lire » et « Ecrire ».

Symbole	***Unité***	***Description***
T	K	Température
P	Pa	Pression
x(i)		Composition liquide du composé « i »
y(i)		Composition gaz du composé « i »
z(i)		Composition globale du composé « i »
h, h|l, h|v	J/mol	Enthalpie
s, s|l, s|v	J/mol/K	
TauVap		Taux de vaporisation (= -1 lors d'un calcul ELV Px, Tx, Ty ou Py).
h0	J/mol	Enthalpie de référence.
s0	J/mol	Entropie de référence
Mw(i)	kg/mol	Masse molaire du composé « i »

Méthodes de calcul

Voici une liste des symboles des méthodes disponibles (elle sera complétée au fur et à mesure des besoins).

Symbole	***Type de calcul***[7]	***Description***
Tx	MEL	Calcul à l'équilibre LV en spécifiant la température et la composition liquide.
Ty	MEL	Calcul à l'équilibre LV en spécifiant la température et la composition gaz.
Px	MEL	Calcul à l'équilibre LV en spécifiant la pression et la composition liquide.
Py	MEL	Calcul à l'équilibre LV en spécifiant la pression et la composition gaz.
PTz	MEL	Calcul en spécifiant la composition globale, la pression et la température (données à l'équilibre ou hors équilibre LV).

Pour le moment la méthode PTz semble pouvoir suffire car valable en corps purs et en mélange (suivant les valeurs de z). De plus elle est continue à et hors ELV. Ainsi une méthode numérique simple devrait permettre d'estimer les valeurs à h et s constant.

[7] *MEL concerne les mélanges et CP, les corps purs.*

ANNEXE 5 : CALCULS DES GAZ HUMIDES DANS THERMOPTIM

Introduction

On peut représenter un gaz humide dans Thermoptim de deux manières équivalentes : soit directement comme un corps composé comprenant au moins deux constituants : H_2O et un autre gaz, pur ou composé, soit comme un gaz sec dont on connaît l'humidité spécifique. La première manière présente l'avantage que la composition du gaz humide est accessible à tout moment. En revanche, elle implique que, pour un même gaz sec, on crée un nouveau corps humide pour chaque valeur de l'humidité. La seconde représentation est quant à elle beaucoup plus concise, étant donné qu'elle ne fait appel qu'au gaz invariant et à la valeur de l'humidité. C'est pourquoi c'est celle qui est utilisée par les fonctions de calcul des gaz humides, alors que c'est la première qui est de règle dans l'environnement de calcul standard des cycles de Thermoptim.

Rappelons que l'on appelle **humidité relative ε** le rapport de la pression partielle de la vapeur d'eau à sa pression de vapeur saturante, et que, par définition, l'indice gs correspondant aux valeurs relatives au gaz sec, **l'humidité absolue ou spécifique w** est égale au rapport de la masse d'eau contenue dans un volume donné de mélange humide à la masse de gaz sec contenue dans ce même volume, soit :

$$w = \frac{y_{H_2O}}{y_{gs}} = \frac{M_{H_2O} x_{H_2O}}{M_{gs} x_{gs}} = \frac{18\, x_{H2O}}{M_{gs}\,(1 - x_{H2O})}$$

Cette relation permet de calculer w lorsqu'on connaît la composition du gaz humide.

Méthodes disponibles dans les classes externes

Généralement, les calculs de gaz humides sont effectués (à partir des classes externes) au niveau d'un point, mais une méthode permet de directement modifier l'humidité d'un gaz. L'humidité du gaz est entrée en indiquant soit son humidité absolue w, soit son humidité relative ε. Les deux méthodes employées sont définies dans la classe ExtProcess, dont dérivent toutes les transfos et tous les nœuds externes.

La méthode **updatepoint("nomPoint",…)** permet d'effectuer les calculs humides tandis que la méthode **getPointProperties("nomPoint")** récupère, en sus des propriétés standard (P,T,h,s…) les valeurs des propriétés humides d'un point par les grandeurs suivantes : Wpoint pour l'humidité absolue w, Epsipoint pour l'humidité relative ε, Qprimepoint pour l'enthalpie spécifique q', Tprimepoint pour la température adiabatique t' (en °C), Trpoint pour la température de rosée tr (en °C), VPrimepoint pour le volume spécifique vs, Condpoint pour les condensats, et M_secpoint pour la masse molaire du gaz sec.

Recherche de l'humidité d'un point

Lorsque l'état du point a été sauvé, w est connu, et la méthode **getPointproperties("nomPoint")** permet d'y accéder directement.

Lorsque la composition du gaz est déterminée par programmation, il peut être nécessaire de recalculer w, ce qui peut être fait par la formule ci-dessous :

//calcul de l'humidité absolue du gaz

*double inlet_w=18.01528*fractH2OFuel/fuelM/(1-fractH2OFuel);// w =* $\frac{M_{H_2O} x_{H_2O}}{M_{gs} x_{gs}}$

Mise à jour des propriétés humides d'un point

La méthode **updatepoint("nomPoint",…)** est une méthode généique de mise à jour et de recalcul des variables d'état d'un point, qui a été généralisée pour permettre les calculs humides. Son code est donné ci-dessous :

```
public void updatepoint(String name, boolean updateT, double T,
   boolean updateP, double P, boolean updateX, double x,
   boolean melHum, String task, double value){
         String[] args=new String[2];
    Vector vPoint=new Vector();
    vPoint.addElement(name);
    vPoint.addElement(Util.aff_b(updateT));
    vPoint.addElement(Util.aff_d(T));
    vPoint.addElement(Util.aff_b(updateP));
    vPoint.addElement(Util.aff_d(P));
    vPoint.addElement(Util.aff_b(updateX));
    vPoint.addElement(Util.aff_d(x));
    vPoint.addElement(Util.aff_b(melHum));
    vPoint.addElement(task);
    vPoint.addElement(Util.aff_d(value));
    proj.updatePoint(vPoint);

         }
```

Comme le montre ce code, elle construit un Vector puis passe le relais à la méthode **updatePoint()** de Projet.

Si le booléen melHum vaut "false", la mise à jour du point concerne T, P ou x, selon que les booléens updateT, updateP et updateX valent "true" ou "false" : il s'agit d'une mise à jour standard sans calcul des propriétés humides.

Si le booléen melHum vaut "true", seuls les calculs humides sont effectués, même si updateT, updateP et updateX valent "true".

Ces calculs sont définis par les deux grandeurs **task** et **value**.

task est une chaîne de caractère précisant le type de calculs à effectuer, et **value** un double fournissant la valeur de la grandeur à modifier.

1) Calculs sans modification de la composition du gaz

Les calculs étant effectués par rapport au gaz sec, la composition du gaz n'est pas modifiée.

Imposer l'humidité spécifique w

Si task ="setW and calculate all", Thermoptim impose w (passé dans value) et calcule toutes les propriétés humides

Lorsque la température d'un point est élevée, des problèmes de convergence peuvent survenir pour le calcul de la température humide t'. Pour contourner cette difficulté, le paramétrage ci-dessous ne calcule que l'enthalpie spécifique

Si task ="setW and calculate q'", Thermoptim impose w (passé dans value) et calcule toutes les propriétés humides sauf t'

Si task ="calcWsat", Thermoptim calcule wsat et toutes les propriétés humides à la saturation sauf t'

Imposer l'humidité relative ε

Si task ="setEpsi", Thermoptim impose ε (passé dans value)

Si task ="setEpsi and calculate", Thermoptim impose ε (passé dans value) et calcule toutes les propriétés humides sauf t'

2) Modification de la composition du gaz

2.1 en opérant indirectement à partir d'un point

Lorsqu'on désire modifier la composition d'un gaz, la méthode updatePoint() permet de le faire avec les paramétrages suivants :

Si task ="modHum", Thermoptim modifie la composition du gaz pour que son humidité soit égale à Wpoint (il n'y a alors pas besoin de passer de valeur dans value).

Si task ="setGasHum", Thermoptim modifie la composition du gaz pour que son humidité soit égale à w (passé dans value)

2.2 en opérant directement sur le gaz

Il est aussi possible de modifier l'humidité d'un gaz indépendamment de l'état d'un point, en utilisant la méthode **updateGasComp()** de GazIdeal, accessible par **public void updateGasComposition(Vector vComp)** de Corps : si le premier élément de vComp est un Integer d'une valeur négative, un traitement particulier est effectué. L'humidité absolue passée en troisième élément de vComp est imposée au gaz.

```
else{//modifications gaz humides
   String task=(String)vComp.elementAt(1);
   String value=(String)vComp.elementAt(2);
   if(task.equals("setGasHum")){//sets the gas humidity
      double w=Util.lit_d(value);
      setGasHum(w);
   }
}
```

L'exemple ci-dessous, issu de la classe BiomassCombustion, montre comment modifier la composition d'un gaz sec pour qu'elle corresponde au gaz humide dont la fraction molaire en eau est fractH2Ofuel :

```
//mise en forme du Vector
 Vector vComp=new Vector();
  vComp.addElement(new Integer(-1));
  vComp.addElement("setGasHum");
  vComp.addElement(Util.aff_d(inlet_w));
 //modification de la composition du gaz
  NewFuelSubstance.updateGasComposition(vComp);
```

Affichage en unités spécifiques

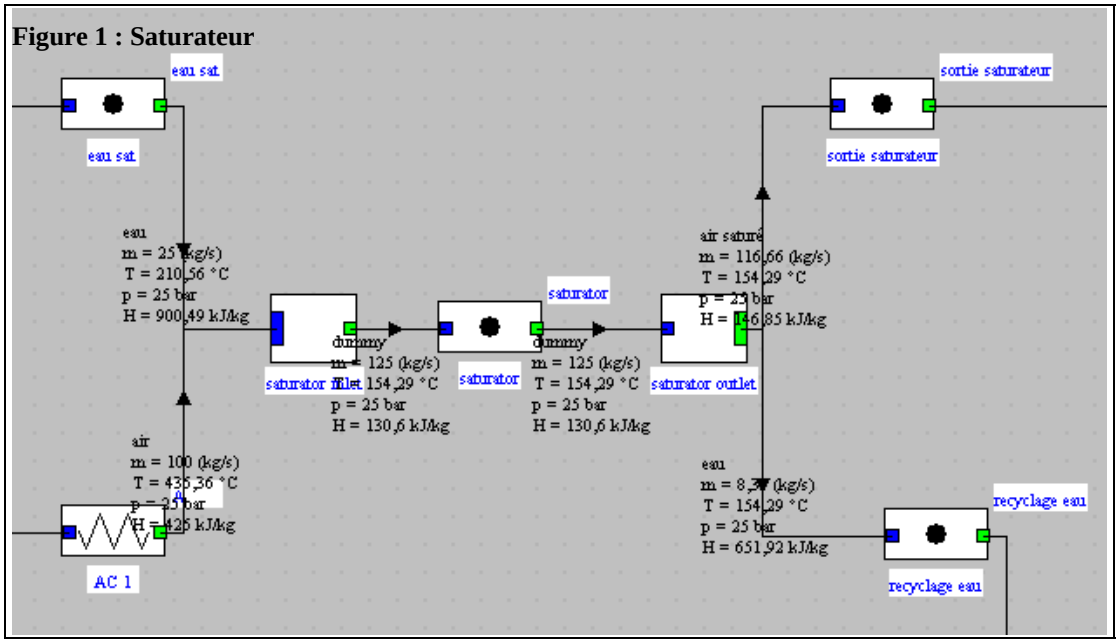
Afin de donner la possibilité de changer de système de référence, une option d'affichage en unités spécifiques a été rajoutée dans l'écran des paramètres globaux de Thermoptim, comme indiqué dans le tome 1 du manuel de référence. Pour les gaz humides, le débit affiché est celui du gaz sec, qui est invariant, et l'enthalpie H est remplacée par l'enthalpie spécifique q'.

Exemple d'utilisation : modèle de saturateur

Dans une turbine à gaz à air humide, on augmente la puissance de la machine et l'efficacité du cycle en humidifiant dans un saturateur l'air comprimé avant entrée dans la chambre de combustion. Le cycle est assez complexe afin d'optimiser la récupération de chaleur, mais le modèle de saturateur se présente de manière relativement simple (figure 1) : de l'eau à une température de l'ordre de 280 °C entre dans le saturateur, où elle est mise en contact avec un flux d'air chaud (200 °C) et relativement sec sortant du compresseur. Une partie de

l'eau est vaporisée et sert à augmenter l'humidité de l'air, qui sort proche de l'état saturé. Le reliquat d'eau est recyclé. On suppose ici que le saturateur est adiabatique et que l'eau et l'air sortent à la même température.

Figure 1 : Saturateur



Le code de la classe se présente comme suit :

1) on commence par chercher la composition du gaz entrant, et on met à jour la composition du gaz en sortie, précaution au cas où les deux gaz secs seraient différents

```
//Propriétés humides du gaz entrant
args[0]="process";//type of the element (see method getProperties(String[] args))
args[1]=wq.gasProcess;//name of the process (see method getProperties(String[] args))
Vector vProp=proj.getProperties(args);
String amont=(String)vProp.elementAt(2);//gets the downstream point name
getPointProperties(amont);//direct parsing of point property vector
getSubstProperties(nomCorps);
double M=Msubst;
Vector vComp=lecorps.getGasComposition();
outletGasSubstance.updateGasComposition(vComp);//on met à jour la composition du gaz en sortie
//cette ligne de code permet d'utiliser le saturateur avec n'importe quel gaz
```

2) on calcule l'humidité absolue en entrée par la formule de définition, pour éviter, compte tenu de la température élevée du gaz, d'avoir à estimer les conditions de saturation

```
//on calcule w inlet par la formule de définition, pour éviter, compte tenu de la
//température élevée du gaz, d'avoir à estimer les conditions de saturation
double fractH2O=Util.molarComp(vComp,"H2O");//fraction molaire de H2O
double inlet_w=fractH2O*18.01528/M_secpoint/(1-fractH2O);//(M_H2O)(x_H2O)/(M_gs)/(x_gs)
getPointProperties(amont);
double inletT=Tpoint;
```

3) on détermine le débit de gaz sec et l'enthalpie spécifique du gaz en entrée, en utilisant les deux méthodes updatepoint() et getPointProperties()

```
//estimation du débit d'air sec
Double f=(Double)vProp.elementAt(3);
double flow=f.doubleValue();//débit massique de gaz humide
flow_as=flow/(1+inlet_w);//débit massique de gaz sec

updatepoint(wq.gasPoint, false, 0,  //T
    false, 0, false, 0,//P,x
    true, "setW and calculate q'", inlet_w);
getPointProperties(wq.gasPoint);
qPrimeAmont=QPrimepoint;//enthalpie spécifique de l'air entrant dans le saturateur
```

4) A ce stade, les conditions humides du gaz amont sont parfaitement calculées. Il faut maintenant déterminer la température de sortie du saturateur Ts, en résolvant simultanément :

- le bilan hydrique (le débit d'eau consommé est égal au produit du débit de gaz sec par la variation d'humidité du gaz)
- le bilan enthalpique (la somme des débits d'enthalpie entrants (en unités spécifiques pour le gaz humide) est égal à la somme des débits d'enthapie sortants.

Etant donné que Ts est inconnue, on fait une recherche de solution par dichotomie, en utilisant la fonction générique Util.dicho_T(), qui fait appel à f_dicho(). Le code est donné ci-dessous :

- on fait passer l'humidité d'entrée en argument, à la place de la pression, connue par ailleurs
- on commence par calculer l'enthalpie de l'eau en sortie heau(Ts)
- on modifie la température du gaz en sortie, puis son humidité, à partir de la valeur lue à l'écran, et on récupère les valeurs de son humidité absolue et de son enthalpie spécifique
- on calcule le débit d'eau restant (il faudrait pour bien faire effectuer un test pour vérifier qu'il reste positif)
- on écrit que l'enthalpie perdue par l'eau se retrouve dans l'air, et on calcule le résidu diff
- la température Ts est déterminée lorsque diff=0.

```
//       double T=Util.dicho_T(this, 0, inlet_w, "saturator", 373.15, 450., 0.01);
      if (fonc.equals("saturator")){
          double diff;
          double w_dicho=P;
          //enthalpie de l'eau en sortie
          updatepoint(waterPoint, true, T,  //T
              false, 0, false, 0,//P,x
              false, "", 0);
          getPointProperties(waterPoint);//état de l'eau en sortie
          double hEau=Hpoint;

          //enthalpie spécifique et humidité spécifique de l'air en sortie
          updatepoint(gasPoint, true, T,  //T
              false, 0, false, 0,//P,x
              false, "", 0);
          updatepoint(gasPoint, false, 0,  //T
              false, 0, false, 0,//P,x
              true, "setEpsi and calculate", Util.lit_d(outletEpsi_value.getText()));
          getPointProperties(gasPoint);//propriétés humides
          waterFlow=wq.waterFlow -(Wpoint-w_dicho)*flow_as;//débit d'eau restant
          //on écrit que l'enthalpie perdue par l'eau se retrouve dans l'air
          diff=wq.waterFlow*wq.waterH-hEau*waterFlow+flow_as*(qPrimeAmont-QPrimepoint);
          return diff;
      }
```

5) Ts étant déterminé, on modifie la composition du gaz humide en sortie

```
//modification de la composition du gaz
outletT=T;
updatepoint(gasPoint, false, 0,  //T
        false, 0, false, 0,//P,x
        true, "modHum", 0);
```

ANNEXE 6 : DIAGRAMMES UML DE CLASSES EXTERNES

Diagramme des classes de corps

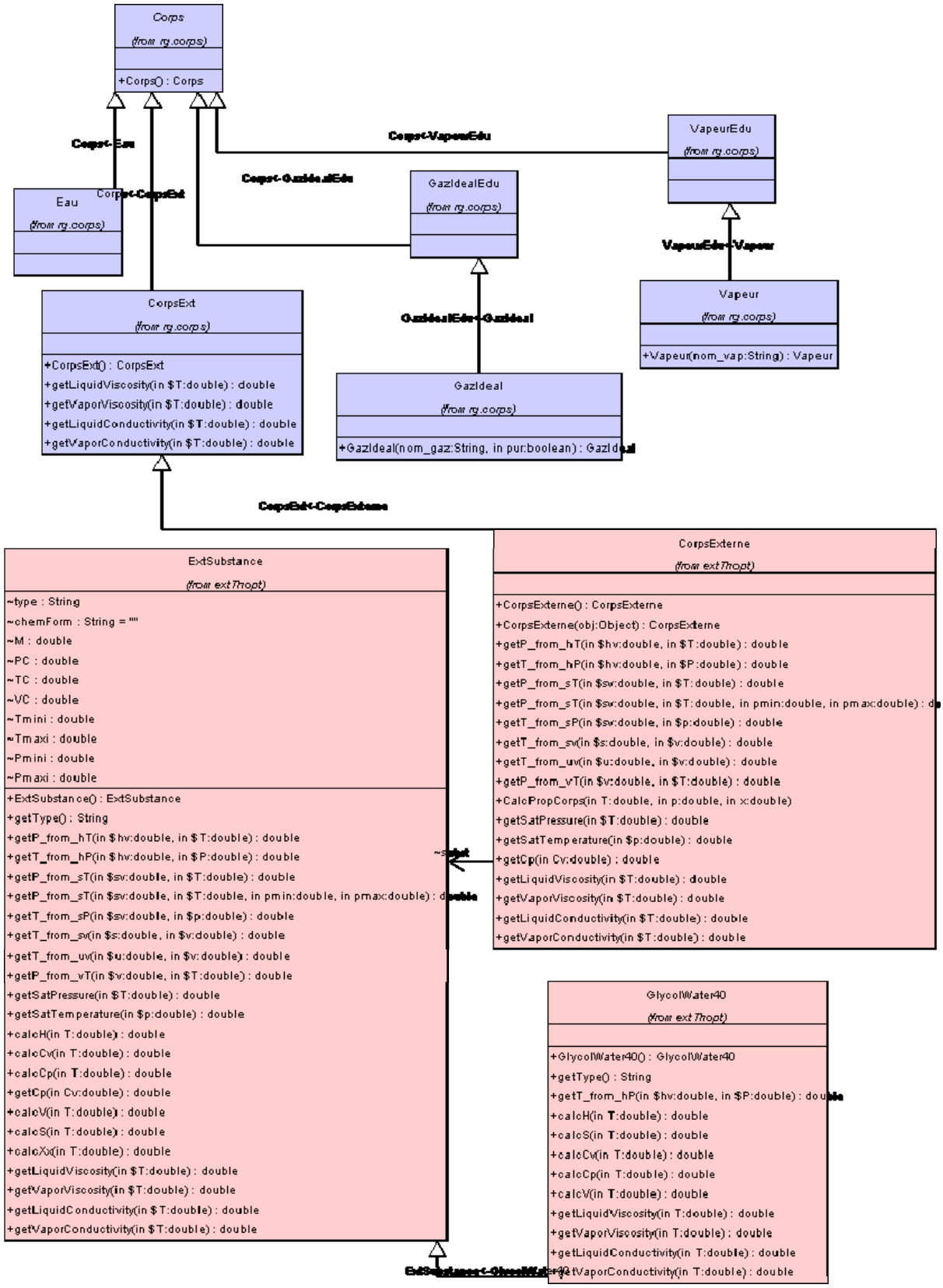


Diagramme des classes des composants externes